Tuning behavior-based robotics in mixed reality using unmanned aerial vehicles

Master's thesis in Systems, control and mechatronics

Shao-Hsuan Huang
Fengxiang Xue

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis in Systems, control and mechatronics

Tuning behavior-based robotics in mixed reality using unmanned aerial vehicles

Shao-Hsuan Huang
Fengxiang Xue

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Tuning behavior-based robotics in mixed reality using unmanned aerial vehicles
Shao-Hsuan Huang
Fengxiang Xue

Supervisor: Arion Pons, Department of Mechanics and Maritime Sciences
Examiner: Ola Benderius, Department of Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: An autonomous quadcopter UAV navigating and performing tasks in unknown environments without a prior map.

Typeset in LaTeX
Gothenburg, Sweden 2025

Tuning behavior-based robotics in mixed reality using unmanned aerial vehicles

Shao-Hsuan Huang
Fengxiang Xue
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

This thesis investigate a *behavior-based robotics* (BBR) approach for enabling autonomous *unmanned aerial vehicle* (UAV) navigation and obstacle avoidance in unknown environments without relying on global maps. The system leverages lightweight reactive control architectures, *finite state machine* (FSM) and *subsumption* architecture, to support real-time decision making based on local sensor feedback.

A *hardware-in-the-loop* (HIL) framework is employed to safely evaluate controller performance under realistic conditions. The HIL setup integrates the Crazyflie UAV platform, lighthouse-based state estimation, simulated sensor inputs, and a modular software stack based on microservice architecture.

To further enhance navigation efficiently, *genetic algorithm* (GA) optimization is applied to tune key controller parameters, including safe distances, tuning angles, and behavior transition timings. Experimental results demonstrate that both FSM-based and subsumption-based controllers enable robust mapless navigation and effective obstacle avoidance in complex indoor scenarios. The subsumption controller exhibits superior performance in cluttered environments, while the FSM controller performs better in open spaces.

The findings highlight the feasibility of behavior-based UAV control in GPS-denied, mapless indoor settings, and demonstrate the value of modular HIL testing for rapid prototyping and validation of autonomous navigation strategies.

Keywords: UAV, autonomous navigation, mapless navigation, FSM, subsumption architecture, HIL, obstacle avoidance, GA, indoor navigation, BBR.

Preface

This report presents the outcome of our master's thesis project carried out at the Department of Mechanics and Maritime Sciences at Chalmers University of Technology during the spring of 2025.
The project explored autonomous UAV navigation and obstacle avoidance using BBR control architecture within a HIL framework.
We would like to express our sincere gratitude to our examiner, Associate Professor Ola Benderius, for his invaluable guidance and support throughout the project. We also would like to thank our supervisor, Dr. Arion Pons, who kindly offers a lot of support and help throughout the project. We also thank teaching assistant Krister Blanch for his technical advice and encouragement during the development process. Finally, we would like to extend our appreciation to all those who provided support during this period.

Gothenburg, June 2025
Fengxiang Xue & Shao-Hsuan Huang

Contents

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

UAV	Unmanned aerial vehicles
SIL	Software-in-the-loop
DIL	Data-in-the-loop
HIL-rig	Hardware-in-the-loop
BBR	Behavior-based robotics
PFM	Potential field method
FSM	Finite state machine
EKF	Extended Kalman filter
PID	Proportional-integral-derivative
CI/CD	Continuous integration and continuous deployment
IMU	Inertial measurement unit
LPS	Lighthouse positioning system
CRTP	The Crazyflie real-time protocol
GA	Genetic algorithm

Nomenclature

Below is the nomenclature of parameters, and variables that have been used throughout this thesis.

Parameters

T_{task}	Total time taken to complete the task.
N_{stuck}	Number of times the UAV gets stuck during the task.
$N_{\text{overlimit}}$	Number of times the UAV exceeds operational or safety limits.
N_{timeout}	Number of times the task is terminated due to timeout.

Variables

α, β, γ	Penalty coefficients weighting the importance of each penalty term in the fitness function.

List of Figures

1
Introduction

This chapter introduces the concept of UAV, their common types of hardware and software components, and the core technologies that enable autonomous behavior. This chapter also outlines the motivation and focus of this thesis.

1.1 Overview of UAV Systems

An UAV, often referred to as a quadrotor, is an aircraft that operates without an onboard human pilot. A common type of UAV consists of a flight control board, four motors with propellers, a power source (battery), and supporting electrical connections. The system is capable of stable flight by continuously adjusting motor speeds to maintain balance and direction.
The autonomous system is suitable for UAV, since most UAV are designed to execute tasks are in the height and unknown environments, thus need to sense nearby environment, and react to the environment inputs. In this project, an autonomous UAV integrating multiple sensors and a controller architecture was implemented on real hardware. Based on predefined rules and parameters, the UAV was able to perform sequential tasks in a customized testing scenario.

1.2 Application and technology

UAVs have found widespread applications in surveillance, aerial inspection, environmental monitoring, search and rescue, and military operations. To operate autonomously, UAVs rely on a variety of sensors and perception systems, including cameras for visual input, range sensors for obstacle detection, and localization sys-

Figure 1.1: An example of a UAV.

tems for estimating their position in space. These inputs allow the onboard control software to perceive the environment, build a representation of its current state, and make decisions to execute tasks such as navigation, tracking, and avoidance. The control system must balance responsiveness, safety, and computational efficiency, especially in dynamic or unknown environments.
Various approaches have been proposed to model and control UAV movement—from classical control algorithms such as PID, to modern learning-based and behavior-based methods. The implementation in this thesis is focus on lightweight, reactive decision-making model enabling mapless navigation in an unknown environment.

1.3 Objective

The primary objective of this thesis is to explore a lightweight, modular behavior-based control approach that enables UAVs to perform mapless navigation and obstacle avoidance in unknown indoor environments. Instead of using advanced control methods such as model predictive control (MPC) or deep learning, which require significant computational resources, this study will emphasize local navigation and reactive control to ensure real-time responsiveness and practicality. To achieve this, the study implements and evaluates two distinct behavior-based controller architectures: FSM and subsumption. These architectures are particularly well-suited for reactive decision-making in dynamic, unmapped environments, where global path planning is impractical or unavailable.
To further improve controller performance, the GA is employed to optimize key behavior parameters in the FSM controller, such as safe distances, turning angles, and execution timings. The optimized parameter set is then directly applied to the controller without additional tuning, demonstrating the generalizability and effectiveness of the GA-tuned behavior parameters across different control architectures.
Testing and validation are conducted within a HIL simulation framework. This setup allows realistic interaction between simulated environments and physical UAV hardware, enabling safe and effective evaluation of behavior-based navigation strategies. CI/CD workflows are used to support modular software development and iterative testing. The entire control system is implemented using a distributed, microservices-based architecture built on OpenDLV, which facilitates flexible integration of perception, decision-making, and actuation modules.

1.4 Research questions

Based on the objectives outlined above, this thesis addresses the following research questions related to the use of BBR methods for autonomous UAV control in a simulated and semi-physical (HIL) environment.

1. Given the structure of HIL-rig, is it possible to control a UAV in the real world while its environment is simulated in the software? What performance can we observe?
2. Given that there are no localization and global mapping, can we control the Crazyflie with BBR, inside the HIL-rig process, to explore around without

crashing into obstacles and also go back to the charging pad before running out of the battery?
3. Giving the GA tuning on FSM, which controller architecture will have better performance?

1.5 Limitations

This section outlines the predefined limitations to the thesis.

- To maintain system simplicity and to evaluate the mapless navigation capabilities, this work does not generate a global map based on prior knowledge or perform global path planning. Instead, the BBR approach is used to search for locally valid paths, enabling obstacle avoidance and target reaching in real time.

- As the Crazyflie platform includes a built-in PID controller capable of executing basic motion commands (e.g., takeoff, landing, velocity control), this thesis does not re-implement or modify the low-level PID control layer.

- The system assumes the use of a LPS system for indoor localization; accordingly, all testing scenarios are conducted in indoor environments.

2
Background

This chapter introduces the prior knowledge and topics related to the research questions and details in the implementation.

2.1 UAV navigation and sensing

This thesis focuses on enabling autonomous UAV flight in unknown environments without prior maps. To achieve this, a reactive navigation approach is adopted. Instead of building an explicit map or using a global planner, the UAV responds directly to real-time sensor input such as infrared distance measurements and target detections. This approach aligns with the BBR theory, where local observations directly drive state transitions and action selection. Obstacle avoidance, in particular, is treated as a high-priority behavior that can override other tasks.

Autonomous In the context of autonomous, the perception and distance detector are vital for building the sensor input, Wang *et al.* [29] developed an indoor navigation system for quadrotors that fuses monocular vision with inertial data to achieve stable flight and obstacle avoidance in GPS-denied environments. This work shares a similar motivation of combining the camera and rangefinder to build local knowledge for navigation and obstacle avoidance in unknown indoor spaces.

Mapless navigation Mapless navigation allows UAVs to explore unknown or dynamic environments without relying on prior maps. Chang *et al.* [9] reviewed various mapless strategies, highlighting both classical control methods and learning-based approaches. Beyond UAVs, recent works have explored mapless navigation in other indoor robotic applications. For example, Kamthe *et al.* [15] proposed a mapless indoor navigation system for guiding visually impaired users, leveraging reactive obstacle avoidance and local path planning without constructing a global map. Such approaches further highlight the general applicability and effectiveness of reactive mapless navigation strategies in complex, unstructured environments.
In this work, a reactive navigation approach is adopted. Instead of building a full map, the UAV responds directly to real-time sensor input—such as infrared distance measurements and target detections—to make immediate motion decisions based on a dynamically updated local perception of the environment.

Sensor fusion for localization Sensor fusion remains a fundamental component for UAV navigation in unknown environments. Typical fusion schemes combine IMU

data with optical or visual sensors to estimate UAV state. Mac *et al.* [20] proposed a position-estimation method by fusing IMU and optical sensor to get UAV's position and orientation in an partly unknown environment, which is a low-cost and effective way to realize the unknown indoor localization. Huang *et al.* [12]integrated sensor fusion within an RL-based navigation framework, highlighting its importance for providing robust state estimation inputs to downstream decision-making modules.

PFM and Reactive obstacle avoidance strategy The PFM is widely used for real-time obstacle avoidance in mobile robotics and UAVs. The core concept, first introduced as *artificial potential field* by Khatib [16], models navigation using an artificial potential field composed of two components: an attractive field pulling the robot toward the goal, and a repulsive field pushing it away from obstacles. The robot follows the resultant force vector derived from these fields.
Over the years, various enhancements to the original PFM have been proposed to improve performance in complex environments and mitigate issues such as local minima. Tang *et al.* [27] introduced a novel potential field formulation to address unreachable goals and improve obstacle avoidance behavior. Zhou and Li [19] explored adaptive potential field methods to improve path planning flexibility. Li *et al.* [18] proposed a hybrid approach combining artificial potential fields with Dubins path planning for coordinated multi-UAV obstacle avoidance. Vision-based obstacle avoidance strategies have also drawn inspiration from PFM concepts. For example, optical flow combined with fuzzy control has been used to implement reactive obstacle avoidance behaviors in lightweight UAV platforms [30]. These methods leverage the idea that increasing perceived motion or proximity triggers stronger evasive actions, an approach conceptually similar to the repulsive component of artificial potential fields.
In this work, the UAV's obstacle avoidance logic is similarly inspired by the repulsive field principle of PFM. The system uses a rangefinder to continuously monitor the distance to obstacles in various directions. When an obstacle is detected within a predefined safety threshold, the UAV immediately executes an evasive maneuver, such as lateral movement, backward motion, or ascent, toward a direction with greater clearance. Unlike classical PFM or its modern variants, our implementation does not compute a continuous resultant force vector from combined attractive and repulsive fields. Instead, it employs discrete, rule-based decision logic triggered by sensor thresholds. This design choice prioritizes computational efficiency and real-time responsiveness, making it particularly well-suited for embedded UAV platforms with limited onboard processing resources. While this reactive approach may not handle complex local minima scenarios as effectively as full PFM-based planners, it offers a robust and lightweight solution for high-speed, mapless navigation in cluttered indoor environments.

2.2 UAV control architectures

Decision-making in autonomous systems refers to the computational process by which a robot or UAV selects actions based on sensor inputs, internal states, and

predefined goals. In unknown, dynamic environments, designing lightweight and reactive decision logic is critical for ensuring robust navigation and obstacle avoidance. Behavior-based architectures provide an effective framework for such decision-making [8]. They enable modular and flexible control by dynamically switching between behaviors in response to real-time sensor inputs. This section reviews two commonly used behavior-based control architectures—FSM and ubsumption—that are particularly suitable for mapless UAV navigation in indoor environments.

FSM FSMs are a widely adopted mechanism for implementing behavior-based control [26]. In FSM-based systems, robot behaviors are encoded as discrete states, with explicit transition rules dictating when and how the system switches between behaviors. This structure offers clear logic flow, ease of debugging, and modularity, making it well-suited for UAV applications requiring reliable navigation and collision avoidance. FSMs have been applied in diverse robotic domains, from legged robots [26] to autonomous vehicles [14], demonstrating their flexibility and effectiveness.

Subsumption architecture Subsumption architecture, introduced by Brooks and later extended in various implementations [8], offers an alternative behavior-based control model emphasizing prioritized behavior layers. In this framework, high-priority behaviors (such as emergency obstacle avoidance) can preempt or suppress lower-priority behaviors (such as goal following), allowing the robot to react appropriately to changing environmental conditions. Subsumption architecture has been effectively applied to mobile robot navigation and is well-suited for UAVs operating in complex, cluttered environments where rapid reactive control is critical. Recent work by Xu [31] further explored behavior programming for UAV navigation, emphasizing modular behavior design and dynamic adjustment. These approaches validate the applicability of behavior-based control paradigms—including both FSMs and Subsumption architectures—for autonomous UAV navigation in indoor, mapless settings.
This thesis builds upon these principles by implementing and evaluating both FSM-based and subsumption-based control architectures separately. The comparative analysis of these two behavior-based strategies provides insights into their respective strengths and suitability for indoor UAV navigation without global maps.

2.3 Evolutionary optimization in robotics

Optimizing controller parameters is crucial for improving the performance of autonomous robotic systems. Various studies have demonstrated the effectiveness of GA in this domain. Vincenzo *et al.* [10] proposed a hybrid control design using GA to tune the weights of a *robust linear quadratic regulator* (RSLQR) controller for UAV flight control. Meng *et al.* [21] applied GA to optimize PID controller parameters for quadruped soft robots, while Vu *et al.* [28] formulated PID tuning as a GA-based optimization problem.

GA GA is widely used for optimization in robotics and control applications, particularly well-suited for tuning parameters in behavior-based control systems. Inspired by natural selection, GAs evolve populations of candidate solutions through selection, crossover, and mutation to progressively improve system performance.
In the context of behavior-based robotics, GAs have been successfully applied to optimize controller parameters and behavior switching logic. Nantogma *et al.* [22] proposed a behavior-based genetic fuzzy control system for cooperative *unmanned surface vehicles* (USVs), demonstrating the adaptability of GA to behavior-layer optimization problems. Hristov and Stefanov [11] explored the use of GA for optimal coding of FSM states, highlighting its effectiveness in improving state-based control architectures. More broadly, evolutionary optimization techniques such as GA and artificial bee colony algorithms have also been applied to path planning and navigation tasks. For example, Li and Wu [17] employed an artificial bee colony algorithm for UAV path replanning under dynamic conditions, underscoring the versatility of evolutionary methods in adaptive navigation.

2.4 HIL testing

Testing UAV control systems directly in physical environments is often costly and prone to hardware failure, particularly when tuning reactive behaviors or testing in complex indoor scenarios. To mitigate these risks, modular simulation frameworks such as SIL or HIL testing have become standard practice in UAV development.

SIL Lepejit *et al.* [24] present a flexible simulation framework that supports both SIL and HIL testing. In SIL setups, the UAV software, including navigation, planning, and control modules, executes entirely within a virtual environment, allowing for rapid iteration and debugging without risk to physical hardware. Such frameworks typically require minimal modification to localization and control interfaces, enabling easy transition between simulation and real-world deployment. They are particularly well-suited for testing complex mission logic and behavior tuning in controlled conditions.

HIL HIL simulation provides a closer approximation of real-world performance by integrating physical hardware components into the testing loop. In HIL setups, the UAV control software interacts with actual hardware—such as flight controllers or sensors—while the environment and dynamics are simulated. Modular HIL frameworks enable safe and flexible testing of UAV control architectures [23]. HIL-based simulation has been widely used for UAV behavior tuning and navigation algorithm validation [25]. In particular, for complex exploration tasks in risky environments, HIL tuning provides a critical step toward real-world deployment [1]. In this thesis, a similar HIL-based setup is adopted to evaluate the performance of behaviour-based UAV control architectures and optimize controller parameters via genetic algorithms.

3

Methods

This chapter describes the experimental setup and methodology, following the logic illustrated in Figure 3.1. The structure outlines the implementation of Crazyflie control within the HIL framework, covering the following key components:

- Hardware and sensors: The Crazyflie platform equipped with onboard sensors and a state estimator (based on LPS) provides real-time state feedback.
- System architecture: Mircroservices used in this projects and virtual map environment construction.
- Behavior-based controllers: Implementation of behavior-based control architectures—FSM and subsumption—for high-level decision making. These behavior-based controllers perform high-level decision making, while interfacing with the Crazyflie's built-in PID controller for low-level motion execution.
- Parameter optimization: Use of GA to optimize controller parameters, improving navigation efficiency and obstacle avoidance performance.
- The real world environment: A quick view of the experiment site and equipments.

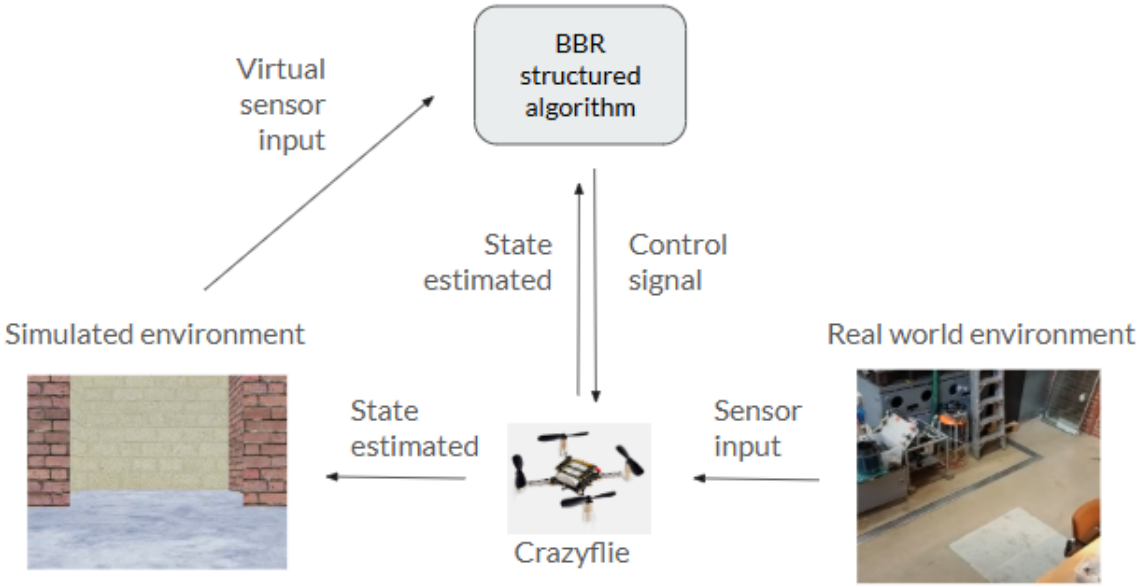


Figure 3.1: The structure description of HIL implemented in the thesis.

FSM or subsumption controller continuously receives the current state of Crazyflie (including position, velocity, and yaw angle) from the onboard state estimator.

Based on the estimated state and virtual sensor input, FSM and subsumption controller computes the next behavior to react, such as finding a new target, which needs calculating the heading angle and distance to the desired target position(setpoint). These data are then sent to Crazyflie via the wireless communication link. Crazyflie's onboard PID controller executes these setpoints, enabling the drone to perform complex behaviors such as navigation and obstacle avoidance.

3.1 Crazyflie

This thesis work is implemented using Crazyflie platform, which provides an accessible and modular development environment, and can simplify development by allowing high-level commands to be executed with minimal effort. With integrated components such as onboard sensors and mounted decks, it offers a complete hardware stack for autonomous flight experiments.

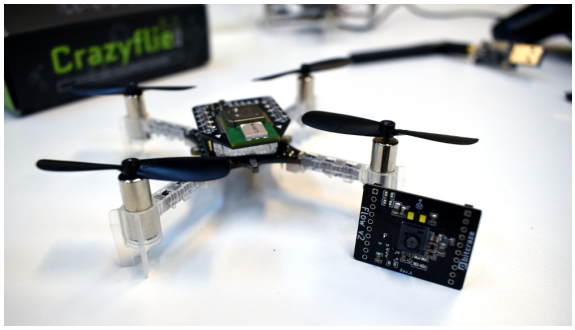


Figure 3.2: Illustration of the Crazyflie drone.

IMU The IMU consists of a gyroscope that measures the angular velocities of roll, pitch, yaw direction, and an accelerometer to provide linear acceleration along x,y,z directions. These sensor readings are fused using a complementary filter to estimate the drone's roll, pitch, and yaw angles. In our experiment, only yaw is used, as the heading direction of Crazyflie, while roll and pitch are primarily used for flight stabilization.

LPS The lighthouse positioning system provides high-precision, GPS-free indoor localization for Crazyflie. It utilizes multiple SteamVR Base Stations as optical beacons, each emitting rotating laser sweeps. Crazyflie is equipped with a dedicated LPS deck, which detects these laser signals and calculates its global 3D position (x, y, z) using triangulation algorithms. With centimeter-level accuracy, LPS enables robust autonomous navigation and real-time localization in complex indoor environments.

State estimator Crazyflie uses an onboard EKF to output higher precision state estimation. The EKF fuses high-frequency IMU data with absolute position measurements from the LPS,which removes drift from IMU and improves the positioning

accuracy by LPS position corrections. The output of this EKF is global positions x,y,z, and orientation angles:yaw , roll and pitch. In this experiment, only the x,y,z coordinates and the yaw angle are used to control Crazyflie's 2D movements in the 3D simulation environment.

Crazyflie real-time protocol The *Crazyflie real-time protocol* (CRTP)[7] is a wireless communication protocol specifically developed for Crazyflie platform. CRTP enables real-time bidirectional communication between Crazyflie and the host computer by Crazyflie Dongle. The reference link could be found on GitHub[13] repository.

3.2 System architecture

The software experiments in this thesis were conducted on *OpenDLV*[5] platform, a microservice-based framework originally developed for autonomous vehicle research. The system was implemented as a set of containerized microservices running under `Docker`, enabling modular development, flexible deployment, and reproducible experiments.
This section first introduces the OpenDLV architecture and the role of the message interface and middleware. Then, it details the microservices designed and implemented for this work, explaining how they interact to form the complete control loop for Crazyflie navigation and obstacle avoidance in the HIL framework.

OpenDLV and Libcluon OpenDLV[5] is a modern software framework, designed specifically for cyber-physical systems, robots and the Internet of Things (IoT). It is the first microservice-based software architecture specifically for the development of autonomous vehicles originating from 2015, and it is free and open source. Inter-service communication within the microservice architecture is handled by Libcluon[6], a single-file, header-only C++ library allowing lightweight message exchange between microservices via UDP, TCP, or shared memory. Libcluon adopts a publish/subscribe model, where data serialization and message definitions are specified in the *OpenDLV Standard Message Set* [2]. For communication between Crazyflie hardware and host computer, the system relies on CRTP, while Libcluon remains the backbone for inter-process communication in the simulation and software-in-the-loop environment.

Message interface The system employs the OpenDLV standard message set, a standardized message definition file used in the OpenDLV framework. It defines a comprehensive set of message types for exchanging data between distributed microservices, covering vehicle state, sensor data, control commands, and simulation interfaces. The .odvd file defines a unified set of messages for inter-service communication. In this thesis, key messages include `opendlv::logic.action.CrazyFlieCommand` for UAV control commands, `opendlv::proxy::DistanceReading` for range sensor

outputs, and `opendlv::logic.sensation.CrazyFlieState` for state estimation. This standardization ensures interoperability between independently developed microservices and enables modular system integration across simulation and real-world environments. By adhering to the OpenDLV standard message set, the system ensures interoperability and modularity: each microservice can be developed, tested, and replaced independently while maintaining seamless integration with the rest of the system.

Microservices in this project In this work , different sensor modules are implemented as individual microservices. Each microservice is packaged as a `Docker` image and run as a container with its dependencies and configuration. The modularity of microservices makes them well-suited for use with BBR controllers, as each function can be developed, deployed, and scaled independently.

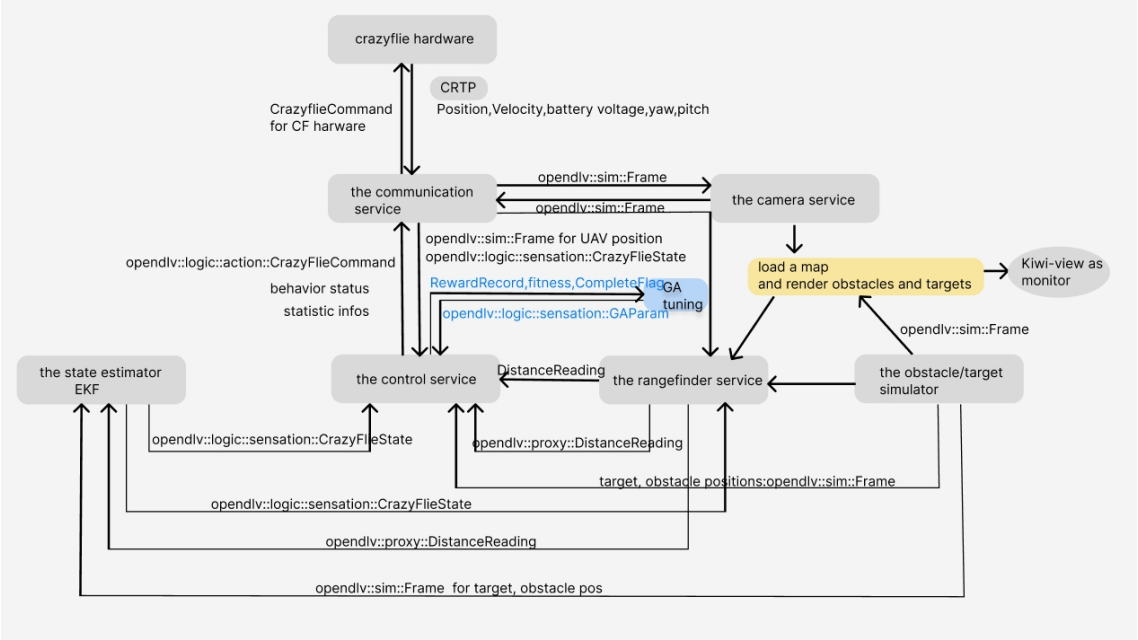


Figure 3.3: The microservice system architecture.

Virtual rangefinder service The rangefinder microservice is responsible for simulating distance measurements between Crazyflie and static obstacles (such as walls) in the environment. Its core computational steps are as follows:

- Loading the environment map: The service loads a map file specified by the `map-file` parameter, where obstacles (e.g., walls) are defined as a set of line segments.
- Receiving Crazyflie pose: The rangefinder subscribes to `opendlv::sim::Frame` messages to obtain the current position and orientation of the Crazyflie in real time.
- Ray casting and geometric computation: At each simulation cycle (with a typical frequency specified by the `freq` parameter, e.g., 10 Hz), the rangefinder emits a virtual ray from the Crazyflie's current position in the direction of the sensor's orientation. It then computes the intersection points between this ray

and all obstacle line segments in the map, selecting the shortest intersection distance as the current sensor reading.
- Outputting distance measurements: The computed distance is published as an `opendlv::proxy::DistanceReading` message, which can be consumed by other microservices such as the control service and the state estimator.

This process enables the rangefinder to provide realistic, real-time distance measurements between the Crazyflie and environmental obstacles, supporting tasks such as obstacle avoidance and navigation.

Camera simulation service The Camera simulation service [4] emulates an onboard camera for Crazyflie, generating visual data for perception and navigation tasks. Visual output is rendered efficiently using OpenGL and made accessible via shared memory, enabling other services to perform operations such as target recognition or scene mapping. The camera's position and orientation are continuously synchronized with the drone state through the communication service, ensuring realistic and up-to-date visual input throughout the simulation.

Target and dynamic obstacle simulation service This service generates a moving obstacles in each map, assigns it a predefined horizontal trajectory, and broadcasts its real-time position to other components using Libcluon messages. The dynamic object moves back and forth at a constant speed of 0.1 m/s, providing a repeatable scenario for testing perception and obstacle avoidance algorithms. The simulator performs the following operations:

- **Crazyflie state reception:** The service subscribes to `opendlv::sim::Frame` messages to obtain the Crazyflie's current position (x_{cf}, y_{cf}), which is used to evaluate proximity to targets.
- **Target generation and switching:** Depending on the map type (`rooms` or `maze`), one or more target positions (x_t, y_t) are initialized. During runtime, the Crazyflie-to-target distance is computed as:

  $$dist = \sqrt{(x_{cf} - x_t)^2 + (y_{cf} - y_t)^2}$$

  When the distance falls below a predefined threshold (e.g., $0.3m$), the simulator triggers a target switching logic, updating the target's location to simulate its disappearance, relocation, or progression to the next goal. The number of successfully reached targets is recorded via a counter (n_{TF}) for task statistics and simulation completion checks.
- **Dynamic obstacle / target motion:** To simulate moving obstacles or dynamic targets, the simulator updates their positions in real time. For example, a simple linear motion along the x-axis is implemented using a velocity parameter and current position, optionally adapting the motion based on proximity to the Crazyflie or other conditions.
- **Output signals:** The current positions of both static and dynamic obstacles and targets are broadcast via `opendlv::sim::Frame` messages, which are consumed by the rangefinder service, camera service, and control service. Additionally, the simulator publishes `opendlv::TargetFoundState` messages to communicate task progress (i.e., number of targets found).

State estimator service This service uses the OpenCV library to process camera images, detecting targets and obstacles, and calculates relative distance and direction to the targets within the visual field. The calculation of distance and angle in the image space is performed using the pixel coordinates of the target and the image center are extracted from the processed camera frame.

$$d_{\text{img}} = \sqrt{(x_{\text{i,c}} - x_{\text{i,t}})^2 + (y_{\text{i,c}} - y_{\text{i,t}})^2} \tag{3.1}$$

$$\theta_{\text{img}} = \arctan\left(\frac{x_{\text{i,c}} - x_{\text{i,t}}}{y_{\text{i,c}} - y_{\text{i,t}}}\right) \tag{3.2}$$

where $(x_{\text{i,c}},\ y_{\text{i,c}})$ represents the pixel coordinates of the image center, and $(x_{\text{i,t}},\ y_{\text{i,t}})$ denotes the pixel coordinates of the target in the image. These measurements represent the perceived position and angle on the camera view, not the true metric distance, and are transmitted via Libcluon messages for use by the control service in navigation and obstacle avoidance.

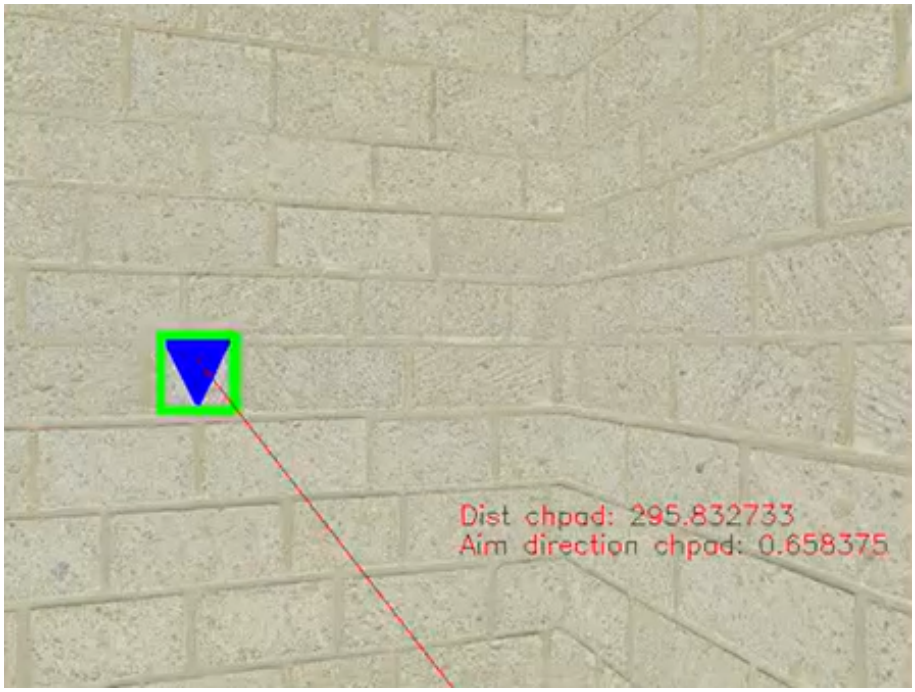


Figure 3.4: An example of measured pixel distance and direction of charging pad.

In addition to perception, the state estimator also monitors overall system performance. It collects Crazyflie's true position from the communication service and receives dynamic obstacle positions from the obstacle simulation service.

$$d = \sqrt{(x_{\text{cf}} - x_{\text{obs}})^2 + (y_{\text{cf}} - y_{\text{obs}})^2} \tag{3.3}$$

$$\theta = \arctan\left(\frac{y_{\text{obs}} - y_{\text{cf}}}{x_{\text{obs}} - x_{\text{cf}}}\right) \tag{3.4}$$

where $(x_{\text{cf}},\ y_{\text{cf}})$ denotes the true position of the Crazyflie, and $(x_{\text{obs}},\ y_{\text{obs}})$ denotes the position of the obstacle. By comparing these positions, it can determine whether

a collision has occurred and thus evaluate the effectiveness of the BBR control algorithm.

Crazyflie communication service This service manages all communication between Crazyflie and other microservices using the CRTP protocol. It receives motion commands from the control service (via Libcluon messages) and transmits them to the drone, while also retrieving state information such as position, altitude, and yaw. These states are made available to the camera simulation service for visual synchronization and to the control service for behavior selection based on the current yaw angle. In this setup, only the yaw angle is accessed by the control logic, reflecting the focus on reactive behavior rather than global optimization. Additionally, battery status information is also communicated, allowing the system to determine when the drone should return to the charging pad before the battery is depleted.

Control service The control service acts as the central behavior arbitrator, implementing FSM and subsumption controllers to map sensor inputs to appropriate drone actions. It receives real-time data from the virtual rangefinder (distances to walls and obstacles), the state estimator (distance and direction to targets, charging pad, and dynamic obstacles), and the Crazyflie communication service (current yaw angle and battery status). Based on these inputs and a predefined priority structure, the controller dynamically selects the most suitable behavior for Crazyflie.

Simulated Environment

The camera service supports customizable simulation environments, including walls, floors, ceilings, charging pads, fixed targets, and static obstacles of varying sizes. Simulation experiments were conducted in two maps, each measuring three by three by two meters. A four-room layout and a maze with indirect corridors (see Figure 3.5 and Figure 3.6). In both maps, the drone must visit all targets (green triangles) and return to the charging pad (blue triangle) while avoiding dynamic obstacles (purple sphere) and preventing collisions. Target locations appear sequentially in the four-room map and are distributed throughout the corridors in the maze, resulting in different levels of navigation complexity.

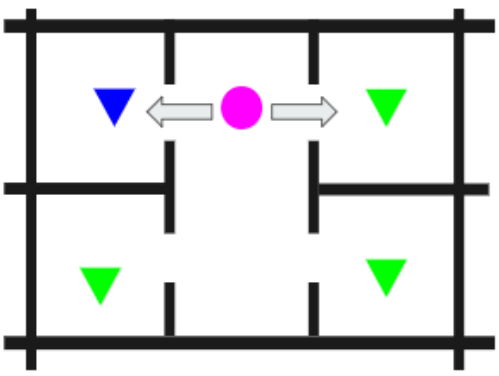


Figure 3.5: Map of four rooms.

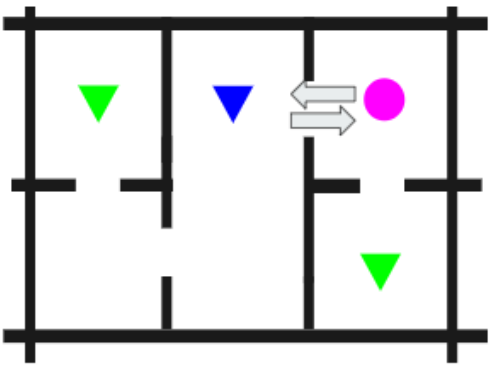


Figure 3.6: Map of a maze.

3.3 Behavior-based controller

This section introduces the controller architecture,and the detailed behavior blocks forming the architectures.The high-level BBR controllers are developed above the PID low-level controller. Virtual sensors and layered control logic interact with the real Crazyflie hardware through the CRTP protocol, enabling simulation and real-time HIL operation.

PID controller The onboard PID controller computes errors between the current state and the setpoints, then generates actuator commands for the four motors to maintain stable flight and follow high-level commands. Altitude, position, and attitude information can be provided by any sensor source fused by the estimator, ensuring modularity and hardware flexibility. This onboard controller enables developers to issue high-level commands—such as "go to position" or "change yaw"—without interacting directly with motor-level control. Its robust and modular design makes it suitable for both real hardware and simulation environments.

3.3.1 BBR controller structure

The high-level controller in this work adopts a reactive, hierarchical structure based on FSM and subsumption architecture. Both approaches implement a set of modular atomic behaviors, directly mapping sensory inputs to actions without the need for global environment modeling. The FSM controller consists of discrete states, with transitions determined by sensor-triggered conditions. At any given time, the system occupies a single state, and transitions occur when specific criteria are met based on current sensor readings. In implementation, the FSM decision logic monitors sensor inputs and executes the corresponding behavior for the active state. When transition conditions are satisfied, the system exits the current state and enters a new behavior module.

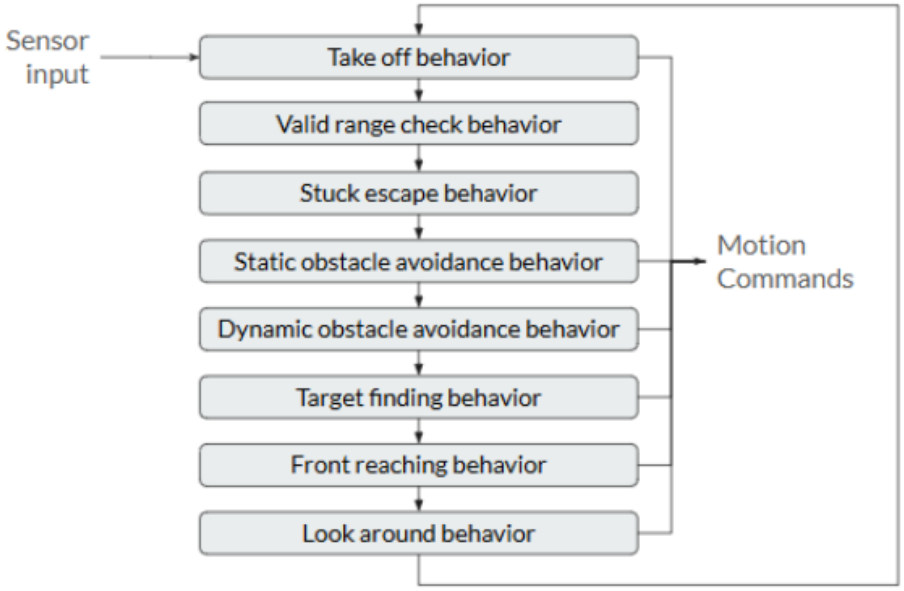


Figure 3.7: The FSM controller structure.

In contrast, the subsumption controller organizes behaviors in parallel, each capable of responding to relevant sensory inputs. An arbitrator enforces a strict priority scheme: higher-priority behaviors can suppress or override lower-priority behaviors, ensuring that safety-critical or survival-related actions (such as obstacle avoidance) always take precedence. This approach provides greater flexibility and robustness compared to the strictly sequential nature of FSMs.

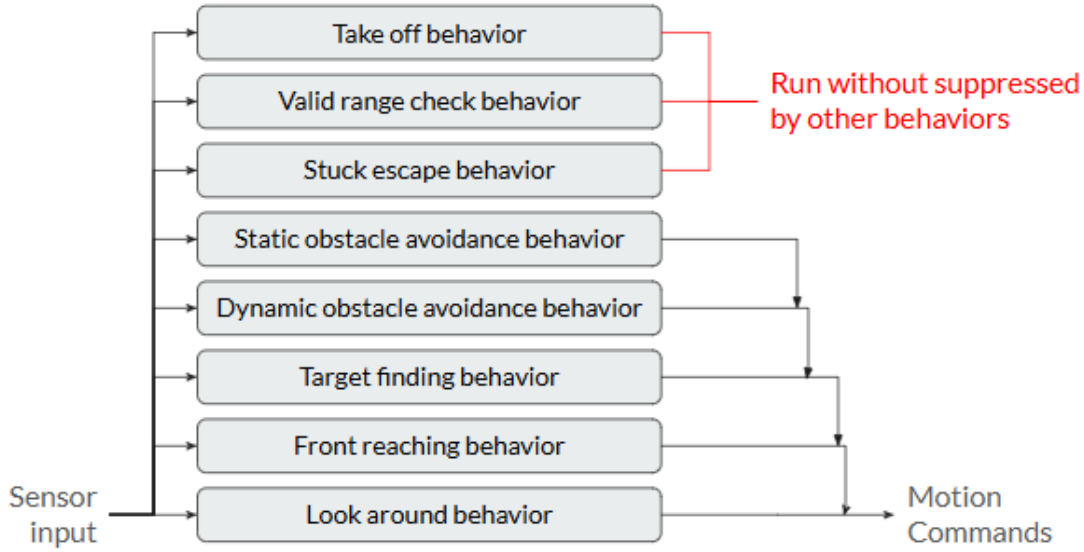


Figure 3.8: The Subsumption controller structure.

Both architectures utilize the following sensor inputs:

- Virtual rangefinder (distances to walls and static obstacles)
- Target detection and valid path checking

- State estimator (locations of the charging pad and dynamic obstacles)
- Crazyflie communication service (current yaw angle and battery state)

3.3.2 Behavior blocks

This part presents the atomic behaviors used in the FSM and Subsumption controllers, including behaviors: take off, check valid path, escape stuck state, avoid static obstacle, avoid dynamic obstacle, find target, reach front, and look around.

Take off behavior Before any mission can begin, the Crazyflie must take off. This behavior performs a simple takeoff maneuver and notifies the rest of the system that the drone has taken-off.

Algorithm 1 Take off behavior

1: **if** Battery state≥3.8V **then**
2: Take off to 1.5 meter upon the floor.
3: **end if**

Check valid path behavior This behavior traverses all possible heading angles to determine whether there exist accessible paths that are free of obstacles. If such paths are found, the algorithm selects the optimal one—typically the direction with the longest clear distance. If no valid path is available, Crazyflie will re-enter the look around behavior to search for new possibilities. The valid path check is activated after the target finding and look around behaviors, and it determines whether the Crazyflie can proceed to the reach target behavior. To assess whether a path is obstacle-free, the algorithm ensures that no obstacle detected at any other angle projects into the candidate direction within a threshold distance of 0.1 meters. In other words, as illustrated in figure 3.9, a path is considered clear and safe only if the perpendicular distance from any obstacle to the intended flight line exceeds 0.1 meters. By comparing all possible directions, the algorithm selects the one with the maximum clear distance as the best candidate for subsequent maneuvers, such as dodging to the left, right, or rear.

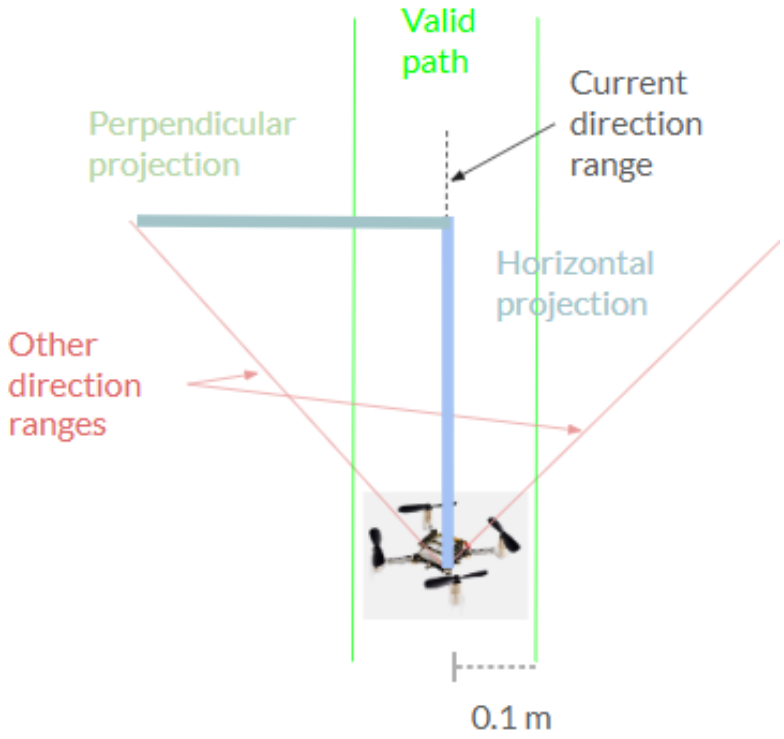


Figure 3.9: The explanation of a valid path.

Algorithm 2 Check valid path behavior

```
if Direction ranges are available then
    for each candidate direction (rear, left, right) do
        Evaluate the clear distance in this direction
        Ensure no obstacle projects into this direction within the safety threshold
        Record the clear distance for this direction
    end for
    Select the direction with the maximum clear distance as the best path
else
    No valid path found; trigger Look Around behavior
end if
```

Escape stuck behavior During experiments, the drone occasionally becomes stuck while executing behaviors such as reach front, find target, or look around. The escape stuck behavior is designed to detect such situations, that is, when forward distance does not change for over 10 seconds, this behavior will interrupt the current state and reenter the reach front, find target, or look around behaviors. Stuck conditions typically arise when the sensor update rate fails to keep pace with the controller's action rate.

Algorithm 3 Escape stuck behavior

```
if No significant forward movement detected then
    Increment stuck counter by one time step
else
    Reset stuck counter to zero
end if
if Stuck counter ≥ 10 seconds then
    Interrupt current behavior
    Trigger a restart of find target, reach front, or look around behavior as appropriate
    Reset stuck counter to zero
end if
```

Avoid Static obstacle behavior The static obstacle avoidance behavior is designed to ensure the safety of the Crazyflie when encountering stationary obstacles such as walls or columns. This behavior continuously monitors the distance to obstacles in all directions using onboard range sensors. If an obstacle is detected within a predefined safety threshold (e.g., 0.08 m for critical proximity, 0.25 m for the front, and 0.1 m for the sides), Crazyflie immediately interrupts its current action and executes an avoidance maneuver. Specifically, when a static obstacle is detected within the critical distance in the front, left, right, or rear direction, the drone will either stop its current motion or perform a lateral or backward movement to evade the obstacle, depending on the available free space. The avoidance action is maintained until the drone is no longer within the unsafe proximity of the obstacle. Once Crazyflie has successfully avoided the obstacle, it resumes its previous behavior, such as find target or follow path. This behavior module operates with high priority in the control architecture, ensuring that obstacle avoidance actions preempt other behaviors to prevent collisions and maintain safe operation in cluttered environments.

Algorithm 4 Avoid static obstacle behavior

if Front valid path $\leq$ danger distance **then**
    Interrupt and trigger a restart of find target, reach front, and look around behaviors
    if Rear valid path $\geq$ dodge distance **then**
        Move rearward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
else if Front valid path $\leq$ safe distance **then**
    Stop all current motion commands
    Instruct Crazyflie to exit reach front behavior
end if
if Left valid path $\leq$ danger distance **then**
    Interrupt and trigger a restart of relevant behaviors
    if Right valid path $\geq$ dodge distance **then**
        Move rightward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
else if Left valid path $\leq$ safe distance **then**
    Compare current and previous left wall distance
    Notify relevant behaviors of the interruption
    if Getting closer to the left wall **and** right valid path $\geq$ dodge distance **then**
        Move rightward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
end if

Algorithm 5 Avoid static obstacle behavior (continued)

if Right valid path $\leq$ danger distance **then**
    Interrupt and trigger a restart of relevant behaviors
    if Left valid path $\geq$ dodge distance **then**
        Move leftward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
else if Right valid path $\leq$ safe distance **then**
    Compare current and previous right wall distance
    Notify relevant behaviors of the interruption
    if Getting closer to the right wall **and** left valid path $\geq$ dodge distance **then**
        Move leftward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
end if
if Rear valid path $\leq$ danger distance **then**
    Interrupt and trigger a restart of relevant behaviors
    if Front valid path $\geq$ dodge distance **then**
        Move forward by the predefined dodge distance
    else
        Stop all current motion commands
    end if
else if Rear valid path $\leq$ safe distance **and** currently dodging dynamic obstacles **then**
    Stop all current motion commands
    Instruct *Crazyflie* to stop rearward dodging in avoid dynamic obstacle behavior
end if

Avoid dynamic obstacle behavior The avoid dynamic obstacle behavior is designed to enable the drone to safely navigate in environments containing moving obstacles. In this experiment, the dynamic obstacle is represented by a moving purple ball in the map. This behavior continuously monitors the relative position and velocity of dynamic obstacles using state estimator, virtual camera and range finder. When a dynamic obstacle is detected within a predefined safety threshold and is on a potential collision course with Crazyflie, the system evaluates possible evasive maneuvers. Depending on the direction and proximity of the approaching obstacle, the drone may execute lateral (left or right), rearward, or upward dodging maneuvers to avoid collision. The selection of the evasive direction is based on the valid path of Crazyflie. During the avoidance maneuver, the drone temporarily suspends its current task and prioritizes collision avoidance. Once the dynamic obstacle is no longer within the critical range, Crazyflie resumes its previous behavior, such as reach target or follow path. This behavior module operates with high priority in the control architecture, ensuring that avoid dynamic obstacle actions preempt other behaviors to maintain safe and robust operation in dynamic and unpredictable environments.

Algorithm 6 Avoid dynamic obstacle behavior

```
1: if Obstacle distance ≤ danger distance then
2:     if Obstacle is on the left then
3:         if Left valid path ≥ dodge distance then
4:             Execute leftward evasive maneuver (dodge distance)
5:         else if Rear valid path ≥ dodge distance then
6:             Execute rearward evasive maneuver (dodge distance)
7:         else
8:             Halt current action
9:         end if
10:    else                                          ▷ Obstacle is on the right
11:        if Right valid path ≥ dodge distance then
12:            Execute rightward evasive maneuver (dodge distance)
13:        else if Rear valid path ≥ dodge distance then
14:            Execute rearward evasive maneuver (dodge distance)
15:        else
16:            Halt current action
17:        end if
18:    end if
19:    Interrupt and trigger a restart of find target, reach front, and look around
   behaviors
20: else if Obstacle distance ≤ safe distance then
21:    Halt current action
22:    Interrupt and trigger a restart of affected behaviors
23: else
24:    If previously dodged, return to original position
25:    Resume previous behavior
26: end if
```

Find target behavior The find target behavior is responsible for orienting the Crazyflie towards a designated target, such as a green ball or a charging pad, and determining a clear path for approach. This process begins by checking if a valid target exists and whether Crazyflie has a direct, unobstructed path to it. If a target is detected and the path is not clear, Crazyflie initiates a turning maneuver to align itself with the target direction. During this maneuver, Crazyflie records distance measurements at various angles to map the surrounding environment. The drone then analyzes these measurements to identify an angle with sufficient clearance, ensuring that no obstacles block the path to the target. If such a direction is found, Crazyflie turns to face it and prepares to move forward. If the direct path remains obstructed, Crazyflie continues to search for alternative directions by further turning and re-evaluating the environment. This iterative process continues until a viable path is found or the Crazyflie determines that no safe route is available, in which case it may initiate a look-around or obstacle avoidance behavior. Throughout the target finding process, Crazyflie maintains robust handling of interruptions, such as dynamic obstacles or unexpected environmental changes, by resetting its state and reattempting the alignment and path-finding steps as necessary. This ensures reliable and adaptive navigation towards the target in complex and dynamic environments.

Algorithm 7 Find target behavior

1: **if** Need to redo **then**
2: Escape any action in finding the target and redo the finding process.
3: **else if** An interruption occurs **then**
4: Try to continue the interrupted action.
5: **end if**
6: **if** A target appears in sight **then**
7: Try to turn to the target for set finding angle, and record angle in the mean time.
8: Use the aim direction of the target to record target's true direction angle.
9: Stop the turning after Crazyflie has pointed to the target or the set finding angle has been reached.
10: Find the valid path direction closed to the target direction angle the most.
11: Turn to the target direction angle.
12: Tell reach front behavior that Crazyflie is ready to go to that path.
13: **end if**

Reach front behavior The reach front behavior enables the Crazyflie to perform a controlled forward movement while continuously ensuring flight safety. When the behavior is initiated, the system first checks whether the previous front reaching process has been interrupted or needs to restart. In such cases, any ongoing forward motion is aborted, and the reaching process is reinitialized. Before proceeding, the front range sensor is evaluated to ensure that no obstacle exists within a predefined safety threshold; if an obstacle is detected, Crazyflie halts its movement to prevent collision. If a valid path toward the target is available, the system determines the

appropriate forward distance based on the source of the path. When the path is derived from the target finding behavior, the forward distance is set to a predefined distance, whereas if it originates from the look around behavior, a different distance for look around is applied. After the distance is determined, the Crazyflie begins moving forward along the specified path. During the forward motion, the system continuously monitors the front range sensor to detect any obstacles in real time. The motion continues until either the specified forward distance is reached or an obstacle is detected ahead. Once either condition is met, the drone halts its forward motion and terminates the behavior, allowing for a smooth transition to subsequent actions or decision processes.

Algorithm 8 Reach front behavior

```
1: if Reach front is interrupted or needs to restart then
2:     Abort current forward motion and reinitialize reaching process
3: else if Front range sensor detects obstacle within safety threshold then
4:     Halt current flying action
5: end if
6: if A valid path to the target is available then
7:     if Path is obtained from Target Finding then
8:         Set forward distance to predefined target finding distance
9:     else if Path is obtained from Look Around then
10:        Set forward distance to predefined look around distance
11:    end if
12:    Start moving forward along the path
13:    while Moving forward do
14:        Continuously monitor front range sensor
15:        if Specified forward distance is reached or front obstacle is detected then
16:            Halt forward motion
17:            Exit loop
18:        end if
19:    end while
20: end if
```

Look around behavior The look around behavior is a crucial component of the Crazyflie's navigation system. When the drone needs to explore its environment or find a new path, this behavior enables it to systematically scan the surroundings while searching for viable navigation routes. The behavior is designed to be both efficient and safe, allowing Crazyflie to gather environmental information while maintaining stable operation.
During the look around process, the drone maintains several state indicators to track its scanning progress. These include flags for path checking initiation and turning states, as well as angle records for movement control. The drone keeps track of its previous angle, current angle, and target angle to ensure precise orientation control during the scanning process. The scanning process begins with the Crazyflie clearing any previous scanning data and recording its initial orientation. It then executes a

systematic 120-degree turn while continuously monitoring the environment. During this rotation, the Crazyflie records distance measurements in multiple directions, checking for obstacles and evaluating path feasibility. The behavior is designed to identify the longest clear path available, taking into account both the distance to obstacles and the angular proximity to the current orientation. To ensure safe operation, the behavior includes several safety features. It can dynamically handle interruptions, such as when obstacles are detected, and automatically adjusts its scanning process accordingly. The angle control is precise, with a 5-degree tolerance to ensure accurate orientation. The behavior also continuously monitors environmental changes and can revalidate paths when conditions change. Throughout the scanning process, the behavior tracks various performance metrics. This includes the duration of each scanning operation, how successfully it finds viable paths, and how quickly it responds to environmental changes. These metrics help in evaluating the behavior's effectiveness and identifying areas for improvement.
The look around behavior has proven effective in various scenarios. It can systematically explore unknown environments, identify and validate navigation paths, and adapt to dynamic changes in the surroundings. The behavior maintains precise control over the drone's orientation while providing reliable path information for subsequent navigation behaviors. This makes it an essential component of the drone's navigation system, enabling safe and efficient exploration of the environment. The implementation of this behavior has been tested extensively in different environments. It has demonstrated reliable performance in maintaining safety margins while gathering essential environmental data. The behavior's ability to provide accurate path information has been crucial for the overall success of the navigation system.

Algorithm 9 Look around behavior

```
if Need to redo (has_InterruptNeedToReDo) then
    if Currently turning or checking paths (turnStarted or clearPathCheckStarted) then
        Stop current turning/checking action
        Reset flags: turnStarted, clearPathCheckStarted
        Reset timer: nTimer = 0
        Set on_TURNING_MODE to false
        Record interruption time
    end if
else if Interruption occurs (has_possibleInterrupt) then
    Record interruption time
    Restart look around timer
    Continue current action with adjusted angle
end if
if Not started path checking (clearPathCheckStarted is false) then
    if First time (nTimer is 0) then
        Increment look around counter
        Record start time
        Clear previous angle records
        Set preAngle = current_yaw + π
    end if
    Record startAngle = current_yaw
    Turn 120° (2 π/3 radians)
    Set clearPathCheckStarted to true
    Set on_TURNING_MODE to true
else if Not started turning (turnStarted is false) then
    if Not reached look around angle (|angleDifference| < 110°) then
        if Interrupted by target or obstacle then
            Adjust turn angle and continue
            Record interruption time
        end if
        Record angles and distances in four directions
        for each recorded angle do
            if Path is clear (no obstacles) then
                Set targetAngle
                Calculate turn angle with 5° tolerance
                Turn to target angle
                Set turnStarted to true
                Break loop
            end if
        end for
        if No clear path found then
            Turn back to preAngle
            Set turnStarted to true
        end if
    end if
end if
```

Algorithm 10 Look around behavior(continued)

```
if Successfully found path then
    Set pathReadyToGo to true
    Record original front distance
    Update target range records
    Reset look around flags
    Set on_TURNING_MODE to false
    Record completion time
end if
```

3.4 Parameters tuning by GA

GA is an optimization method inspired by natural selection and biological evolution processes. The algorithm mimics how nature selects the fittest individuals to produce offspring for the next generation. The individual means a chromosome that could generate a behavior model or a control module. Each generation, pairs of individuals are selected for crossover and mutation, producing new candidates for evaluation. The quality of each individual is assessed using a fitness function. The evaluation is through the fitness function, which is designed to quantify system performance. GA is particularly suitable for optimizing non-differentiable functions or systems that are difficult to model analytically. For example, in a robot navigation task, the fitness function may evaluate how efficiently the robot reaches a target while avoiding obstacles. Relevant metrics could include navigation time, the number of collisions, or the frequency of proximity to obstacles.A sample fitness function is as follows:
Each time two individuals are selected. The fitness could be

$$F(g_{i1}, g_{i2}) = -k_1 \cdot \text{num\_collision} - k_2 \cdot \text{navigation\_cost\_time} \tag{3.5}$$

where k_1 and k_2 are positive integers.
The overall GA workflow is illustrated in Figure 3.10.

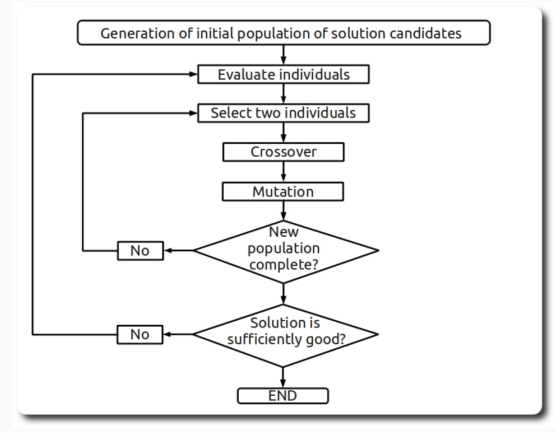


Figure 3.10: The workflow of GA optimization process.

It begins with an initial population of randomly generated individuals. After evaluation, the fittest individuals are selected to generate the next generation via crossover (which combines traits from parents) and mutation (which introduces random variations). This evolutionary loop continues for a fixed number of generations or until the convergence criteria are met.

GA implementation In this thesis, GA is leveraged to optimize the behavior parameters of the FSM-based UAV controller, including safe distances, the number of times being disconnected or stuck, and execution timings. The optimized parameter set is then directly applied to both the subsumption-based and the FSM-based controllers without additional tuning. Interestingly, this transfer also resulted in improved performance for the subsumption controller, demonstrating the generalizability and effectiveness of the GA-tuned behavior parameters across different control architectures. The fitness function is designed to evaluate both the overall mission efficiency and individual behavior performance. The fitness function used for GA tuning is defined as follows:

$$\mathrm{F}_{\mathrm{GA}} = \frac{1}{T_{\mathrm{task}} + \alpha N_{\mathrm{stuck}} + \beta N_{\mathrm{overlimit}} + \gamma N_{\mathrm{timeout}}} \tag{3.6}$$

where:

- T_{task} is the total time taken to complete the task,
- N_{stuck} is the number of times Crazyflie gets stuck during the task,
- $N_{\mathrm{overlimit}}$ is the number of times Crazyflie exceeds operational or safety limits,
- N_{timeout} is the number of times the task is terminated due to timeout,
- α, β, γ are penalty coefficients that weight the importance of each penalty term.

A higher fitness value indicates better performance, i.e., faster task completion with fewer failures or unsafe events. The penalty coefficients can be adjusted to emphasize safety or efficiency according to the requirements of the application. For the implementation, this thesis uses a header-only library[3] for efficient training in C++. The configuration parameters are as follows: population size of 30, 100 generations, a tournament selection method with size two, a crossover rate of 30 percent, a mutation rate of 10 percent, and elitism preserving the best individual per generation. Training is performed in a simulation-based environment built in Webots, running in headless mode. This not only accelerates training but also protects the physical Crazyflie hardware from unstable parameter sets during the optimization process. The GA tuning with the FSM controller allows the GA to directly optimize the state transitions and behavior parameters within the FSM framework. This results in significant performance improvements for the FSM-based controller. However, its effect is less pronounced in the subsumption architecture, where behavior arbitration is inherently more adaptive.

3.5 The real-world environment

The real-world experiments were carried out in a laboratory space that was adapted for indoor drone testing. The available flying area is roughly three meters wide, two

meters long, and three meters high, which provides just enough room for Crazyflie to perform takeoff, landing, and a variety of autonomous maneuvers.

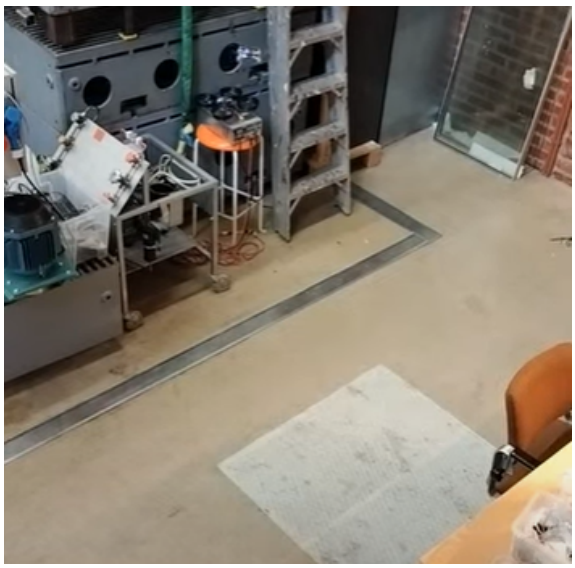


Figure 3.11: The real-world environment.

4

Results

This chapter analyzes the task performance of the two controllers before and after GA tuning. The results are presented using data tables, histogram graphs, and pie charts. The data table provides an overall comparison between the two controllers, with and without GA tuning, and illustrates the time consumption differences for both the overall task and each individual behavior. The histogram graph compares the GA optimization effects based on the variants defined in the fitness function—cost–time and count–time—across two map scenarios. Finally, the pie chart presents the cost–time distribution of each controller, with and without GA tuning, revealing which parts of the task contribute most to the total execution time and where further improvements can be made.

4.1 Result analysis

Each controller was tested in ten flights per map. With two maps, two controller architectures, and two configurations (with and without GA tuning), a total of 80 flight experiments were conducted. Each experiment involved the same tasks—finding paths, navigation, avoid obstacles, find targets, and homing—performed in both environments. The controllers were evaluated based on four performance metrics: total task completion time, time spent in each behavior state (e.g., reach front or avoid static obstacle), the number of transitions between behaviors indicating stability, and safety-related durations such as the time spent being stuck or remaining close to dynamic obstacles like the ball.

4.1.1 General performance

Here is the overall performance comparison in simpler(four-room map) and complex(maze map) scenario.

		FSM		Subsumption	
		Original	GA	Original	GA
Static Obs. avoid.	avg. time spend	1.91	0.1	1.4	1.5
	execution count	4.3	0.9	3.2	2.9
Dynamic Obs. avoid.	avg. time spend	1.62	1.6	1.5	1.6
	execution count	7.5	5.6	6.9	7.2
Target Finding	avg. time spend	2.8	2.1	3.1	2.5
	execution count	13	10.8	11.9	10.7
Front Reaching	avg. time spend	2.3	1.6	1.7	1.5
	execution count	14.4	13.5	13	12.7
Look Around	avg. time spend	2	1.6	2.2	2.6
	execution count	14.9	14.6	10.7	10.1
Stuck count		0.2	0	0.1	0
Overall time spend		120.1	85.6	111.1	94.1

Figure 4.1: Result for the map of four rooms.

Performance analysis: four-room map The figure above illustrates the overall performance on the four-room map. Among all configurations, the original FSM controller required the longest time to complete the task, with a total duration of 120.1 seconds. In comparison, the GA-tuned FSM completed the task in 85.6 seconds. Before GA tuning, both FSM and subsumption controllers showed similar performance, with comparable execution times across individual behaviors. After tuning, however, both architectures demonstrated clear improvements in efficiency. Specifically, the FSM controller achieved an average 34.3% reduction in behavior execution time and a 28.7% decrease in total mission duration, while the subsumption controller showed a 26.7% average reduction in behavior time and a 15.3% shorter total duration. These results indicate that GA-based parameter tuning improves overall control efficiency, with the FSM architecture showing a greater benefit than subsumption.

		FSM		Subsumption	
		Original	GA	Original	GA
Static Obs. avoid.	avg. time spend	0.7	0.6	1.4	2.4
	execution count	16.3	9.9	9.7	3.3
Dynamic Obs. avoid.	avg. time spend	1.1	1.1	1.1	1.1
	execution count	9.5	7.7	10.6	5.4
Target Finding	avg. time spend	2.8	2	2.9	3.1
	execution count	6.6	4.9	7.1	5.7
Front Reaching	avg. time spend	2.8	1.8	2.3	1.8
	execution count	22.2	17.5	16.7	14.5
Look Around	avg. time spend	3.8	4.3	3.9	4.3
	execution count	31.1	19.9	17.8	13
Stuck count		0.1	0	0	0
Overall time spend		221.4	137.6	157.5	119.2

Figure 4.2: Result for the map of maze.

Performance analysis: maze map Figure 4.2 presents the performance statistics for the maze map. Among all configurations, the original FSM controller re-

quired the longest time to complete the task, with a total duration of 221.4 seconds. In comparison, the GA-tuned subsumption controller achieved the shortest completion time of 119.2 seconds, representing an improvement of approximately 46% over the original FSM. These results are particularly noteworthy given that Crazyflie operated without access to a global map, relying solely on local sensing and iterative exploration to navigate the environment. Regarding safety performance, the stuck count remained close to zero in all cases, with a maximum average of only 0.1 per trial. This indicates that both controllers were able to robustly avoid dead ends and unresolvable situations. Similarly, no unsafe proximity events to moving obstacles were observed, resulting in a consistently zero "close encounter" metric, which was therefore omitted from the table. These findings confirm that both tuned and untuned controllers maintained appropriate safety margins throughout the experiments. Detailed analysis of behavior-wise execution of the impact of GA tuning on the performance of both controller architectures will be discussed in the following section.

4.2 GA algorithm tuning result

The GA was executed for 100 generations to optimize behavior parameters. The best-performing individuals for maze and four-room are summarized in Figure 4.3. The fitness values achieved were 0.00931 (maze) and 0.01579 (rooms), where fitness is defined as the inverse of mission time with penalties.

Parameters	Original settings	GA Maze	GA Rooms
Safe distance (used in obstacle dodging behaviors)	if front ≥1.0m → 0.25 x front + 0.6m else → 0.6m	0.341499 x front + 0.25m	0.0884062 x front + 0.25m
Dodging distance (used in obstacle dodging behaviors)	0.2m	0.2145m	0.2497m
Distance to move (used in reaching rear in obstacle dodging behaviors)	1.0 x rear – 0.125m	0.857911 x rear	0.960318 x rear
Time to move (used in reaching rear in obstacle dodging behaviors)	if front ≥1.0m → 4s else → 5s	0.240129 x dist. to move cp=> 2s–5s	0.247781 x dist. to move cp=>2s–5s
Distance to move for look around (used in reaching look around found path)	1.0 x front – 0.125m	0.793526 x front	0.370962 x front
Time to move for look around (used in reaching look around found path)	if front ≥1.0m → 4s else → 5s	0.645228 x dist. to move cp=>2s–5s	0.308037 x dist. to move cp=>2s–5s
Distance to move for reaching target (used in reaching target finding found path)	if front ≥0.7m → 0.7m else → 1.0 x front – 0.125m	0.737 x front	0.544328 x front
Time to move for finding target (used in reaching target finding found path)	if dist. to move ≥0.7m → 2s else → 5s	0.715185 x dist. to move cp=>2s–5s	0.164879 x dist. to move cp=>2s–5s
Turning angle for finding target	90.0° / 180·π	109.396° / 180·π	27.5064° / 180·π
Turning angle for look around	360.0° / 180·π	296.5176° / 180·π	320.1346° / 180·π
Time for turning action	Target finding: 4s to find target 2s to turn to target Look around: 3s to look around 2s to turn to target	0.83811 x angle to turn cp=>1s–5s	0.258119 x angle to turn cp=>1s–5s

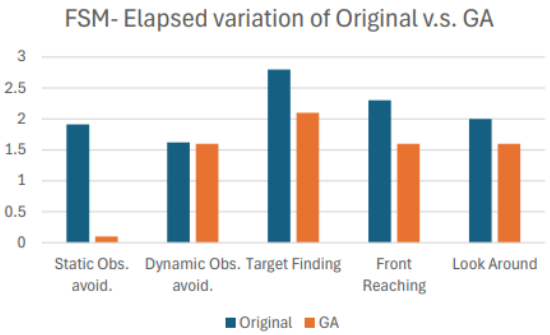
Figure 4.3: Parameters comparison before and after GA tuning.

The optimized parameters indicate that the GA-tuned controllers adopted more dynamic and less conservative control strategies. The safe distance was notably reduced in both maps (for example, from approximately 0.6 meters to 0.09 meters in the four-room map), suggesting that the UAV operated with a smaller safety margin when approaching obstacles. The dodging distance and duration showed only minor adjustments, implying that the original settings were already close to optimal; however, the overall dodging duration decreased, particularly in the room map, improving responsiveness during avoidance maneuvers. The reach front behaviors also became more efficient. When used for exploration, the movement time was reduced from four seconds to two seconds in both maps, with travel distance correspondingly decreased—indicating a faster local exploration strategy. When used for target approach, the travel distance slightly increased in the maze map but decreased in the room map, reflecting adaptive motion based on target proximity. Turning angles and durations were likewise refined: yaw angles were generally reduced—especially during find target behavior in the four-room map—while the rotation time was shortened to approximately one second, enabling quicker reorientation. These results suggest that the GA favored faster, more responsive behaviors, effectively trading some safety margin for improved task efficiency. The detailed behavior-wise time consumption and execution counts after GA tuning are presented in the following

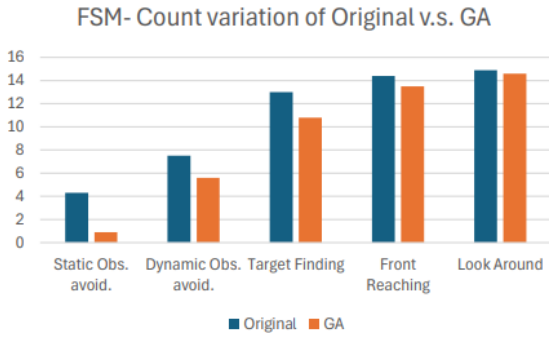
figures, which compare each optimized controller to its original version.

Comparison before and after GA tuning

In this section, we compare the performance of both FSM-based and subsumption-based controllers before and after GA tuning.

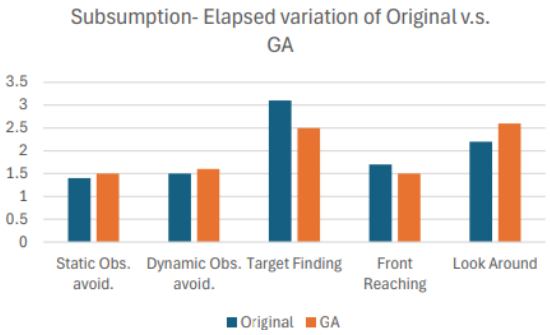


(a) Time spent in each behavior for FSM.

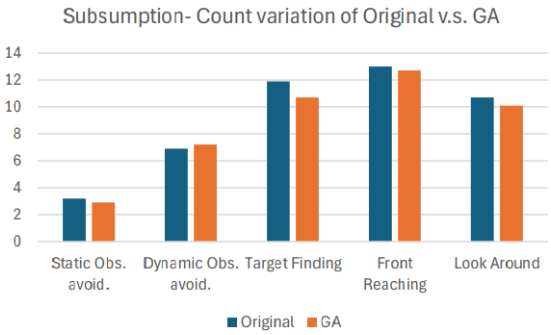


(b) Behavior activation count for FSM.

Figure 4.4: FSM controller performance on the four-room map.

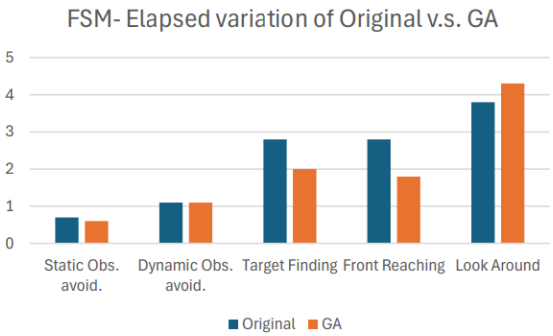


(a) Time spent in each behavior for subsumption.

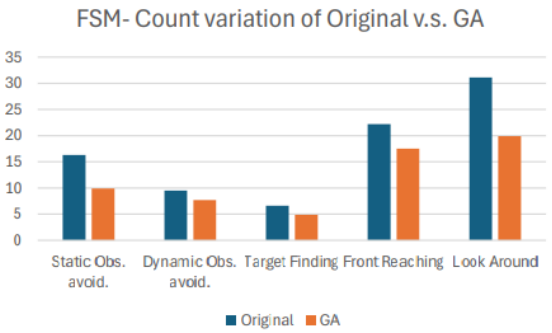


(b) Behavior activation count for subsumption.

Figure 4.5: Subsumption controller performance on the four-room map.

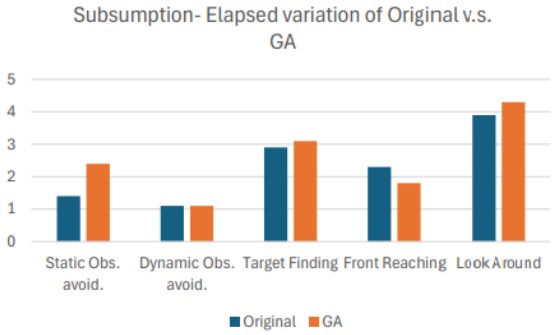


(a) Time spent in each behavior for FSM.

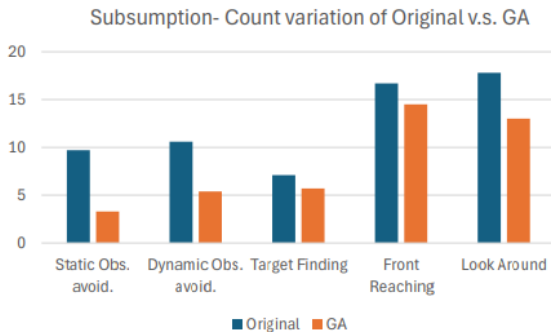


(b) Behavior activation count for FSM.

Figure 4.6: FSM controller performance on the maze map.



(a) Time spent in each behavior for subsumption.



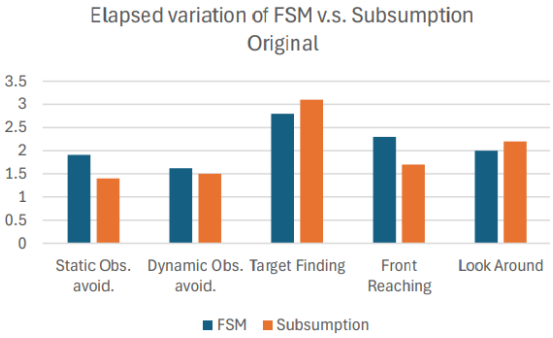
(b) Behavior activation count for subsumption.

Figure 4.7: Subsumption controller performance on the maze map.

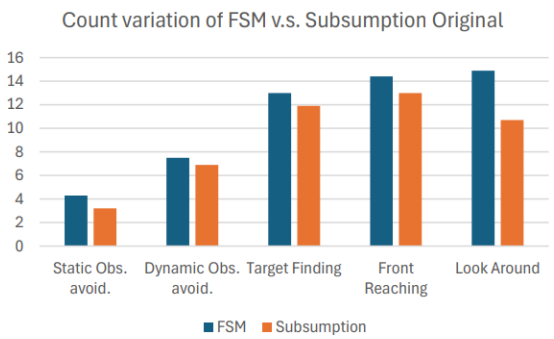
Summary of observations The experimental results highlight several consistent patterns across controllers and environments. First, the FSM controller benefited the most from GA tuning. Its untuned version was less reactive and prone to redundant actions, which left more room for improvement. After optimization, both total execution time and the frequency of wall-avoidance behaviors decreased noticeably, particularly in the maze map. In contrast, improvements in the subsumption controller were more modest but stable. The original design already minimized unnecessary activations, so tuning mainly refined timing and responsiveness. The tuned version adopted slightly more aggressive strategies—using shorter safety distances and faster turns—which led to more frequent triggering of higher-priority behaviors such as avoid obstacles. The impact of GA tuning also depended on the map complexity. The maze environment, with its narrow corridors and dead ends, revealed larger efficiency gaps between original and tuned parameters, while the simpler four-room map offered less opportunity for improvement. Finally, after tuning, the FSM controller outperformed the Subsumption architecture in the four-room map, mainly due to its enhanced performance in avoid static obstacles and look around behaviors.

Comparison between FSM and subsumption structures

In this section, we evaluate the structural efficiency of two behavior-based control architectures—the FSM and subsumption—in both their original and GA-optimized versions.

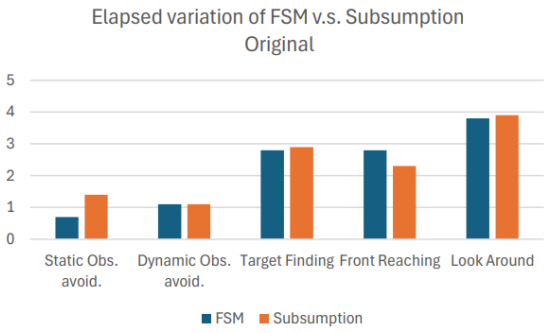


(a) Time spent in each behavior for FSM and subsumption.

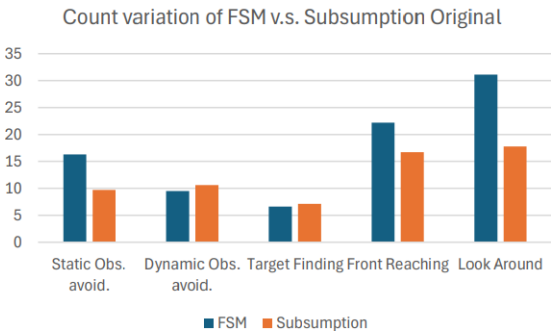


(b) Behavior activation count for FSM and subsumption.

Figure 4.8: Comparison between FSM and subsumption of the original version on the four-room map.

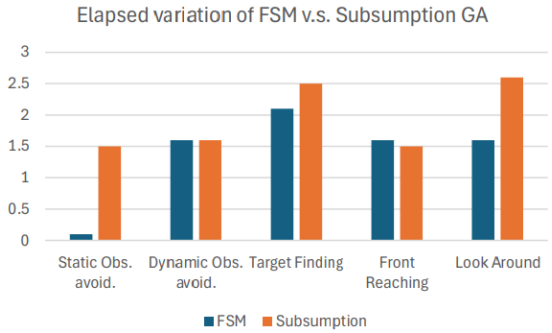


(a) Time spent in each behavior for FSM and subsumption.

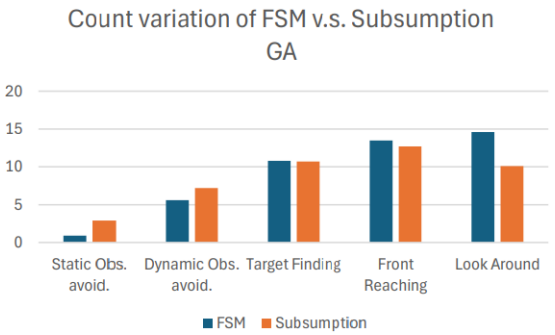


(b) Behavior activation count for FSM and subsumption.

Figure 4.9: Comparison between FSM and subsumption of the original version on the maze map.

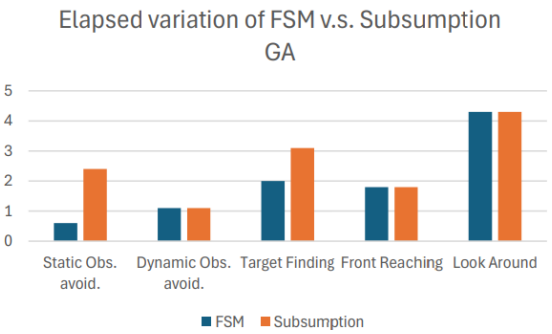


(a) Time spent in each behavior for FSM and subsumption.

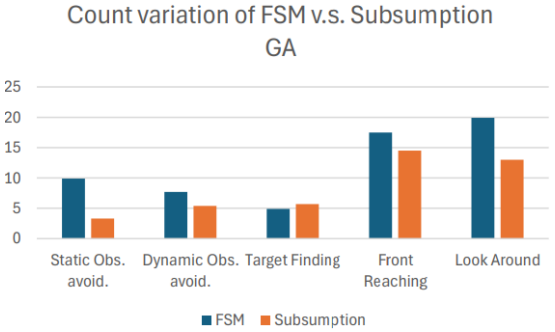


(b) Behavior activation count for FSM and subsumption.

Figure 4.10: Comparison between FSM and subsumption of the GA version on the maze map.



(a) Time spent in each behavior for FSM and subsumption.



(b) Behavior activation count for FSM and subsumption.

Figure 4.11: Comparison between FSM and subsumption of the GA version on the maze map.

Discussion of behavior-wise performance comparison Firstly, figure 4.8 and 4.9 summarize the performance of the FSM and subsumption controllers in both the four-room and maze maps without tuned by GA. In the four-room map, the untuned FSM showed longer average durations in behaviors such as find target (2.8 s vs. 1.9 s) and look around (2.2 s vs. 1.8 s) and triggered behaviors more frequently, for example, reach front occurred 14 times in FSM compared to 12 in subsumption. In contrast, the subsumption controller inherently avoided redundant behavior calls, resulting in shorter durations and fewer activations. In the more complex maze map, these differences were more pronounced. The FSM exhibited higher look around durations (4.2 s vs. 3.1 s) and overall higher behavior invocation counts, such as avoid static obstacle (18 vs. 10), reflecting inefficiencies in dense environments. Subsumption maintained more consistent and reactive behavior activation, avoiding unnecessary repetitions.

In figure 4.10 and 4.11, after GA tuning, the performance gap between the two architectures narrowed considerably. In the four-room map, FSM execution times for avoid static obstacle behavior dropped near zero, while look around remained stable at 1.9 s, and behavior counts became comparable to subsumption. In the maze map, FSM also reduced its avoid static obstacle duration from 3.1 s to 0.7 s, slightly outperforming subsumption, while counts for reach front and look around remained somewhat higher but with better time efficiency. Overall, these observations indicate that GA tuning allowed FSM to achieve major improvements, particularly in behaviors that were previously inefficient. In the simpler four-room map, the tuned FSM even surpassed subsumption in overall behavior efficiency. However, in the maze map, subsumption retained an advantage in robustness, benefiting from fewer interruptions and more responsive task-switching in complex paths.

Distribution among behaviors

Figures 4.12–4.19 present the percentage of time each controller spent in different behavior states across maps and tuning conditions.

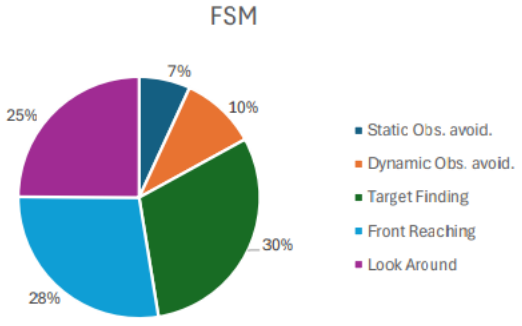


Figure 4.12: FSM in four-room map.

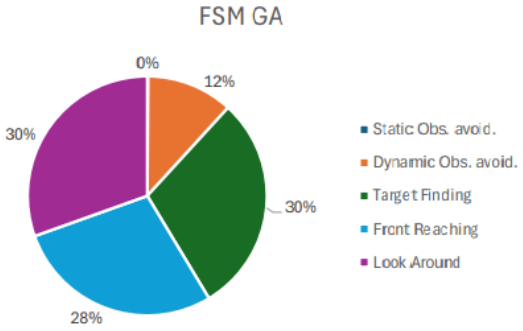


Figure 4.13: FSM after GA training in four-room map.

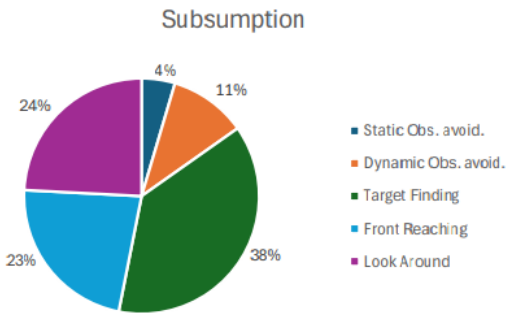


Figure 4.14: Subsumption in four-room map.

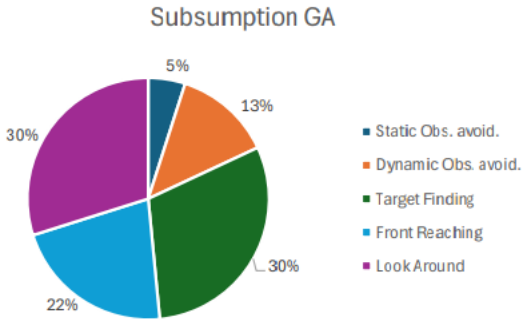


Figure 4.15: Subsumption after GA training in four-room map.

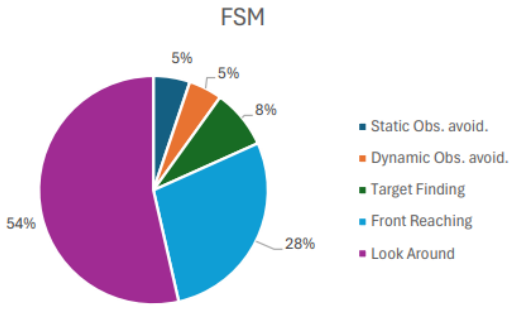


Figure 4.16: FSM in maze map.

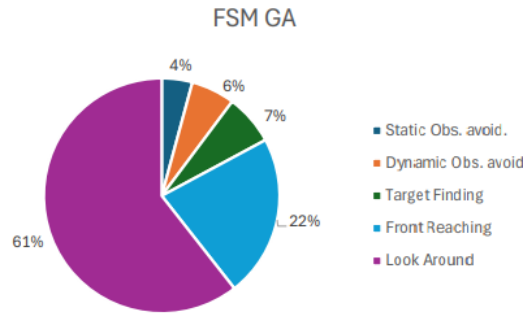


Figure 4.17: FSM after GA training in maze map.

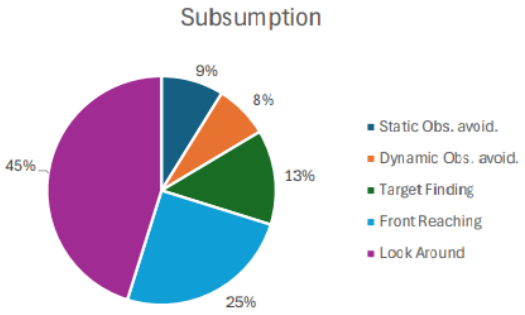


Figure 4.18: Subsumption in maze map.

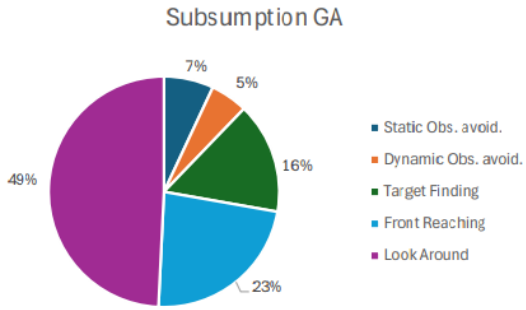


Figure 4.19: Subsumption after GA training in maze map.

Four-room map (Figures 4.12–4.15) In the four-room environment, the original FSM controller allocated most of its time to target-oriented behaviors (find target and reach front), which together accounted for about 58% of total execution. Avoid obstacle represented 17%, divided between 10% avoid static obstacle and 7% avoid dynamic obstacles. After GA tuning, the share of obstacle avoidance dropped sharply to 12%, with static avoidance nearly eliminated. The time devoted to mission-related behaviors remained around 58%, indicating improved efficiency in obstacle handling without altering overall task focus. The original subsumption

controller showed a similar pattern, with 61% of time spent on target behaviors and only 15% on avoiding obstacle (11% static and 4% dynamic), reflecting its naturally reactive design. After GA tuning, subsumption exhibited a slightly more conservative strategy: time spent on target behaviors decreased to 52%, while obstacle avoidance rose modestly to 18%.

Maze map (Figures 4.16–4.19) In the maze map, the FSM controller initially spent most of its time on the look around behavior (54%), while target-related behaviors made up 36%. Avoiding obstacles was relatively low at 13%. After GA optimization, the look around proportion further increased to 61%, while time for target-oriented actions decreased to 29%, suggesting a more cautious exploration strategy adapted to narrow corridors. The original subsumption controller displayed a better balance between exploration and goal seeking, with 45% of time in look around, 38% in target behaviors, and 17% in avoiding obstacles. After GA tuning, subsumption slightly increased its target behavior share to 39% while maintaining 49% for look around, indicating that tuning reinforced its goal-directed behavior while preserving stable exploration.

Summary In the four-room scenario, the GA-tuned FSM achieved the lowest proportion of time in obstacle avoidance (12%) and matched subsumption in target-behavior emphasis, confirming its efficiency in simpler, open layouts. In contrast, within the maze map, subsumption with GA tuning devoted more time to target-oriented behaviors (39%) than FSM (29%), maintaining its advantage in complex environments. Across all experiments, the reach front behavior remained consistently stable (22–30%), highlighting its central role in trajectory execution. Overall, the pie chart analysis aligns with the histogram results, showing that GA tuning primarily enhances efficiency by reducing obstacle-related overhead rather than altering the underlying task structure.

5

Discussion

This chapter discusses the findings in relation to the three *research questions* (RQs) introduced in Section 1.4. Specifically, it examines: (1) the feasibility of controlling a UAV using a HIL setup, (2) the effectiveness of BBR without access to a global map, and (3) the impact of genetic algorithm GA-based parameter tuning on controller performance. Each of the following subsections addresses one of these research questions in detail.

5.1 Feasibility of the HIL-rig

The goal of employing the HIL framework is to enable real-time feedback, component decoupling, and system-level integration validation. These objectives were successfully achieved through three key design aspects: real-time communication, modular decoupling, and system validation. Real-time interaction between the Crazyflie and the simulation environment was established via the CRTP protocol, ensuring synchronized control loops and sensor feedback identical to those of the physical drone. System-level validation was achieved through containerization using Docker, where multiple services operated cohesively via the `docker-compose.yml` configuration. Modular decoupling was further supported by the microservice architecture and the BBR, whose individual behavior blocks could be independently developed and tested. The modular design also enhanced maintainability and resilience. For instance, hardware issues such as motor imbalance—caused by uneven battery placement—were resolved through minor manual adjustments, followed by a complete system restart using a single docker compose up command. Software updates were equally efficient; algorithm logic could be modified, recompiled into containers, and redeployed without any hardware reconfiguration. Even after relocating the system to a different laboratory, all experiments were completed within a week, demonstrating the flexibility and robustness of the architecture. The same modular structure facilitated seamless integration with GA-based tuning. For safety reasons, GA iterations were conducted in a SIL setup, where the control stack connected directly to the Webots simulator. This required only redirecting the control service, with all other modules remaining unchanged. The setup enabled continuous parameter optimization under realistic feedback while ensuring the safety of the physical hardware.

5.2 Behavior-based navigation without global map

This section evaluates whether reliable UAV navigation can be achieved using a behavior-based controller without relying on localization or a global map. The controller successfully completed all core exploring tasks, avoiding obstacles, localizing target, and returning to the charging pad—across all trials. Failures occurred only at take-off due to occasional hardware malfunctions, not controller design. Once in the air, the Crazyflie consistently completed its missions, confirming the robustness of the proposed behavior-based architecture. Behavior execution was stable and efficient. Stuck occurrences were negligible, and no behavior exceeded an average of four seconds per activation, demonstrating smooth transitions and responsive control even in complex environments. Task completion times ranged from 90 to 221 seconds across the two map configurations. Despite the lack of localization or prior map knowledge, this performance is considered highly efficient. The four-room map yielded faster completion due to its open structure, whereas the maze required more time in visually constrained corridors. Time distribution analysis further supports this conclusion: most execution time was devoted to goal-directed behaviors such as find target and reach front, while avoid obstacles accounted for less than 18% of total runtime, indicating that the controller primarily focused on mission progress rather than excessive reactivity.

Performance differences between the two architectures emerged clearly after GA tuning. In the four-room map, FSM+GA demonstrated superior efficiency, achieving shorter durations in key behaviors including avoid static and dynamic obstacles, find target, and look around actions. Although its phase of reach front lasted slightly longer, the difference was negligible, suggesting that FSM+GA performs best in structured, open environments where rapid state transitions enhance agility. Conversely, Subsumption+GA achieved faster task completion in the maze, particularly in find target and look around behaviors, outperforming FSM+GA by 8 and 30 seconds, respectively. While it spent marginally more time on avoiding obstacles, this trade-off was offset by improved consistency in confined, cluttered spaces. These results highlight the fundamental architectural difference between FSM and subsumption control. Subsumption triggers behaviors less frequently and maintains higher-priority actions for longer durations due to its suppression mechanism, minimizing unnecessary state transitions and improving focus in complex settings. However, this persistence can also lead to slightly longer individual behavior durations when the environment changes unexpectedly. In contrast, FSM switches behaviors more frequently in response to sensor input, offering agile adaptation that benefits open environments but may cause redundant switching in dense ones. Together, these findings confirm that both architectures are reliable for navigation without localization, each excelling under different environmental constraints.

5.3 GA tuning performance

This section examines whether GA tuning improves the performance of behavior-based controllers. The experimental results demonstrate that GA optimization sub-

stantially enhances controller efficiency. Both the FSM and subsumption architectures exhibited faster overall task completion and shorter average behavior durations after tuning, confirming that evolutionary parameter adjustment effectively refines behavioral responses. In the four-room map, the GA-tuned FSM achieved a 28.7% reduction in total task time compared to its original version, while the tuned subsumption controller improved by 15.3%. In the more complex maze map, FSM improved by 21.5%, whereas subsumption achieved the highest overall gain at 28.1%. These outcomes indicate that both architectures benefit from GA optimization, with each showing strengths in different spatial contexts. Behavior execution also became more consistent after tuning. Average durations of most behaviors decreased, transitions stabilized, and redundant activations or "stuck" conditions were notably reduced. These trends suggest that GA tuning effectively calibrates timing and distance parameters, enabling smoother and more adaptive control dynamics. In open environments, FSM benefited more due to its rapid switching capability, whereas subsumption gained greater stability and efficiency in dense or cluttered settings where reactive suppression offers an advantage. Overall, GA tuning proves to be a reliable and effective strategy for optimizing behavior-based robotic controllers, enhancing both efficiency and robustness across diverse environmental conditions.

6
Conclusion

This thesis investigated the design, implementation, and evaluation of two behavior-based control architectures—FSM and subsumption—within a HIL framework. GA tuning was applied to both structures to optimize their performance in indoor navigation tasks. The central research objective was to determine whether UAVs can navigate autonomously in complex environments without relying on global maps or localization. Experimental results demonstrated that both controllers enabled the drone to complete key tasks: exploration, obstacle avoidance, target localization, and return to a designated endpoint. The HIL system proved to be a realistic, rapid, and flexible testing environment, successfully replicating real-world control dynamics and enabling fast iteration. This was particularly evident in the low-latency control response and stable behavior switching observed during different simulation environments. Comparative analysis revealed that FSM+GA controller was more efficient in open, structured environments such as the four-room map, while subsumption+GA excelled in complex, enclosed layouts like the maze. Overall, subsumption+GA achieved the best average performance, particularly in environments requiring adaptive prioritization of behaviors. These findings confirm that behavior-based controller, when enhanced through GA tuning, offers a robust and generalizable approach to autonomous UAV navigation. The method requires minimal prior knowledge of the environment, and its modular structure supports safe and scalable development within simulated or semi-physical systems.

6.1 Future work

Several directions can extend the contributions of this thesis. Firstly, GA tuning is currently designed based on an FSM-style parameter structure. Developing a GA logic specifically tailored to the subsumption architecture could unlock higher performance through more effective parameter discovery. Although navigation occurs in a three-dimensional space, most decisions are confined to a two-dimensional plane; implementing true 3D avoidance would require extending the control service to interpret vertical sensor data and generate full 3D motion vectors. The current testing environment is limited to a three by three by two meters area, which restricts the study of long-range behaviors such as energy-efficient flight, corridor traversal, or memory-based exploration. Moreover, only two static maps were used in the experiments; incorporating procedurally generated or more cluttered environments could better assess generalization and robustness in unfamiliar spatial arrangements.

Dynamic obstacles in the current simulation followed predefined paths, which limits realism. Future work should consider erratic or probabilistic movement patterns to test the adaptability of the dynamic obstacle avoidance logic. In addition, all experiments involved a single drone. Extending the framework to multi-agent systems may reveal emergent group behaviors, such as cooperative exploration and distributed sensing. Finally, although the HIL setup closely approximates real-world physics, no physical obstacle tests were performed. Integrating real hardware with tangible obstacles would be the ultimate step toward validating the proposed architecture under realistic operating conditions.

Bibliography

[1] G. Astuti, D. Longo, C. D. Melita, G. Muscato, and A. Orlando. Hil tuning of uav for exploration of risky environments. *International Journal of Advanced Robotic Systems*, 5(4):36, 2008.

[2] O. Benderius. odvd message set. `https://git.opendlv.org/core/opendlv-message`.

[3] O. Benderius. tinyso.hpp, 2023.

[4] O. Benderius. opendlv-virtual-camera. 2024.

[5] O. Benderius and C. Berger. Opendlv: Microservice-based software for autonomous vehicles. `https://opendlv.org/index.html`, 2015.

[6] C. Berger. libcluon: A single-file c++ library to glue microservices. `https://github.com/chrberger/libcluon`.

[7] Bitcraze AB. Crtp - crazy realtime protocol. `https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/`.

[8] R. A. Brobrooksoks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume 2, pages 14–23, 1986.

[9] Y. Chang, Y. Cheng, U. Manzoor, and J. Murray. A review of uav autonomous navigation in gps-denied environments. *Robotics and Autonomous Systems*, 170:104533, 2023.

[10] V. D'antuono, G. De Matteis, D. Trotta, and A. Zavoli. Optimization of uav robust control using genetic algorithm. *IEEE Access*, 2023.

[11] V. Hristov and H. Stefanov. Generalization of a genetic algorithm for optimal coding of the states of a finite state machine. In *2023 International Scientific Conference on Computer Science (COMSCI)*, pages 1–4, 2023.

[12] H. Huang, J. Gu, Q. Wang, and Y. Zhuang. An autonomous uav navigation system for unknown flight environment. In *2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, pages 63–68, 2019.

[13] W. Hönig. crazyflie cpp library. 2021.

[14] M. Jaswanth, N. K. L. Narayana, S. Rahul, and M. Supriya. Autonomous car controller using behaviour planning based on finite state machine. In *2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 296–302, 2022.

[15] S. Kamthe, C. Shidnal, U. Mishra, G. Gandhi, and S. Chaudhury. Wanderguide: Indoor map-less robotic guide for exploration by blind people, 2024. arXiv preprint arXiv:2502.08906.

[16] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 500–505, 1985.

[17] H. Li and H. Wu. Multi-strategy synthesis based artificial bee colony algorithm for uav path replanning. In *2023 5th International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 442–448, 2023.

[18] X. Li, Y. Fang, and W. Fu. Multi-uav cluster obstacle avoidance algorithm based on artificial potential field and dubins path planning. In *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pages 593–598, 2019.

[19] Z. Lin and L. Wei. Application of adaptive artificial potential field method in obstacle avoidance path planning. In *2014 Seventh International Symposium on Computational Intelligence and Design (ISCID)*, volume 2, pages 429–432, 2014.

[20] T. T. Mac, C. Copot, R. D. Keyser, and C. M. Ionescu. The development of an autonomous navigation system with optimal control of an uav in partly unknown indoor environment. *Mechatronics*, 49:187–196, 2018.

[21] Z. S.-Z. W. e. a. Meng, H. Optimizing actual pid control for walking quadruped soft robots using genetic algorithms. *Sci Rep 14*, 2024.

[22] S. Nantogma, W. Ran, X. Yang, and H. Xiaoqin. Behavior-based genetic fuzzy control system for multiple usvs cooperative target protection. In *2019 3rd International Symposium on Autonomous Systems (ISAS)*, pages 181–186, 2019.

[23] R. Perez-Segui, P. Arias-Perez, J. Melero-Deza, M. Fernandez-Cortizas, D. Perez-Saura, and P. Campoy. Bridging the gap between simulation and real autonomous uav flights in industrial applications. *Aerospace*, 2023.

[24] A. S.-N. Peter Lepej and J. Sola. A flexible hardware-in-the-loop architecture for uavs. *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017.

[25] D. Stojcsics and A. Molnar. Fixed-wing small-size uav navigation methods with hil simulation for aerobot autopilot. In *2011 IEEE 9th International Symposium on Intelligent Systems and Informatics*, pages 241–245, 2011.

[26] S. Supratno, Rohamid, P. W. A. Sucipto, A. Firasanti, R. A. Adara, and E. A. Z. Hamidi. Obstacle avoidance behavior design in hexapod robots using finite state machine. In *2023 IEEE 9th International Conference on Computing, Engineering and Design (ICCED)*, pages 1–4, 2023.

[27] L. Tang, S. Dian, G. Gu, K. Zhou, S. Wang, and X. Feng. A novel potential field method for obstacle avoidance and path planning of mobile robot. In *2010 3rd International Conference on Computer Science and Information Technology (ICCSIT)*, volume 9, pages 633–637, 2010.

[28] N. D. M. P. H. N. Vu Ngoc Son, Pham Van Cuong. Optimize the parameters of the pid controller using genetic algorithm for robot manipulators. *arXiv*, 2025.

[29] F. Wang, J. Cui, S. K. Phang, B. M. Chen, and T. H. Lee. A mono-camera and scanning laser range finder based uav indoor navigation system. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 694–701, 2013.

[30] F. Q. W. J. G. L. W. J. H. Wei. Obstacle avoidance of flapping-wing air vehicles based on optical flow and fuzzy control. *Transactions of Nanjing University of Aeronautics Astronautics . Apr2021, Vol. 38 Issue 2, p206-215. 10p*, 2021.

[31] Y. Xu. Research on uav navigation system based on behavioral programming. In *2024 IEEE 7th International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, pages 419–425, 2024.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY