POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Politecnico
di Torino

Master's Degree Thesis

Active Inference for Autonomous Exploration on Nano-drones

Supervisors

Prof. Daniele Jahier PAGLIARI

Prof. Alessio BURRELLO

Dr. Matteo RISSO

Beatrice Alessandra MOTETTI

Carlo MARRA

Candidate

Alessandro VALENTI

July 2026

Abstract

Nano-drones are well suited to indoor object-search tasks because their small size allows them to operate in confined spaces and acquire visual observations from viewpoints that fixed sensors or ground robots may not reach. Approaches that rely exclusively on object detection are not optimal for exploration, as they do not account for the fact that motion planning influences future observations. Instead, since each movement directly determines the next observation, search performance can be improved by exploiting visual evidence also for trajectory generation and action selection. This thesis studies this interaction in a robotic simulation environment with a downward-facing camera, using a physical Crazyflie drone model implemented in Webots. The drone moves in continuous space, while a discrete grid provides the task-level representation used to relate camera coverage, detector observations, target locations, and policy state. This representation supports direct comparisons between coverage-based exploration and perception-aware navigation under both learned and ground-truth perception. The main contribution is the development and evaluation of two Bayesian-based navigation policies based on the so-called active inference. The first maintains a binary target-presence belief over grid cells and selects future viewpoints using an expected-free-energy-inspired score that combines task preference, expected information gain, and movement cost. The second extends this approach to a more complex environment in which target objects can appear on raised surfaces as well as on the floor. This second policy adds elements such as categorical target beliefs, explicit table-surface beliefs, table-conditioned target priors, and a table-aware coverage model for raised surfaces. In this setting, desks are not targets, but contextual surfaces that affect both where objects are likely to appear and how camera coverage should be interpreted. The evaluations show that perception-aware navigation based on active inference improves search performance over the baselines in the test environments. In the floor-only environment, the simple active-inference-inspired policy reduces average task-completion time by 10.1% under ground-truth perception. In the environment containing tables, the zone-of-interest policy improves the success rate from 75% to 87% under ground-truth perception, and from 67% to 83% under learned YOLO-based perception. These results indicate that probabilistic, perception-aware navigation improves object search efficiency, while also showing that the contextual scene structure, such as table surfaces, can be integrated into belief-driven search policies using only observations available at runtime.

Table of Contents

List of Tables

List of Figures

Acronyms

AIF
active inference

AI
artificial intelligence

API
application programming interface

CNN
convolutional neural network

FoV
field of view

IMU
inertial measurement unit

PID
proportional-integral-derivative

RL
reinforcement learning

SSD
Single Shot MultiBox Detector

UAV
unmanned aerial vehicle

YOLO
: You Only Look Once

GT
: ground truth

ZOI
: zone-of-interest

MCU
: microcontroller unit

ULP
: ultra-low power

Chapter 1

Introduction

Autonomous object search with nano-drones is a constrained robotics problem in which perception, navigation, and control must operate together under severe physical and computational limitations. Nano-UAVs are attractive platforms for indoor inspection, monitoring, and object-search tasks because their small form factor supports safer close-proximity operation in confined spaces, while their three-dimensional mobility enables flexible camera placement and access to otherwise difficult-to-observe areas [1, 2]. The same scale, however, also imposes important constraints. Limited payload capacity restricts the available sensing and onboard computation. At the same time, flight control must remain stable in real time, and the system must make decisions from incomplete visual information while continuing to move through the environment.

This thesis studies a simulated version of this problem using a Crazyflie nano-drone equipped with a downward-facing camera. The drone flies in continuous space inside a closed indoor environment, while the task representation discretizes the room into grid cells for target localisation and camera coverage estimation. The camera footprint is therefore not treated as a binary cell-to-cell transition, but as a continuous FoV projection that produces coverage values over grid cells. The drone must search for and classify bottles and cans while navigating through the environment using different exploration and active-sensing policies. In the two extended table scenarios, desks are also present as non-target contextual objects: they are not part of the task objective, but they define raised surfaces where targets are more likely to appear. The project uses Webots as the simulation platform. This work develops the controller, supervisor, and reporting components required to compare navigation policies across learned and ground-truth detector settings.

A simple way to approach indoor drone search is to use navigation policies that do not rely on visual perception for deciding where to move. In this thesis, such methods are referred to as naive policies: they are used as lightweight baselines whose movement decisions do not depend on detector evidence or internal search

beliefs. They are attractive because they can be implemented as simple local rules, without requiring the policy to maintain an explicit representation of the search task. However, they do not reason about where targets are likely to be, how reliable detector observations are, which regions remain uncertain, or whether a previously detected object still needs to be classified. In the context of object search, this creates an important limitation: covering space is not always the same as gathering the most useful information. A policy may repeatedly observe low-value regions, fail to prioritize areas with target evidence, or treat all locations as equally important even when the environment contains semantic structure.

This motivates a perception-aware approach to navigation. More generally, perception should not only be used to report detections after a trajectory has been executed, but should also be able to influence future navigation choices. Different approaches could be used to achieve this, including learned navigation policies, heuristic perception-driven rules, or probabilistic models. In this thesis, the chosen approach is to define lightweight probabilistic policies that maintain beliefs over the discretized task representation while still commanding continuous drone motion. Within this representation, a cell that has been observed with no detection may become less likely to contain a target, while a cell with weak or partial target evidence may become a useful candidate for closer inspection. Likewise, a policy can trade off exploration of uncertain regions against exploitation of promising target evidence. This view connects the drone search problem to active perception, in which sensing is treated as an action-dependent data acquisition process: an agent selects motions that shape its subsequent observations and can prefer those expected to yield task-relevant information [3, 4].

The AIF-inspired policies developed in this thesis are motivated by this active sensing perspective. Active inference [5] provides a probabilistic framework in which an agent maintains beliefs over hidden states and evaluates candidate actions through their expected future observations, prior preferences, and expected uncertainty reduction. In this thesis, movement cost is included as an additional task-specific term in the action score. The specific inspiration for this project comes from an active-inference saccade-agent model [6], where an agent controls the fixation point of a camera over a discretized visual field. In that setting, the agent can explore a scene when it lacks information, but it can also shift fixation toward regions where object-like evidence suggests that a class could be resolved. The idea is to adapt the principles of the original saccade model to a mobile drone: the drone must choose target cells and yaw orientations, navigate continuously through the simulated room, and update beliefs from detector observations projected into grid cells.

The first AIF-inspired policy developed in this project is the simple AIF saccade policy. This policy adapts the saccade idea to an environment with two classes of target objects placed on the floor of an otherwise empty room. It uses a binary

belief state for each grid cell: a target is either absent or present. Detector evidence updates the target-presence belief map, and the policy scores candidate future views using an expected-free-energy-inspired objective that combines a preference term, expected information gain, and movement cost. Camera coverage is incorporated into the belief update so that observations with limited cell coverage have a weaker effect than observations with stronger coverage. This policy is intentionally simpler than the original saccade-agent model because it does not represent target classes internally. It uses a pragmatic de-fixation mechanism that prevents repeated fixation on cells that have already produced enough target detections. This mechanism is useful for the binary model, but it is not a fully class-aware account of when an object has been sufficiently classified.

The second AIF-inspired policy is the ZOI AIF saccade policy, developed for the table-enriched environments. This policy extends the binary model with a richer representation of both targets and context. Each cell maintains a categorical target belief over empty, unknown target, bottle, and can states. In addition, the policy maintains an explicit probabilistic belief that a cell belongs to a table surface. This table belief affects the prior probability that a target is present, reflecting the structure of the table-enriched environments, in which targets are much more likely to appear on desks than on the floor. The policy also inherits a key aspect of the original saccade-agent idea: peripheral observations can raise generic target belief without immediately resolving the class, while fixation-level observations can provide class-specific evidence. This supports a staged behaviour in which the drone is attracted by possible target evidence, moves to inspect it more directly, and then reduces the need to keep fixating once the object's class belief becomes confident.

The experimental design is organised around five main questions. The first is whether an AIF-inspired saccade policy can outperform naive exploration policies in a floor-only object-search task. This question is addressed in the environment containing bottles and cans on the floor, where the simple AIF saccade policy is compared with coverage-oriented baselines. The second question concerns detector quality: how learned detector variants affect downstream task performance compared with a ground-truth detector. This matters because the quality of visual evidence restricts the information available to the search policy. The third question concerns the effect of raised surfaces on search performance. The presence of tables changes where targets are likely to occur and alters the relationship between camera pose and visual coverage, because the camera footprint is smaller on raised surfaces. The fourth asks whether a zone-of-interest-enriched AIF saccade policy can exploit this structure through table beliefs, class-specific target states, and table-aware observation geometry. The fifth question examines how the policies are affected by visual complexity in the `mixed_hard` environment, which retains the table-enriched search task while introducing a visually complex floor and non-target

decoy objects that must not be interpreted as targets. This setting provides an additional benchmark for evaluating object search under more demanding visual conditions.

The thesis makes several contributions. It develops two AIF-inspired navigation policies for detector-driven drone search: a binary-state policy for the floor-only setting and a zone-of-interest extension for scenes with tables. The latter maintains categorical target beliefs together with perception-derived table beliefs, allowing the policy to adapt its target priors and planning decisions to the structure of the scene. These policies are supported by an experimental framework based on Webots that was developed for this work. A downward-facing Crazyflie model is coupled with detector integration, grid-based coverage estimation, and supervisor-validated target claims, allowing policy behaviour to be evaluated under consistent conditions. The framework also supports dataset generation, training, and evaluation for learned detection models. Alongside the floor-only and table scenarios, it includes the `mixed_hard` environment, which adds a visually complex floor and non-target decoy objects as an additional benchmark. Together, these components support the study of lightweight perception-aware navigation for autonomous object search in structured indoor environments.

The remainder of the thesis is organised as follows. Chapter 2 introduces the background needed to understand the project, including nano-drone definition and constraints, simulation-based robotics development, object detection for resource-constrained platforms, and the active inference concepts used in the policies. Chapter 3 positions the project with respect to nano-drone object-search benchmarks, active visual sensing, and perception-guided navigation. Chapter 4 describes in detail the methodology of the Webots environment, the controller and supervisor architecture, the detector pipeline, the coverage model, the navigation policies and the experimental settings. The results chapter reports the final evaluation outcomes across all environments, comparing naive policies, AIF-inspired policies, learned detectors, and ground-truth perception, while also discussing trajectories and the evolution of internal belief states where relevant. Finally, the conclusion summarizes the findings, discusses the limitations of the simulation-based approach, and outlines possible extensions toward more realistic environments, richer semantic contexts, and future real-world deployment.

Chapter 2

Background

The work developed in this thesis depends on several layers that have to be understood together. The drone platform defines the physical scale of the problem: it is small, mobile, and suitable for indoor search, but its limited payload and computation constrain what kind of sensing and decision-making can realistically be considered. Visual perception provides the evidence used by the search system, turning camera frames into object hypotheses that can be mapped back to the task space. Simulation then makes it possible to study this closed-loop interaction under repeatable conditions, where the same environment structure can be evaluated across different policies and detector settings. Finally, active inference provides the probabilistic motivation for policies that use observations not only as outputs of a trajectory, but as information that can shape where the drone should look next. This chapter introduces these foundations before the implemented system is described in Chapter 4.

2.1 Nano-Drone Hardware, Control, and Navigation

Autonomous search with a nano-drone is shaped by the physical limits of the platform. The vehicle must carry its sensors and computation while preserving stable flight, and every high-level decision must eventually be expressed as motion commands that the drone can execute. For this reason, the search policy cannot be separated entirely from the platform: camera placement, pose estimation, body-frame commands, and low-level control all determine what viewpoints can be reached and how visual evidence is acquired. This section introduces the nano-drone setting, the Crazyflie reference platform, and the control abstractions used later in the thesis.

Table 2.1: UAV taxonomy by vehicle class size

Vehicle class	∅ : Weight [cm:kg]	Power [W]	Onboard device class
standard-size	$\sim 50 : \geq 1$	≥ 100	Desktop
micro-size	$\sim 25 : \sim 0.5$	~ 50	Embedded
nano-size	$\sim 10 : \sim 0.01$	~ 5	MCU
pico-size	$\sim 2 : \leq 0.001$	~ 0.1	ULP

2.1.1 Nano-UAVs and Indoor Autonomous Search

UAVs span a range of platforms, from larger inspection drones to small-size nano-drones with highly constrained payload, energy, and computational resources. A useful UAV taxonomy by vehicle class size [7] is reported in Table 2.1. The onboard device classes range from desktop and embedded processors to MCUs and ULP platforms.

Within this taxonomy, this thesis considers nano-quadrotors: palm-sized aerial platforms whose limited payload, energy, and power budgets constrain the sensing and computation available onboard [7, 1]. Their relevance to indoor robotics derives from their ability to move through restricted spaces while carrying a camera across multiple viewpoints within the same environment. This makes nano-UAVs suitable platforms for inspection, monitoring, and object-search tasks [1, 8].

The same form factor also makes autonomous operation difficult. Limited payload and energy budgets constrain battery capacity, sensor selection, and onboard compute. In a visual-search system, flight control and obstacle avoidance must coexist with image acquisition, detector inference, and action selection. These functions must operate within limited memory, processing, timing, and power budgets [1, 9].

Indoor deployment additionally requires state estimation. To navigate towards a selected viewpoint and relate camera observations to locations in the room, the drone must maintain an estimate of its pose relative to the environment. This is distinct from obstacle perception: state estimation concerns the drone's own position and orientation, whereas obstacle perception concerns collision hazards in its vicinity. Both functions must be supported by onboard sensing and computation on a nano-UAV [9, 8].

Camera observations are partial and viewpoint-dependent. In camera-based indoor object search, a target's image position, apparent size, and visibility depend on the relative pose between the camera and the object, as well as on possible occlusions in the scene. The chosen motion determines which detector evidence is acquired, while new evidence can change which subsequent viewpoint is worth inspecting.

Geometric exploration is a legitimate baseline for object search strategies. A policy can spread observations across the room and react to nearby obstacles without maintaining an explicit search belief. However, reactive obstacle avoidance alone does not use tentative object detections to choose a better view. When movement selection is independent of detector evidence, a weak detection or newly observed contextual structure does not alter the next sensing action. The policy may continue its geometric exploration pattern rather than deliberately seeking a view that confirms, rejects, or classifies the observed target.

Active visual object search addresses this missing connection by maintaining a representation of the search state and using accumulated observations to prioritize subsequent sensing actions [10]. The environment studied in this thesis is not an unknown-map exploration problem: the room geometry and task grid are predefined. Instead, the relevant uncertainty concerns the presence and class of targets within the grid and, in the table scenarios, the location of contextual table surfaces. The proposed policies use detector evidence to change the relative priority of candidate cells and viewing directions.

2.1.2 Crazyflie Platform

The Crazyflie 2.1 [11] is an open-source flying development platform developed by Bitcraze for education, research, and prototyping. It is a palm-sized drone, with a takeoff weight of approximately 29 g and a motor-to-motor size of about 92 mm, making it representative of the class of nano-UAVs considered in this thesis. Its recommended payload limit of 15 g constrains the sensing and computing hardware that can be carried onboard. Its relevance comes from providing a compact and extensible nano-drone architecture with flight-control hardware, onboard sensing, and support for expansion decks.

The core Crazyflie 2.1 hardware includes an STM32F405 microcontroller [11] as the main application processor, an additional radio and power-management microcontroller, an IMU, a pressure sensor, radio communication, Bluetooth Low Energy support, and a small onboard battery. The STM32 processor is responsible for the main embedded application logic and low-level flight-related computation, while the onboard sensors support stabilization and state estimation. The platform can also be extended through Bitcraze expansion decks, which add capabilities such as optical flow, ranging, positioning, and embedded vision [11].

For visual perception and embedded AI experiments, the Crazyflie ecosystem includes the AI-deck 1.1 expansion board [12]. The AI-deck adds a GAP8 ultra-low-power RISC-V processor, a Himax low-power monochrome camera, external memory, and Wi-Fi connectivity. This provides an example of the additional, but still constrained, onboard processing available within the Crazyflie ecosystem. Such hardware can support lightweight vision and embedded-AI experiments, but model

Figure 2.1: Bitcraze Crazyflie 2.1 nano-UAV. Official product image reproduced from Bitcraze documentation [11].

size, inference rate, memory use, and power consumption remain important design constraints [1].

In this thesis, the Crazyflie serves as a hardware reference for a simulated drone. The simulated model is adapted with a downward-facing camera so that the FoV can be projected onto the environment and related to grid-cell coverage. The simulation preserves the task-level coupling between continuous aerial motion, camera viewpoint, and detector observations, but it does not directly reproduce onboard processor performance. Pose is supplied by the simulator, so onboard localisation is not implemented or evaluated. The Crazyflie platform provides a concrete nano-drone reference point for evaluating detector-based object search and lightweight active-sensing policies in simulation.

2.1.3 Quadcopter Pose, Coordinate Systems, and PID Control

The pose of a quadcopter is described by its position and orientation in three-dimensional space [13]. The translational position of the vehicle in the world frame is written as

$$\mathbf{p}_W = \begin{bmatrix} x & y & z \end{bmatrix}^\top . \tag{2.1}$$

where x, y, and z denote the position of the drone with respect to a fixed environment coordinate system. Its attitude can be represented using Euler angles,

$$\boldsymbol{\eta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^\top . \tag{2.2}$$

where ϕ is roll, θ is pitch, and ψ is yaw. Roll and pitch describe the inclination of the drone body, while yaw describes its heading around the vertical axis.

For navigation, it is important to distinguish between the world frame and the body frame. The world frame is fixed to the environment and is used to describe positions, trajectories, and object locations. In this thesis setting, the simulated drone pose, target objects, tables, and room geometry are all represented relative to this frame. The body frame, by contrast, is attached to the drone and rotates with it. Motion commands such as forward velocity, sideways velocity, and yaw rate are naturally expressed relative to the drone's current orientation.

The planar navigation abstraction used in this thesis relies primarily on yaw to relate body-frame motion to displacement in the environment. This is appropriate because the drone is regulated around a fixed altitude and the navigation policies reason about horizontal motion and viewing direction rather than full three-dimensional trajectory generation. Yaw also determines the orientation of the downward-facing camera footprint projected onto the environment.

Low-level flight control is typically separated from high-level navigation. The high-level layer specifies desired motion, altitude, or yaw commands, while the lower-level controller tracks these setpoints and stabilizes the vehicle. A common feedback mechanism for this purpose is proportional–integral–derivative (PID) control [14]. For a scalar reference signal $r(t)$ and measured output $y(t)$, the tracking error is

$$e(t) = r(t) - y(t). \tag{2.3}$$

The PID controller computes a control signal

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}. \tag{2.4}$$

where K_p, K_i, and K_d are the proportional, integral, and derivative gains. The proportional term reacts to the current error, the integral term compensates for

accumulated steady-state error, and the derivative term reacts to the rate of error variation.

Multirotor controllers commonly use cascaded feedback structures, in which outer loops regulate quantities such as position, velocity, or altitude, while inner loops regulate attitude and angular rates [13, 15]. In the simulated controller used in this thesis, navigation policies ultimately provide desired forward velocity, sideways velocity, and yaw rate. These commands are translated into motor commands through a fixed-height feedback controller, allowing the policy-level problem to focus on horizontal motion and viewing direction.

2.2 Simulation for Autonomous Robotics

In an autonomous search task, performance depends on the full loop between movement, sensing, detection, and action selection. During a run, the drone trajectory determines the camera views, the camera views determine the detector evidence, and that evidence may affect the following movement decisions. A useful experimental setup must therefore preserve the closed-loop nature of the task while still allowing controlled comparison between different policies. Simulation provides this role in the thesis: it gives access to repeatable scene generation, consistent sensor and detector inputs, and supervisor-side ground truth for evaluating the outcome of each run. This section introduces simulation as an experimental tool for robotics and then describes the Webots framework used for the project.

2.2.1 Role of Simulation in Robotics Development

Simulation is widely used in robotics development because it provides controlled environments in which algorithms can be designed, tested, and compared before deployment on physical hardware [16]. This is valuable across robotics domains because real-world experiments can be time-consuming, expensive, difficult to reproduce, and potentially dangerous. A simulated environment allows researchers to iterate over robot controllers, sensor configurations, object placements, control policies, and evaluation protocols without requiring manual reconstruction of the experimental setup after every run.

One important role of simulation is to accelerate the development of robotic control and decision-making systems. Collecting large amounts of interaction data on physical platforms can require substantial time, labour, and hardware maintenance. Simulation can instead generate varied trajectories, sensor observations, and labelled data at a scale that would be impractical to obtain through repeated physical experiments [17]. This is particularly important for learning-based robotics, where control policies or perception systems may require substantial experience before they become reliable. Even when a final system is not trained end-to-end in

simulation, simulated environments remain useful for testing design choices and identifying failure cases before physical deployment.

A second key advantage of simulation is repeatability. In physical experiments, even small differences in initial conditions, lighting, sensor noise, or object placement can make controlled comparisons difficult. In simulation, these factors can be fixed, randomised systematically, or recorded through known seeds. This makes it possible to compare algorithms under matched conditions and to separate changes in algorithm behaviour from changes in the experimental setup. For aerial object-search tasks, repeatability is particularly important because performance depends on the interaction between the drone trajectory, camera FoV, detector outputs, and target locations. In this thesis, the simulator also retains exact scene annotations, which are used to generate detector labels and to validate target claims after a run. A separate simulator-derived ground-truth detector condition is used deliberately as a best-case perception benchmark.

Simulation does not remove the need for caution when interpreting results. Simulated sensors, dynamics, lighting, textures, and object appearances may differ from their physical counterparts. This discrepancy is commonly referred to as the sim-to-real gap [17, 18]. A policy or detector that performs well in simulation may not transfer directly to physical hardware if it relies on idealized sensor behaviour, simplified dynamics, or visual features that do not generalize. Performance in simulation therefore does not, by itself, establish comparable performance on physical hardware.

In this thesis, simulation is used as an experimental framework for studying autonomous nano-drone object search under controlled conditions. It enables repeatable comparison of policies, learned detector variants, and a simulator-derived ground-truth detector condition. It also makes it possible to evaluate how changes in environment structure, such as the introduction of table surfaces, affect visual coverage and downstream search performance. In this project, learning is limited to the object-detection models. The role of simulation is to provide a precise and reproducible setting in which the perception, navigation, and evaluation components of the system can be developed and analysed.

2.2.2 Webots Simulation Framework

Webots is a robotics simulation framework used to model, program, and simulate robotic systems in three-dimensional virtual environments [19]. It provides an integrated environment in which a user can define a scene, add robots and objects, assign physical and visual properties, attach sensors and actuators, and run controller programs that interact with the simulated world. In the context of this thesis, Webots provides the simulation layer used to implement the drone environment, the camera-based perception pipeline, the object-spawning logic, and

the evaluation infrastructure.

A Webots simulation is defined by a world file. Conceptually, a world is organised as a scene tree composed of nodes. Nodes describe elements of the simulated environment, including world settings, lights, viewpoints, shapes, solid objects, robots, sensors, and actuators. Each node has fields that specify properties such as position, orientation, geometry, appearance, mass, or controller assignment. Objects can be defined through reusable `PROTO` models, which package a node structure into a parameterized object that can be instantiated multiple times [20].

Robots in Webots are represented by `Robot` nodes. A `Robot` node is a physical object in the scene tree associated with a controller program that reads simulated sensors and sends commands to actuators. Controllers can be written in several programming languages, including Python, C, C++, Java, and MATLAB. During a simulation, the controller interacts with the robot through the Webots controller API, for example by reading camera images and sensor measurements or commanding motors. In the default synchronised execution mode, the controller advances by repeatedly calling the `step` function; Webots advances the simulation by the corresponding control interval before returning control to the controller [20].

Webots also supports `Supervisor` nodes, which are special robot nodes with additional access to the simulation world. A supervisor can inspect or modify elements of the scene tree, reset objects, spawn or remove nodes, read object poses, and coordinate experimental runs. This capability separates the autonomous robot controller from the experimental infrastructure. The robot controller can operate from its controller-side sensor and detector inputs, while the supervisor manages scenario generation, ground-truth validation, run termination, and evaluation reporting [20]. Figure 2.2 summarizes the separation between the simulated world, the robot controller, and the supervisor controller in Webots.

The simulator combines physics simulation and rendering. Physical properties such as mass, collision geometry, friction, and joints determine how objects interact under the physics engine. At the same time, the rendering system produces visual observations for simulated cameras, including the effects of geometry, textures, lighting, viewpoint, and occlusion.

2.3 Neural Networks and Object Detection

Visual perception provides the evidence that connects camera observations to task-level search decisions. In this thesis, that role is assigned to object detectors: learned models identify bottles, cans, and, in the table environments, desks. Their outputs are interpreted by the controller, mapped to grid cells, and used either to generate target claims or to update the beliefs of the active-sensing policies. The navigation policies themselves remain explicitly defined, so the role of neural

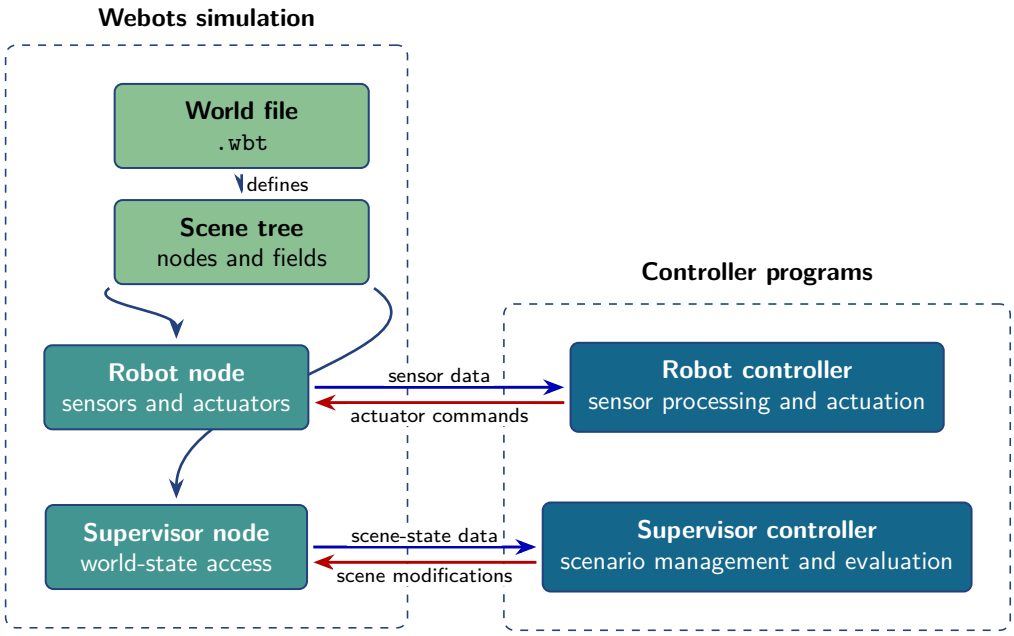

Figure 2.2: Webots world, robot-controller, and supervisor-controller interactions. A `.wbt` world file defines a scene tree containing robot and supervisor nodes. Robot controllers receive sensor data and send actuator commands. Supervisor controllers read scene-state information and can apply scene modifications through the Supervisor API [20].

networks in this thesis is concentrated on perception. The following subsections introduce the basic supervised-learning formulation behind these detectors and then focus on lightweight object detection models suitable for constrained robotic settings.

Neural networks are parameterized functions that learn mappings from input data to output predictions by adjusting their parameters from example data. A model with parameters θ can be written abstractly as

$$\hat{\mathbf{y}} = f_\theta(\mathbf{x}), \tag{2.5}$$

where $\mathbf{x}$ is an input, such as an image or sensor measurement, and $\hat{\mathbf{y}}$ is a predicted output, such as a class label, bounding box, or continuous control command.

In supervised learning, the model is trained from a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ by minimizing an empirical loss,

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f_\theta(\mathbf{x}_i), \mathbf{y}_i\right). \tag{2.6}$$

Here, $\mathbf{y}_i$ denotes the target output associated with input $\mathbf{x}_i$. Modern deep neural

networks compose multiple learned transformations and nonlinear activation functions, allowing them to learn useful representations from high-dimensional data such as images [21].

2.3.1 Neural Networks for Perception and Control

For autonomous robotics, two roles are particularly relevant: perception and control. In perception, neural networks process sensor data into task-relevant estimates. In visual robotics, this includes image classification, object detection, semantic segmentation, and depth estimation [22]. For object search, a detector transforms a camera image into structured predictions such as object classes, image-space bounding boxes, and confidence scores. These predictions can then be used by downstream navigation or decision-making modules.

Neural networks can also be used for control and navigation. They may map observations directly to actions, estimate value functions, predict future states, or provide learned components within planning systems. An end-to-end visuomotor policy, for example, can map image observations directly to control commands, while RL policies can learn behaviour through interaction with an environment and an optimization objective [23, 24]. These methods can learn complex relationships between visual observations and control decisions. However, their behaviour depends on the training data, the learning objective, and how closely deployment conditions resemble the conditions encountered during training.

This thesis uses neural networks for perception rather than for learning the navigation policy itself. The learned detector models process simulated camera images to predict target objects and, in the table environments, desks. A separate simulator-derived ground-truth detector condition is used only as a perception reference and is not a learned model. The navigation policies are explicitly defined algorithms rather than end-to-end learned visuomotor policies.

2.3.2 Lightweight Object Detection

Object detection is the computer-vision task of identifying and localising object instances in an image. Unlike image classification, which assigns one or more labels to an entire image, an object detector returns a set of predictions associated with individual image regions. Each detection typically contains a predicted class, a bounding box, and a confidence score. For an input image $\mathbf{x}$, the output of a detector can be represented as

$$\mathcal{D}(\mathbf{x}) = \{d_j\}_{j=1}^{N}, \qquad d_j = (\ell_j, \mathbf{b}_j, s_j), \tag{2.7}$$

where ℓ_j is the predicted class of detection j, $\mathbf{b}_j$ is its bounding box, $s_j \in [0,1]$ is its reported confidence score, and N is the number of detections returned for the

image. In robotic object search, these detections can be converted into task-level information such as target claims, spatial maps, or policy observations.

On a nano-UAV, visual perception competes with flight control, sensing, communication, and navigation for limited computational and energy resources. The relevant question is not only whether a detector recognizes objects accurately, but whether it can produce useful observations frequently enough for a moving camera and a time-limited search mission. Lamberti et al. [1] illustrate this constraint in an onboard nano-drone system that combines object detection with autonomous exploration. When detector updates are sparse relative to drone motion, an object can enter and leave the camera FoV without producing usable evidence for the search controller.

Modern object detectors are commonly grouped into two-stage and single-stage architectures. Two-stage methods, such as Faster R-CNN [25], first generate candidate image regions and then classify and refine them. Single-stage methods, such as SSD [26] and YOLO [27], predict object locations and classes directly from the image in one network evaluation. This avoids a separate region-proposal stage and is often preferable when computational cost and response time are important.

Lightweight detectors typically combine an efficient feature extractor with single-stage prediction heads. Computational cost can be further reduced through model scaling, quantization, pruning, or architecture-specific optimizations, such as early-exit mechanisms [28]. These approaches are relevant to nano-drone vision because the detector must provide usable evidence without monopolizing the resources required by the rest of the system.

In this thesis, learned detectors form the perception component of the search system. They transform simulated camera frames into detections of target objects and, in the table environments, desks. The controller then converts the retained detections into class labels, confidence values, bounding-box locations, mapped grid cells, and target claims. The detector therefore provides the interface between visual sensing and the higher-level search policy.

The following subsections introduce the two detector families used in the experiments: MobileNetV2-SSD and YOLO.

2.3.3 MobileNetV2-SSD

MobileNetV2-SSD combines an efficient convolutional feature extractor with a single-stage detection architecture. MobileNetV2 [29] extracts visual features from an input image, while SSD [26] predicts object classes and bounding boxes from selected feature maps. This combination is relevant to embedded vision because it avoids the separate region-proposal stage used by two-stage detectors and uses a backbone designed to reduce computational cost.

MobileNetV2 is a CNN architecture designed for mobile and embedded vision

applications. Its central component is the inverted residual block with a linear bottleneck. Each block begins with a narrow feature representation, expands it into a higher-dimensional intermediate space, applies depthwise convolution, and then projects the result back to a narrow output representation. Residual connections link compatible input and output bottlenecks.

The efficiency of MobileNetV2 relies partly on depthwise separable convolution, originally introduced for MobileNet [30]. A conventional convolution jointly performs spatial filtering and channel mixing. Depthwise separable convolution separates these operations: spatial filtering is applied independently to each channel, and pointwise convolution then combines channel information. This substantially reduces computational cost relative to a conventional convolution with the same input and output dimensions.

SSD [26] predicts detections directly from convolutional feature maps in a single forward pass. It defines default boxes with different scales and aspect ratios at selected feature-map locations. For each default box a_k, the network predicts a localisation adjustment and class-score vector,

$$(\Delta \mathbf{b}_k, \mathbf{p}_k)\,, \tag{2.8}$$

where $\Delta \mathbf{b}_k$ encodes the adjustment from the default box to the predicted bounding box and $\mathbf{p}_k$ contains class scores. Predictions are generated from feature maps with different spatial resolutions, allowing SSD to detect objects at different image scales.

For the floor-only experiments, this thesis includes a MobileNetV2-SSD detector reconstructed in PyTorch to approximate the MobileNet-SSD configuration and computational scale reported by Lamberti et al. [1]. It provides a detector condition directly connected to the nano-drone platform and onboard perception setting that motivated this project. The common detector-processing cadence used during evaluation is described in the methodology chapter.

2.3.4 YOLO Models

YOLO [27] is a family of single-stage object detectors designed around efficient real-time detection. The original YOLO formulation treated object detection as a direct regression problem from an input image to bounding boxes and class probabilities. Rather than generating object proposals and classifying them in separate stages, the detector predicts localisation and classification outputs through one network evaluation.

Modern YOLO models differ substantially from the original architecture, but retain the same broad principle. They extract image features, combine information across multiple spatial scales, and produce object-class and bounding-box predictions in a single detection pipeline. Multi-scale features matter because the same

target can occupy very different image areas as the camera viewpoint changes.

This thesis uses the Ultralytics YOLO11 family [31] as its main learned-detector family. The floor-only experiments compare YOLO11-nano and YOLO11-small models trained to detect bottles and cans. The table-enriched and hard experiments use YOLO11-small models trained with an additional `desk` class, since desk detections are used to infer table-surface locations as well as to identify target objects.

2.4 Active Inference and Expected Free Energy

The policies developed in this thesis are inspired by active inference because the search problem is fundamentally about choosing actions that change future observations. A drone looking for objects should choose viewpoints that are expected to clarify uncertain parts of the task. Active inference provides a probabilistic language for this idea: the agent maintains beliefs about hidden states, updates those beliefs from observations, and evaluates actions through the observations they are expected to produce. The following subsections introduce the free-energy and expected-free-energy concepts needed to understand the later policy formulations.

2.4.1 Variational Free Energy

The Free-Energy Principle [32] is a broad theoretical account of perception, learning, and action based on probabilistic inference. At its broadest, it relates adaptive behaviour to biological self-organisation: an organism must remain within a limited range of sensory states over time, and perception and action are interpreted as processes that help maintain those conditions. The computational treatment used in this thesis focuses on belief formation: an agent maintains beliefs about the hidden causes of its observations and updates those beliefs as new sensory evidence becomes available.

Consider the discrete-state setting used in this thesis. Let an agent receive an observation o generated by a hidden state s. The agent is assumed to possess a generative model of the relation between hidden states and observations,

$$p(o, s) = p(o \mid s)p(s), \tag{2.9}$$

where $p(s)$ is the prior distribution over hidden states and $p(o \mid s)$ is the likelihood of observing o when the hidden state is s.

Given an observation, the ideal Bayesian posterior is

$$p(s \mid o) = \frac{p(o \mid s)p(s)}{p(o)}, \tag{2.10}$$

where

$$p(o) = \sum_s p(o, s) \tag{2.11}$$

is the model evidence, also called the marginal likelihood. The negative logarithm of the model evidence,

$$-\ln p(o), \tag{2.12}$$

is known as surprisal. An observation has high surprisal when it is assigned low probability by the generative model.

For models with many possible hidden states, evaluating the marginal likelihood exactly may be impractical because it requires summing over all of them. Variational inference addresses this issue by introducing an approximate posterior distribution $q(s)$. The corresponding variational free energy is

$$F[q; o] = \mathbb{E}_{q(s)} \left[\ln q(s) - \ln p(o, s)\right]. \tag{2.13}$$

This quantity can be rewritten as

$$F[q; o] = D_{\mathrm{KL}} \left(q(s) \parallel p(s \mid o)\right) - \ln p(o), \tag{2.14}$$

where D_{KL} is the Kullback–Leibler divergence. Since the Kullback–Leibler divergence is non-negative,

$$F[q; o] \geq -\ln p(o). \tag{2.15}$$

For a fixed observation, the surprisal term is fixed. Minimizing variational free energy therefore reduces the discrepancy between the approximate belief $q(s)$ and the Bayesian posterior $p(s \mid o)$. In practical terms, the agent updates its beliefs so that they better explain the observed sensory data under its generative model [33, 5].

Variational free energy can also be written as

$$F[q; o] = D_{\mathrm{KL}} \left(q(s) \parallel p(s)\right) - \mathbb{E}_{q(s)} \left[\ln p(o \mid s)\right]. \tag{2.16}$$

The first term measures the change from the prior belief to the approximate posterior, while the second measures how well the inferred state explains the observation. This form is commonly described as a balance between belief complexity and prediction accuracy.

In the later policies, beliefs are represented explicitly as discrete probability distributions over task-relevant hidden states. The controller updates them directly from specified priors and likelihood models when new observations are received. The derivation above provides the probabilistic interpretation of this procedure: each update revises the current estimate of the hidden task state in light of new evidence.

2.4.2 Active Inference

Active inference extends belief updating to action selection. Instead of only updating beliefs after receiving observations, an active-inference agent also evaluates actions according to the observations they are expected to make available. Perception updates beliefs about hidden states, while action changes the evidence that the agent is likely to receive in the future [33, 34].

In discrete active-inference models, the generative model is often described through three components. The observation model A represents the likelihood $p(o_t \mid s_t)$, the transition model B^{a_t} represents the state transition probability $p(s_{t+1} \mid s_t, a_t)$, and the preference model C encodes prior preferences over outcomes [5].

Given an observation at time t, perceptual inference can be expressed as

$$q^*(s_t) = \arg\min_{q(s_t)} F_t[q], \tag{2.17}$$

where $F_t[q]$ is the variational free energy associated with the current observation. The resulting belief state represents the agent's estimate of the hidden variables that best explain the available evidence.

In a general active-inference model, actions can affect both future hidden states and future observations. The object-search setting considered in this thesis is more specific. The hidden target state of a grid cell is treated as static while a candidate action is evaluated. Selecting a target cell and yaw does not change whether an object is present. Instead, it changes the future camera viewpoint, the coverage of grid cells, and the expected reliability of the resulting visual evidence. The relevant action dependence therefore appears primarily in the observation model conditioned on the action, $P(o' \mid s', a)$, where a denotes a candidate sensing action.

Preferences specify which observations would be useful for the task. In a visual-search setting, observations corresponding to target detection or confident class resolution can be preferred over observations that leave the state uncertain. These preferences do not directly determine an action. They are combined with the agent's current beliefs and predicted future observations to evaluate which candidate viewpoint is worth selecting.

This introduces two complementary aspects of action selection. An action can be valuable because it is expected to support a preferred task outcome, or because it is expected to reduce uncertainty about a hidden state. Expected free energy provides a formal way to combine these two considerations.

2.4.3 Expected Free Energy

Expected free energy is used in active inference to score the anticipated consequences of future actions. Formal discrete-state formulations usually define a policy π as a

sequence of actions over a future time horizon [5]. The policies developed in this thesis rank the immediate next sensing decision and re-evaluate after new evidence becomes available. The following discussion therefore uses a to denote a candidate action.

To evaluate an action, the agent predicts the observations that it may produce. Let $Q(s' \mid a)$ denote the predictive belief over a future hidden state and let $P(o' \mid s', a)$ denote the observation model conditioned on the action. The predicted distribution over future observations is then

$$Q(o' \mid a) = \sum_{s'} P(o' \mid s', a) Q(s' \mid a). \tag{2.18}$$

In a general active-inference model, an action may affect both future hidden states and future observations. In the object-search formulation used in this thesis, target states are treated as static over one sensing decision. The main role of the action is therefore to change the viewpoint, the visible area of the environment, and the reliability of the resulting observation model.

Expected free energy combines pragmatic and epistemic considerations. The epistemic component is called epistemic value. It measures how much an action is expected to reduce uncertainty about hidden states. For discrete states and observations, it can be expressed as expected information gain,

$$\mathrm{IG}(a) = \sum_{o'} Q(o' \mid a) D_{\mathrm{KL}} \left(Q(s' \mid o', a) \parallel Q(s' \mid a) \right). \tag{2.19}$$

Here, $Q(s' \mid a)$ is the predictive belief before observing the future outcome, while $Q(s' \mid o', a)$ is the belief that would result after observing o'. The information-gain term is therefore high when an action is expected to produce observations that substantially change the agent's belief about hidden states [34, 5].

Preferences encode which observations would be desirable for the task. Let $P(o' \mid C)$ denote a prior preference distribution over future observations. Under the usual posterior-consistency approximation, expected free energy can be expressed as

$$G_{\mathrm{EFE}}(a) \approx \mathbb{E}_{Q(o' \mid a)} \left[-\ln P(o' \mid C) \right] - \mathrm{IG}(a). \tag{2.20}$$

The first term is an expected preference cost. It is low when the action is expected to produce observations preferred by the agent. The second term rewards actions that are expected to reduce uncertainty. This form expresses expected free energy as a balance between preferred task outcomes and information gain [34, 5].

Expected free energy can also be expressed through a risk and ambiguity decomposition,

$$G_{\mathrm{EFE}}(a) = D_{\mathrm{KL}} \left(Q(o' \mid a) \parallel P(o' \mid C) \right) + \mathbb{E}_{Q(s' \mid a)} \left[\mathcal{H} \left(P(o' \mid s', a) \right) \right], \tag{2.21}$$

where $\mathcal{H}$ denotes entropy. The risk term measures the mismatch between predicted and preferred observations, while ambiguity measures the expected uncertainty of observations under the likelihood model [34, 5].

The precise action scores used by the policies are introduced in the methodology chapter. They are adaptations inspired by active inference that combine preference or utility terms specific to the task, expected information gain, and movement cost when selecting the next target cell and viewing direction.

Chapter 3

Related Work

The work in this thesis is positioned between lightweight nano-drone search and broader approaches to active visual navigation. Its closest practical reference is the use of compact detectors and simple exploration rules on Crazyflie-like platforms, where the main challenge is to obtain useful object detections under severe resource constraints. The policies developed here keep that low-complexity setting, but change the role of perception: detections are not only used to report objects encountered along a trajectory, but also to update beliefs that influence where the drone should move next.

This connects the thesis to active visual search, where sensing actions are selected according to the information they are expected to provide. It also relates to semantic object navigation and learned navigation policies, although the problem studied here is deliberately narrower. The room geometry and task grid are predefined, the object categories are limited, and navigation is implemented through explicit belief updates rather than an end-to-end learned policy. The related work is used to place the project in this intermediate space: more perception-aware than detector-agnostic exploration, but more compact and interpretable than general semantic navigation.

3.1 Nano-Drone Object Search with Lightweight Detection

The closest practical reference for this thesis is the nano-drone object-search work of Lamberti et al. [1]. Their study combines a Crazyflie nano-drone, lightweight onboard object detection, and simple autonomous exploration policies in an indoor search task. This thesis takes that work as the starting point for the simulated evaluation setting: the room scale, the Crazyflie-based drone model, and the detector-agnostic baseline policies are adapted from the same object-search scenario.

In Lamberti et al., the navigation policies are deliberately lightweight. The drone follows simple exploration behaviours, while the detector is used to identify target objects encountered during flight. This structure is important because it provides a realistic low-complexity reference for nano-drone search: the navigation strategy does not rely on dense semantic mapping, learned visuomotor control, or expensive planning, but still produces task-level object-finding behaviour when combined with a compact detector.

The baseline policies in this thesis preserve this low-complexity role. They are not intended as state-of-the-art navigation methods; they define the performance obtained when the drone explores using simple motion rules and target detections are used only to produce claims. Reimplementing these policies in Webots makes it possible to compare the original style of detector-assisted exploration with policies in which detections are also used to update an internal search belief.

The main difference introduced in this thesis is at the policy level. Instead of treating detector outputs only as events produced along a trajectory, the AIF-inspired policies convert them into observations over grid-cell states. This allows the navigation policy to prefer locations that are uncertain, likely to contain a target, or useful for resolving a previously observed object. The comparison with the recreated baseline policies tests whether this additional belief structure improves search performance in the same kind of constrained nano-drone task.

Related work on nano-drone perception has also investigated how to reduce the cost of running detectors onboard. BlankSkip [28], for example, reduces average detector computation by skipping full detection on frames predicted to contain no relevant object. This direction is complementary to the present thesis. It focuses on reducing the cost of producing detections, while the work here focuses on how detections, once produced, can be used by the search policy.

3.2 Active Visual Search and Belief-Guided Sensing

In visual search, movement is part of the sensing process. The agent does not only receive images; it selects the viewpoints from which future images will be acquired. This idea is central to active perception, where sensing is treated as an action-dependent process rather than a passive stream of measurements [3]. For object search, the implication is that a good action is not necessarily the one that covers the largest area, but the one that is expected to produce useful evidence for the current task.

Active visual object search extends this principle to robotic systems that must locate objects under uncertainty. Aydemir et al. [10], for example, use uncertain semantic information to guide object search in indoor environments. Their work

is broader than the setting considered here, since it addresses semantic search in unknown environments, but it establishes an important idea for this thesis: object search can be formulated as a decision problem over uncertain spatial and semantic information, not only as a geometric coverage problem.

A related line of work formulates active visual search through explicit planning under partial observability. POMP++ [35] models active visual search in unknown indoor environments using a Partially Observable Monte Carlo Planning framework, with online belief updates and a dynamically growing state space. This type of approach is more general than the grid-based policy used in this thesis, since it addresses object search in larger embodied-navigation benchmarks and reasons over map growth during exploration. Its relevance here is the shared emphasis on belief-guided search: the next action is selected using the current estimate of where useful visual evidence may be obtained.

The most direct conceptual inspiration for the policies developed in this thesis is the active-inference saccade-agent model of Vyas et al. [6]. In that model, an agent controls fixation over a discretized visual field and maintains beliefs over hidden object and class states. Different observation regimes distinguish regions outside vision, peripheral evidence, and fixation-level evidence, allowing the agent to explore when uncertain and to fixate regions where class information is expected to be resolved. The model connects active sensing, object classification, and expected information gain in a compact discrete setting.

Adapting the saccade-agent idea to the drone setting changes the relation between action and observation. In the original model, fixations are selected over a discretized visual field. In this thesis, the policy also reasons over a two-dimensional grid, but the camera FoV is displaced by continuous drone motion and may cover each grid cell only partially. This makes the observation model depend on projected camera coverage and on the physical cost of moving to a new viewpoint. The idea inherited from the saccade model is the use of uncertain object evidence to guide the next sensing action; the action space and observation geometry are adapted to the drone search problem.

Active inference has also been applied to robotic active vision outside the saccade-agent setting. Van de Maele et al. [36] formulate active vision for a robotic manipulator through the Free Energy Principle, using a learned generative model and expected free energy to select viewpoints for an object-reaching task. Their setting differs substantially from the drone search task considered here, but it provides an important connection between active-inference theory and robotic viewpoint selection. In both cases, actions are evaluated through the observations they are expected to make available.

3.3 Semantic Object Navigation and Learned Navigation Policies

A broader line of related work studies object navigation in embodied environments. ObjectNav is commonly defined as the task of navigating to an instance of an object category in an unexplored environment [37]. Compared with the drone-search task considered in this thesis, ObjectNav usually involves a wider range of scenes, object categories, and navigation decisions. The agent may need to explore an unknown layout, recognize semantic categories, decide when the target has been reached, and generalize across different environments. The common element is that object-related visual evidence is tied to action: the agent's movement determines what it can observe, and the observations collected during navigation affect what actions are useful next.

Many strong ObjectNav approaches use modular semantic representations. Goal-Oriented Semantic Exploration [38], for example, builds an episodic semantic map and uses it to guide exploration toward the goal object category. In this type of method, visual observations are not treated as isolated detections. They are accumulated into a structured representation of the scene, which then supports longer-horizon navigation toward semantic goals.

The representation used in this thesis is much smaller and more task-specific. The room geometry and task grid are predefined, and the policy does not build a full semantic map of an unknown environment. The relevant uncertainty is restricted to the search task: whether a grid cell contains a target, which class the target belongs to, and, in the table environments, whether a cell belongs to a raised surface. This places the method between detector-agnostic exploration and general semantic navigation: it uses perception to guide motion, but only through a compact belief state tied to the task grid.

Learned navigation policies provide another possible approach. End-to-end visuomotor policies can learn direct mappings from observations to actions [23], and RL has been widely studied for robotic control and navigation [24]. Such methods can learn complex behaviours from data, but their performance depends on the training objective, the amount and diversity of experience, and the match between training and evaluation conditions. For the type of comparison pursued here, this would make the analysis less transparent: detector quality, coverage geometry, movement cost, and policy memory would all be absorbed into the training process and the learned representation.

The approach followed in this thesis remains deliberately explicit. Learned models are used for object detection, while navigation is defined through interpretable policy rules and belief updates. This makes it possible to compare detector-agnostic baselines, learned detectors, ground-truth perception, and belief-guided action

selection within the same evaluation pipeline. The resulting work is narrower than general ObjectNav, but it isolates a specific question relevant to constrained drone search: whether a compact task belief can improve object-search behaviour over lightweight exploration alone.

Chapter 4

Methodology

This chapter describes the implemented system used to evaluate active-inference-inspired navigation for autonomous nano-drone object search. The system is built around a closed-loop interaction between flight, visual perception, task-level representation, policy selection, and supervisor validation. A simulated Crazyflie moves continuously through the Webots environment, while camera observations and object detections are projected onto a discrete grid used for coverage estimation, belief updates, target claims, and evaluation.

The purpose of the methodology is to make explicit how continuous drone motion is connected to the symbolic quantities used by the search policies. The room geometry, camera footprint, detector interface, controller-supervisor communication, and policy state representations all have to be defined consistently, because each component affects what the drone can observe and how those observations influence future motion. The chapter first defines the environment and task representation, then describes the coverage and perception pipelines, the closed-loop execution architecture, the implemented navigation policies, and the evaluation protocol.

4.1 Simulation Environment and Task Definition

The search task is defined in a simulated indoor room, but the quantities used by the controller are not all represented in the same space. The drone flies in continuous Webots coordinates, while the search objective is expressed over a finite grid of possible target locations. This separation is central to the methodology: continuous motion determines the camera viewpoint, but the resulting visual evidence must be mapped into a representation where coverage, beliefs, claims, and metrics can be updated consistently. This section defines the room, the task grid, the drone-camera configuration, and the object categories used across the evaluated environments.

4.1.1 Continuous Room and Grid Abstraction

The simulated search environment is a rectangular indoor room with width 5.5 m and length 6.5 m. Its metric coordinates are centred on the Webots world origin, such that

$$x \in [-2.75,\, 2.75], \qquad y \in [-3.25,\, 3.25]. \tag{4.1}$$

For task-level reasoning, the room is partitioned into square cells with side length

$$\Delta = 0.5\,\mathrm{m}. \tag{4.2}$$

This produces an 11×13 grid containing

$$N_{\text{cells}} = 11 \times 13 = 143 \tag{4.3}$$

cells. A cell is denoted by (c_x, c_y), where c_x identifies the column and c_y identifies the row. The centre of cell (c_x, c_y) is

$$\mathbf{p}_{c_x,c_y} = \begin{bmatrix} x_{\min} + \left(c_x + \frac{1}{2}\right)\Delta \\ y_{\min} + \left(c_y + \frac{1}{2}\right)\Delta \end{bmatrix}. \tag{4.4}$$

The grid is a symbolic task representation rather than a restriction on flight motion. The drone state and its camera pose remain continuous in the Webots world frame, while cells provide a common spatial reference for camera coverage, detector-output projection, target claims, policy beliefs, and evaluation reports. For instance, a target claim associated with cell (c_x, c_y) denotes a detector-derived assertion about the corresponding region of the room.

This representation also separates the geometry of flight from the spatial granularity used for search. The controller can estimate the fraction of each cell covered by the projected camera footprint, while the navigation policies can reason over candidate cells and viewing directions without treating the drone motion itself as cell-to-cell teleportation. Figure 4.1 illustrates this relationship between the continuous room, the discrete task grid, the drone pose, and the projected camera footprint.

4.1.2 Drone, Camera, and Scene Object Categories

The simulated vehicle is a Crazyflie-derived nano-drone model equipped with a downward-facing camera. Flight is constrained to a fixed nominal altitude of

$$z_{\text{flight}} = 1.5\,\mathrm{m}. \tag{4.5}$$

This configuration makes the camera footprint directly projectable onto the room floor and, where applicable, onto raised desk surfaces. The drone moves continuously

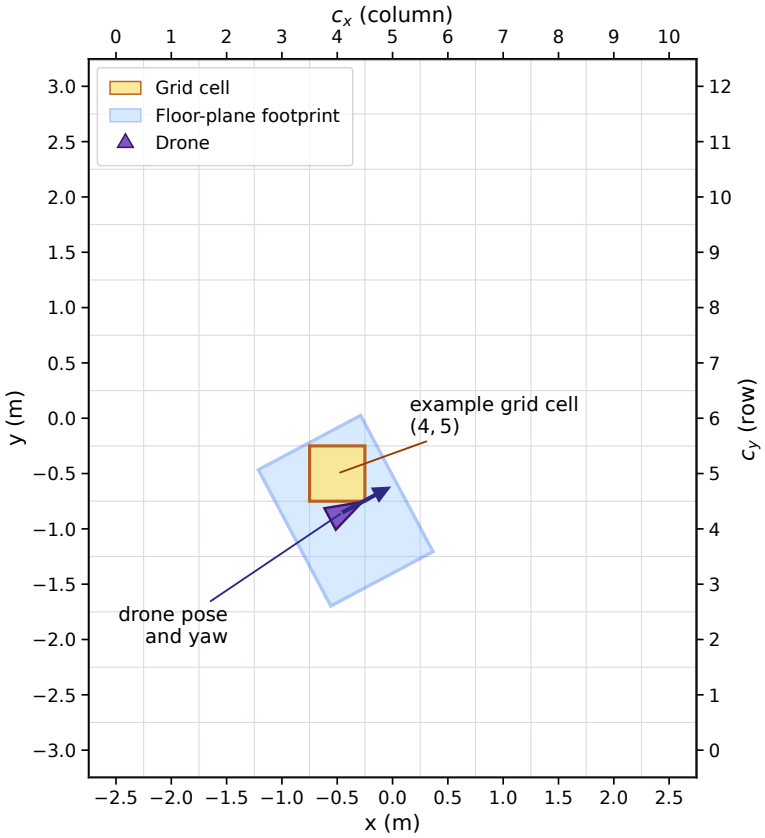

Figure 4.1: Continuous room geometry and task-grid abstraction. The simulated room is represented in metric world coordinates and partitioned into an 11×13 task grid. The drone pose and yaw remain continuous, while the projected downward-camera footprint determines the grid cells that can receive visual evidence.

in the horizontal plane, while the controller associates camera observations with grid cells through the geometric model introduced in the following section.

The visual-search task contains two target classes:

$$\mathcal{C}_{\text{target}} = \{\texttt{target\_bottle}, \texttt{target\_can}\}\,. \tag{4.6}$$

Only detections of these classes can produce task claims and contribute to successful completion of an episode. The table-enriched environments additionally contain desks, represented by the non-target class `desk`. Desk observations provide contextual information about raised surfaces and are used to determine where table-height observation geometry should apply, but they are not themselves objects to be found.

Table 4.1: Scene-object roles across experimental environments.

Objects	Role	Setting
Bottle and can	Claimable task targets	All
Desk	Raised-surface context	Tables, hard
Ball, cereal box, plate, and flower pot	Non-target visual clutter	Hard

The `mixed_hard` environment further includes non-target distractor objects. These are deliberately outside the task label set: they neither create target claims nor provide desk-context evidence. Their purpose is to introduce visually competing scene content while preserving the same bottle, can, and desk task semantics used in the table environment.

4.2 Observation Geometry and Surface-Aware Coverage

The drone does not observe grid cells directly. It observes camera images from a continuous pose, and the controller must infer which parts of the task grid are visible from that viewpoint. This makes the camera-footprint model a key part of the search system: it determines where detector evidence can be applied, how strongly a cell should be updated, and which future viewpoints are expected to be informative. In the table environments, the same camera image can also correspond to different physical surfaces, because raised desks are closer to the camera than the floor. The coverage model has to account for both partial cell visibility and surface height.

4.2.1 Floor-Plane Field of View and Cell Coverage

At the fixed flight altitude defined in Section 4.1.2, the downward-facing camera is represented by a rectangular footprint projected onto the floor plane. Its measured dimensions are 1.394 m in width, with forward and backward extents of 0.553 m and 0.493 m, respectively. The forward–backward asymmetry follows from the camera offset relative to the drone reference point.

For a drone position $\mathbf{p}$ and yaw angle ψ, let $\mathcal{F}_{\text{floor}}(\mathbf{p}, \psi)$ denote the rotated floor-plane footprint. For each grid cell $\mathcal{C}_i$, the controller computes its continuous coverage ratio as

$$w_i = \frac{\text{Area}\left(\mathcal{C}_i \cap \mathcal{F}_{\text{floor}}(\mathbf{p}, \psi)\right)}{\text{Area}\left(\mathcal{C}_i\right)}, \qquad w_i \in [0,1]. \tag{4.7}$$

This continuous quantity distinguishes between fully observed cells, partially observed cells, and cells touched only marginally by the footprint. A cell enters the binary visible-cell mask only when its coverage exceeds the threshold

$$\theta_{\text{cov}} = 0.05. \tag{4.8}$$

Accordingly,

$$m_i = \begin{cases} 1, & w_i \geq \theta_{\text{cov}}, \\ 0, & w_i < \theta_{\text{cov}}, \end{cases} \tag{4.9}$$

where m_i indicates whether cell i is treated as visible for the current observation. This prevents a negligible footprint overlap from being interpreted as meaningful visual evidence for an entire grid cell.

The continuous coverage values are retained after thresholding. Later belief updates use w_i to modulate the influence of visual evidence, whereas the binary mask determines which cells are eligible to receive an update. Figure 4.2 illustrates the distinction between full coverage, partial coverage, and marginal overlap below the threshold.

Figure 4.3 shows the views from the simulated downward-facing camera. The camera image supplies the visual input processed by the detector, while the footprint model determines which grid cells are considered observed by that image.

4.2.2 Table-Height Projection and Persistent Surface Cells

Raised desk surfaces are observed from a smaller camera footprint than the floor because they are closer to the downward-facing camera. The controller therefore maintains separate footprint models for the floor plane and the table plane. Let z_{flight} denote the fixed flight altitude, h_{floor} the floor height, and h_{table} the desk-surface height. The table-plane footprint is obtained by scaling the floor-plane footprint according to the camera-to-surface distance,

$$\alpha(h) = \frac{z_{\text{flight}} - h}{z_{\text{flight}} - h_{\text{floor}}}. \tag{4.10}$$

For the desk surfaces used in the experiments,

$$h_{\text{floor}} = 0\,\text{m}, \qquad h_{\text{table}} = 0.72\,\text{m}, \tag{4.11}$$

with $z_{\text{flight}} = 1.5\,\text{m}$. The resulting scale factor is

$$\alpha(h_{\text{table}}) = \frac{1.5 - 0.72}{1.5} = 0.52. \tag{4.12}$$

The width and forward and backward extents of the floor-plane footprint are multiplied by this factor to obtain the table-height footprint.

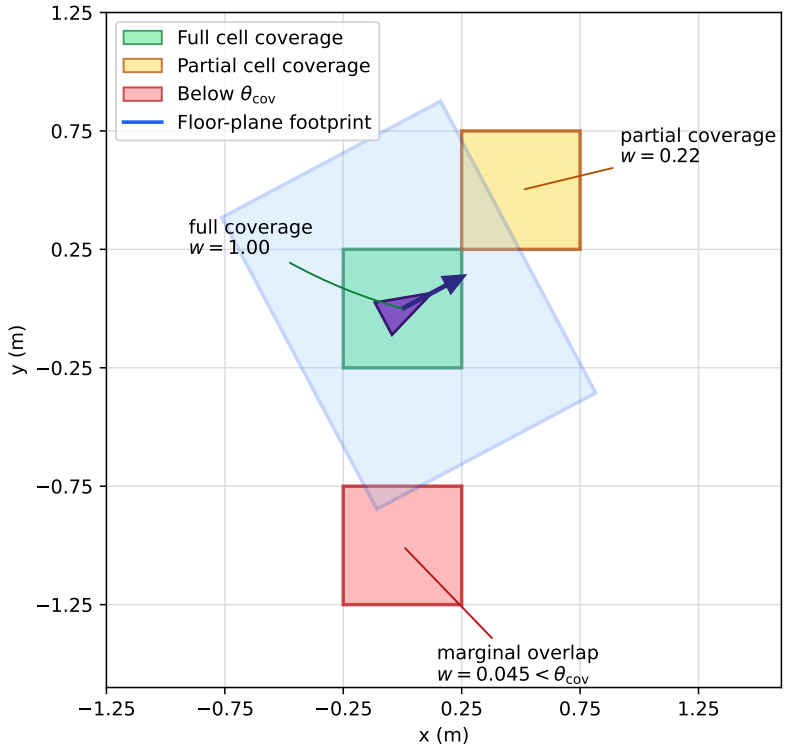

Figure 4.2: Examples of full, partial, and marginal grid-cell coverage. Coverage is defined as the fraction of a grid-cell area contained within the projected floor-plane footprint. The red cell is intersected only marginally, with $w_i < \theta_{\text{cov}}$, and is therefore excluded from the visible-cell mask.

Desk surfaces are discovered through detector observations. When the detector produces a desk bounding box, the controller samples points inside the box and projects them onto the table-height plane. The resulting grid cells are added to a persistent set of table-surface cells,

$$\mathcal{S}_{\text{table}} = \{i \mid \text{cell } i \text{ has been mapped from a detected desk}\}. \tag{4.13}$$

Each persistent cell also retains the highest desk-detection confidence associated with its discovery. This surface map is built from perception-derived desk detections and is used by the controller to select the appropriate observation geometry.

For a given pose, the controller computes floor-plane and table-plane coverage maps. The coverage value supplied for each cell is then selected according to the persistent surface map,

$$w_i = \begin{cases} w_i^{\text{table}}, & i \in \mathcal{S}_{\text{table}}, \\ w_i^{\text{floor}}, & i \notin \mathcal{S}_{\text{table}}. \end{cases} \tag{4.14}$$

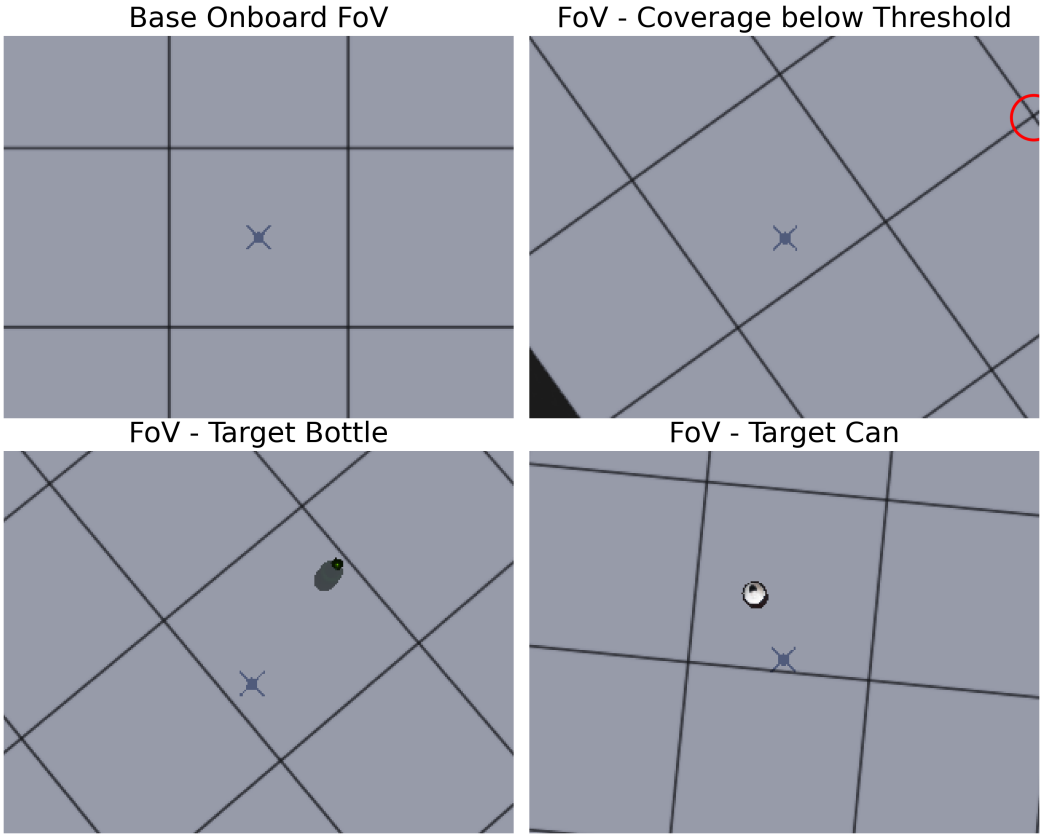

Figure 4.3: Examples of downward-camera field-of-view coverage. The panels show the floor grid as seen by the onboard camera in the `mixed_easy` environment. The upper-right panel highlights a cell whose camera-footprint overlap is below the coverage threshold $\theta_{\text{cov}} = 0.05$; the cell is therefore excluded from the visible-cell mask.

The persistent table-surface map is a geometric controller representation. The zone-of-interest policy additionally maintains a probabilistic table belief for each cell, introduced in Section 4.5.4. That belief conditions target priors during planning, while $\mathcal{S}_{\text{table}}$ determines which camera-footprint model is applied to a cell.

4.2.3 Current-Frame Table-Shadow Correction

The persistent table-surface map identifies cells that correspond to desk surfaces. A detected desk also affects the interpretation of coverage in the current camera frame. For an image location inside a desk bounding box, projection onto the floor plane lies farther from the drone than projection onto the raised desk plane. These paired floor-plane cells can lie behind the detected desk in the viewing direction.

Applying the larger floor-plane footprint to them would overestimate how much of that region is observed in the current frame.

For each processed desk detection, a fixed set of points is sampled inside the bounding box and projected onto both the desk-height plane and the floor plane. The desk-height projections update the persistent table-surface map described in Section 4.2.2. Their paired floor-plane projections form transient correction cells when they fall within the current forward viewing region. These cells use table-height geometry only for the current frame.

Let $\mathcal{T}^{(t)}$ denote the transient correction cells obtained from desk detections in perception frame t. The effective set of cells that use table-height geometry in that frame is

$$\mathcal{S}_{\text{eff}}^{(t)} = \mathcal{S}_{\text{table}} \cup \mathcal{T}^{(t)}, \tag{4.15}$$

where $\mathcal{S}_{\text{table}}$ is the persistent table-surface map. The surface-selected coverage for the current observation becomes

$$w_i^{(t)} = \begin{cases} w_i^{\text{table}}, & i \in \mathcal{S}_{\text{eff}}^{(t)}, \\ w_i^{\text{floor}}, & i \notin \mathcal{S}_{\text{eff}}^{(t)}. \end{cases} \tag{4.16}$$

The set $\mathcal{T}^{(t)}$ is discarded once perception frame t has been processed. It changes only the geometric coverage assigned in that frame, while persistent desk-surface information and policy-facing table evidence continue to originate from the desk-height projections.

Figure 4.4 illustrates the mechanism. Blue cells represent persistent desk-surface cells, while orange cells are their paired floor-plane correction cells for the current observation.

4.3 Perception Data and Detector Interface

The geometric coverage model determines where an observation can have an effect, while the detector determines what visual evidence is available in that observation. In this system, detector outputs are the bridge between rendered camera frames and the task-level search representation. Target detections can become claims and update target beliefs, while desk detections provide the contextual evidence needed for table-surface mapping and table-conditioned priors. This section describes how the simulated detector datasets were generated and how all detector conditions are converted into a common controller-facing format.

4.3.1 Simulation Dataset Generation

The learned detectors were trained on images collected directly from the downward-facing Webots camera used in the corresponding search environments. Dedicated

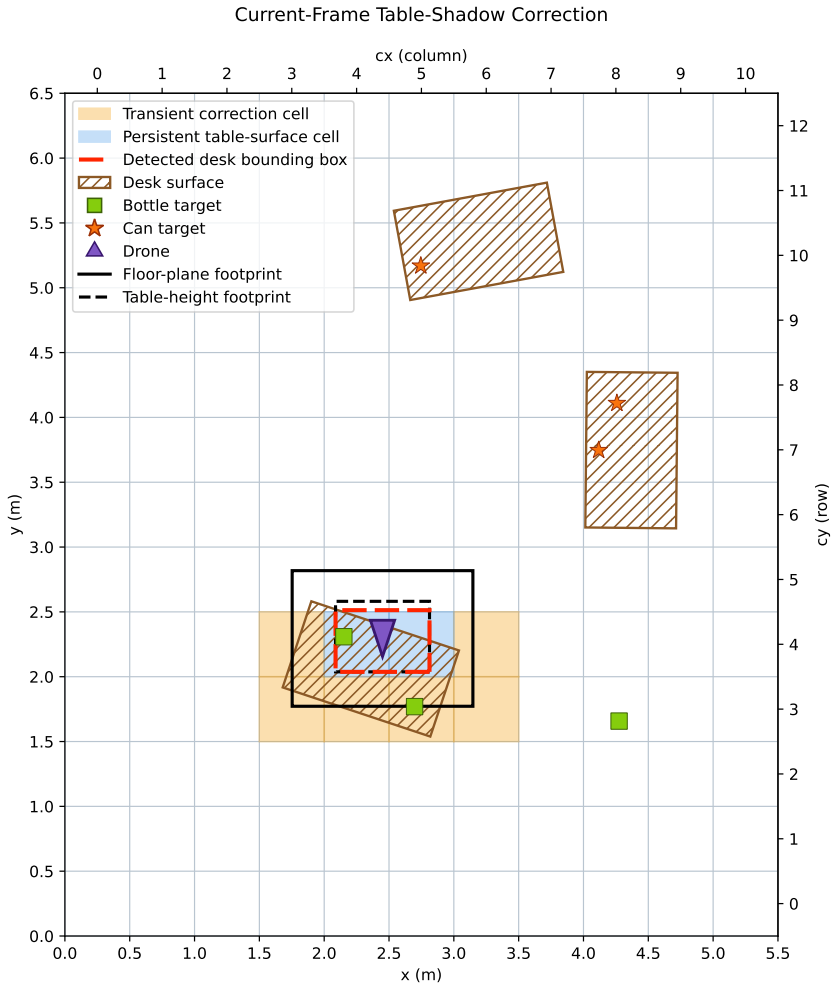

Figure 4.4: Current-frame table-shadow correction. A detected desk bounding box is sampled and projected onto the desk-height plane to form persistent table-surface cells. Paired floor-plane projections identify transient correction cells, which use the smaller table-height footprint only for the current observation. The solid and dashed outlines denote the floor-plane and table-height footprints, respectively.

dataset-generation controllers and supervisors generate scenes, capture rendered camera frames, and create annotations. These collection runs are separate from autonomous evaluation episodes, in which trained detector backends operate during flight.

Annotations are generated automatically through the Webots camera-recognition and segmentation utilities. The resulting labels describe the visible image-space

Table 4.2: Summary of detector-training datasets. Image counts are reported as train/validation/test splits. Background images contain no instances of the dataset label set, while bounding-box totals include all annotated object instances across the three splits.

Dataset	Labels	Images	Background	Boxes
`bottle_can_mixed`	bottle, can	1476 / 184 / 185	500	1675
`mixed_tables`	bottle, can, desk	1593 / 400 / 400	725	2570
`mixed_hard`	bottle, can, desk	1600 / 400 / 400	1342	1655

extent of each relevant simulated object, producing bounding boxes aligned with the rendered camera frame.

Three datasets were generated for the evaluated environments. The floor-only dataset contains bottle and can labels. The table dataset adds the `desk` class, allowing the detector to provide both target and raised-surface evidence. The hard-environment dataset retains these three labels while including distractor objects and the visually complex floor as unlabelled background. Images without annotated objects were retained as background samples.

The dataset splits were used for detector training and image-level validation and testing. Closed-loop evaluation was conducted separately through the experiment supervisors, which generated evaluation scenes and executed the complete detector–controller–policy loop. This allowed detector performance to be assessed both through held-out image predictions and through its effect on object-search performance.

4.3.2 Detector Conditions and Common Output Interface

All detector conditions are adapted to a shared controller-facing event format. For each processed camera frame t, the detector returns a set of events

$$\mathcal{D}^{(t)} = \left\{ d_k^{(t)} \right\}_{k=1}^{K_t}, \qquad d_k^{(t)} = (\ell_k, c_k, \mathbf{b}_k), \tag{4.17}$$

where ℓ_k is the predicted class label, $c_k \in [0,1]$ is the detector confidence, and $\mathbf{b}_k = (x_{1,k}, y_{1,k}, x_{2,k}, y_{2,k})$ is the image-space bounding box. The controller uses target-labelled events for target-cell association and claim generation. Desk-labelled events provide the bounding-box information required by the surface-mapping and current-frame correction mechanisms defined in Section 4.2.

The learned detector conditions include the reconstructed MobileNetV2-SSD model and variants of the YOLO11 family. The `mixed_easy` environment compares MobileNetV2-SSD, YOLO11-nano, and YOLO11-small against a simulator-derived

Table 4.3: Learned detector conditions and output classes by environment.

Environment	Learned detector conditions	Output classes
`mixed_easy`	MobileNetV2-SSD; YOLO11-nano; YOLO11-small	bottle, can
`mixed_tables`	YOLO11-small	bottle, can, desk
`mixed_hard`	YOLO11-small	bottle, can, desk

GT condition. The table and hard environments use YOLO11-small models trained for their respective datasets. The hard-environment model was trained separately from the table-environment model, while retaining the same bottle, can, and desk label set.

The GT condition uses Webots camera-recognition segmentation to determine object visibility. It returns simulator-derived target and desk events through the same controller interface, with perfect class labels and confidence. It provides a best-case perception reference for assessing how detector errors affect the closed-loop search task.

All three environments also include the GT condition described above. Table 4.3 summarizes the learned detector conditions and their output classes.

The shared event format keeps the perception interface fixed across detector conditions. Detector outputs can therefore be compared within the same controller and evaluation pipeline, while the policy determines how the resulting target and desk evidence influences subsequent movement decisions.

4.4 Closed-Loop Execution and Evaluation Pipeline

The evaluation is organised as a closed-loop system rather than as separate perception, planning, and reporting stages. At each step, the drone pose determines the camera view, the detector produces events on processed frames, the controller maps those events into the task grid, and the policy selects the next motion command. A separate supervisor manages the experimental side of the process: it spawns scenes, validates claims, detects task completion, and records results. This separation keeps the navigation policy limited to controller-side observations while still allowing the experiment to be evaluated against simulator ground truth.

4.4.1 Controller, Policy, and Flight-Control Loop

At each Webots control step, the robot controller reads the drone state and camera stream. The current pose is used to update the geometric coverage representation described in Section 4.2. On processed perception frames, the selected detector backend returns target and, where applicable, desk events. These events are combined with the coverage map, visible-cell mask, and current pose to form the policy observation.

Each policy ultimately produces a planar motion command

$$\mathbf{u} = \begin{bmatrix} v_{\text{forward}} & v_{\text{sideways}} & \dot{\psi} \end{bmatrix}^{\top}, \tag{4.18}$$

where v_{forward} and v_{sideways} are body-frame velocity commands, and $\dot{\psi}$ is the commanded yaw rate. The saccade policies first select a target cell and viewing direction; the navigator converts this selection into continuous body-frame commands. Detector-agnostic baselines directly generate commands within the same interface.

The fixed-height velocity PID controller receives the desired planar velocities, yaw rate, and nominal flight altitude, together with the current simulated state. It produces four motor-speed commands that are applied to the Webots drone model. The resulting motion determines the next pose and camera viewpoint, closing the perception–action loop.

4.4.2 Detector Claims, Supervisor Validation, and Reporting

A target detection becomes a claim only after it has been mapped to a task-grid cell. Each claim records the predicted target class and the associated cell coordinates. Desk detections are retained for surface mapping but cannot generate claims. The confidence and temporal criteria used to create target claims are defined in Section 4.6.1.

Claims are written by the robot controller and passed to the supervisor through the shared experiment state. A claim is not treated as confirmation by the controller itself. The supervisor maintains the exact target-object arrangement generated for the current run and independently evaluates each new claim against that arrangement. A claim is validated only when its cell contains a spawned target object and its predicted class matches the corresponding ground-truth class. Claims directed to empty cells or associated with the wrong class are recorded as incorrect, while repeated claims for the same cell and class are ignored.

Validated claims are accumulated throughout the episode. The supervisor updates the number of remaining targets and terminates the run when all required targets have been correctly claimed or when the episode timeout is reached. This

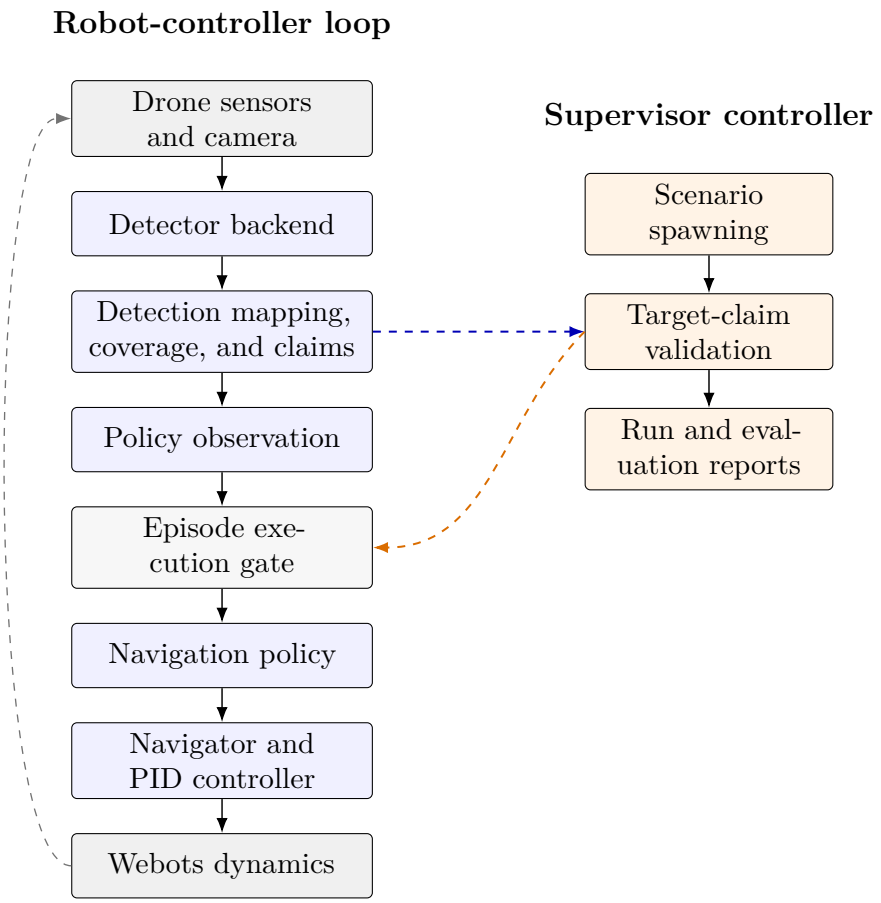

Figure 4.5: Closed-loop controller and evaluation pipeline. The robot controller combines simulated sensor data, detector outputs, and geometric coverage to form policy observations and generate flight commands. The supervisor spawns scenarios, validates controller-generated target claims against the spawned scene, and terminates the episode upon completion or timeout. The blue dashed link transfers claims from the robot controller to the supervisor. The orange dashed link returns episode-control information to the execution gate. The gray dashed loop represents progression to the next control step. No supervisor state is provided to the navigation policy.

termination state is returned to the controller execution gate shown in Figure 4.5, but it is not supplied to the navigation policy.

At the end of each run, the supervisor produces a run-level report containing the validated search outcome and claim-validation results. After all repetitions of an evaluation condition have completed, it also produces an aggregate report used to compute the metrics defined in Section 4.6.5.

Table 4.4: Baseline navigation policies.

Policy	Motion rule
Pseudo-random	Flies forward until a front range measurement indicates an obstacle, then turns by a randomly selected yaw angle.
Wall-following	Tracks the room boundary at a reference distance using range measurements and adjusts its heading around corners.
Spiral	Builds on wall-following by progressively changing the reference boundary distance after completed laps, producing inward or outward coverage paths.
Rotate-and-measure	Rotates through a full scan, samples obstacle distance at fixed yaw intervals, and travels towards the most open measured direction before scanning again.

4.5 Navigation Policies

The navigation policies differ in how they use the information produced by the controller. The baseline policies treat search primarily as geometric exploration: they move according to fixed or range-based rules, while detections can still generate claims through the shared controller pipeline. The active-inference-inspired policies instead use detector evidence, predicted coverage, and internal beliefs to select future sensing actions. This section defines both groups of policies and then describes how the simple binary AIF saccade policy and the zone-of-interest extension score candidate viewpoints.

4.5.1 Baseline Navigation Policies

The baseline policies follow the exploration-policy concepts used by Lamberti et al. [1], reimplemented and adapted for the present controller and evaluation pipeline. Their movement decisions are independent of target detections, desk detections, detector confidence, and policy belief states. Detector events can still produce target claims through the shared controller logic, but they do not influence the trajectory selected by a baseline policy. Table 4.4 summarizes the motion rule implemented for each baseline.

The baselines provide reference behaviours for spatial coverage without perception-driven action selection. Their results indicate how much task performance can be achieved through geometric exploration alone, before introducing policies that explicitly use detector evidence to select where and how to observe next.

4.5.2 Shared AIF-Inspired Planning Structure

The two AIF-inspired policies use the same high-level planning structure. At each planning step, the policy evaluates candidate sensing actions and selects the one with the lowest expected cost. A candidate action is defined as

$$a = (j, \psi), \tag{4.19}$$

where j is a target grid cell and ψ is the yaw angle from which that cell should be observed. After an action is selected, the navigator converts (j, ψ) into continuous body-frame velocity commands using the interface described in Section 4.4.1.

The policies do not optimize a sequence of future actions. They score the immediate next sensing action, execute the corresponding movement command, and update the action choice as new motion and detector evidence become available. This matches the closed-loop structure of the simulated task, where the value of a viewpoint changes as coverage, detections, and validated claims accumulate.

For each candidate action, the controller predicts which cells would be visible from the corresponding viewpoint using the surface-aware coverage model from Section 4.2. The candidate is then scored using the common form

$$G(a) = R(a) - \beta_{\text{info}} I(a) + \lambda_d d(a). \tag{4.20}$$

The selected action is

$$a^\star = \arg\min_{a \in \mathcal{A}} G(a), \tag{4.21}$$

where $\mathcal{A}$ is the set of candidate cell–yaw pairs.

The term $R(a)$ is the policy-specific task cost. In the simple binary policy, it is derived from the mismatch between predicted beliefs and preferred target states. In the zone-of-interest policy, it is replaced by a utility-based term that accounts for target belief, table context, and classification state. The term $I(a)$ represents the expected information gained by observing the cells visible under action a, weighted by the predicted coverage of those cells. The final term $d(a)$ penalizes the travel distance from the current drone position to the candidate cell, with weight λ_d. The distance term does not restrict the candidate set: any grid cell can still be selected. It biases the score toward closer viewpoints when the expected task value and information gain are otherwise similar, reducing unnecessary travel during search.

This shared structure is active-inference-inspired because movement is selected by combining task preference, expected uncertainty reduction, and action cost.

4.5.3 Simple Binary AIF Saccade Policy

The simple AIF saccade policy represents each grid cell with a binary hidden state,

$$s_i \in \{0,1\}, \qquad 0 \equiv \text{target absent}, \qquad 1 \equiv \text{target present}. \tag{4.22}$$

Table 4.5: Binary observation likelihood for the simple AIF policy.

	Hidden state	
Observation	$s_i = 0$	$s_i = 1$
$o_i = 0$	$1 - p_{\text{fa}}$	$1 - p_{\text{hit},i}$
$o_i = 1$	p_{fa}	$p_{\text{hit},i}$

This policy models target presence but not target class. Bottle and can detections both provide evidence that a target object is present in the associated cell. The predicted class is still used by the controller-supervisor claim mechanism, but it is not part of this policy's belief state.

For each visible cell, the detector observation is reduced to a binary variable,

$$o_i \in \{0,1\}, \qquad 0 \equiv \text{no target detection}, \qquad 1 \equiv \text{target detection}. \tag{4.23}$$

The likelihood model is defined by a false-alarm probability p_{fa} and a hit probability $p_{\text{hit},i}$:

When a target detection is mapped to cell i, the detector confidence c_i is used as positive evidence strength. When no detection is produced in a visible cell, the policy uses a fixed no-detection sensitivity parameter η_{nd}. Thus,

$$p_{\text{hit},i} = \begin{cases} \max(c_i, \eta_{\text{nd}}), & o_i = 1, \\ \eta_{\text{nd}}, & o_i = 0. \end{cases} \tag{4.24}$$

A larger η_{nd} makes missed detections more informative, because failing to detect a target in a visible cell becomes stronger evidence against target presence.

For every visible cell, the policy first computes the full Bayesian posterior

$$\tilde{q}_i(s_i) = \frac{P(o_i \mid s_i) q_i(s_i)}{\sum_{s_i'} P(o_i \mid s_i') q_i(s_i')}. \tag{4.25}$$

The update applied to the belief map is then weighted by the geometric coverage w_i of the cell:

$$q_i^{\text{new}}(s_i) = (1 - w_i) q_i(s_i) + w_i \tilde{q}_i(s_i). \tag{4.26}$$

Cells outside the visible-cell mask are not updated. Partial coverage therefore produces a partial belief update, while full coverage applies the complete Bayesian update from Equation 4.25.

Action selection follows the shared scoring structure from Section 4.5.2. For each candidate cell–yaw pair, the policy predicts the cells that would be visible from that viewpoint. The expected information term is accumulated over those

cells, weighted by their predicted coverage. The task-cost term is evaluated for the candidate target cell using the binary preference for target-present observations, and the distance cost biases the score against unnecessary travel.

A limitation of this binary formulation is that successful fixation does not change the policy state. Once a cell has produced repeated target evidence, the policy can continue to prefer that same location because it has no class-specific state representing that the object has already been classified or that the saccade has completed its useful role. The simple policy therefore uses an internal defixation rule: after repeated target detections in processed perception frames, the corresponding cell is removed from the candidate set for future action selection. This is a policy-level workaround and does not depend on supervisor validation. It encourages the controller to return to exploration after a target has generated claims, instead of repeatedly fixating the same target location.

4.5.4 Zone-of-Interest AIF Saccade Policy

The zone-of-interest policy extends the simple binary model by representing both target presence and target class. It is used in environments where target objects may appear on the floor or on raised desk surfaces. The policy keeps a categorical target belief for each grid cell and a separate probabilistic belief over table-surface cells. This allows desk detections to influence where targets are expected, while target detections continue to update the belief over object presence and class.

Each grid cell has a categorical hidden state

$$s_i \in \mathcal{S}, \qquad \mathcal{S} = \{E, U, B, C\}, \tag{4.27}$$

where E denotes an empty cell, U denotes an unclassified target, B denotes a bottle, and C denotes a can. The corresponding belief is

$$q_i = \begin{bmatrix} q_i(E) & q_i(U) & q_i(B) & q_i(C) \end{bmatrix}^\top. \tag{4.28}$$

The policy also maintains a Bernoulli table belief

$$q_{T,i} = P(T_i = 1), \tag{4.29}$$

where $T_i = 1$ denotes that cell i is believed to belong to a desk surface. This probabilistic table belief is distinct from the persistent table-surface map used by the controller for height-aware coverage. The persistent map selects the geometric footprint model, while $q_{T,i}$ conditions the policy's target prior.

The target-present prior for each cell is a mixture of a floor prior and a table prior:

$$p_i^{\text{tar}} = (1 - q_{T,i})p_{\text{floor}} + q_{T,i}p_{\text{table}}. \tag{4.30}$$

The implemented values are

$$p_{\text{floor}} = \frac{1}{125}, \qquad p_{\text{table}} = \frac{5}{18}, \qquad q_{T,0} = \frac{18}{143}, \qquad \rho_U = 0.70. \tag{4.31}$$

Here, $q_{T,0}$ is the initial table-belief prior and ρ_U is the fraction of target-present prior mass assigned to the unclassified-target state. These values encode the intended structure of the table environments: only part of the grid is expected to correspond to desk surfaces, and target presence is assigned a higher prior probability on those surfaces.

Given p_i^{tar}, the context-conditioned prior over target states is

$$P_i(s_i) = \begin{bmatrix} 1 - p_i^{\text{tar}} & \rho_U p_i^{\text{tar}} & \frac{1-\rho_U}{2} p_i^{\text{tar}} & \frac{1-\rho_U}{2} p_i^{\text{tar}} \end{bmatrix}^\top. \tag{4.32}$$

Most of the target-present mass is initially assigned to U, so a cell can become attractive as a likely target location before its bottle-or-can class has been resolved.

When table belief changes, the target belief is adjusted by replacing the old context prior with the new one:

$$q_i^{\text{new}}(s_i) \propto q_i(s_i) \frac{P_i^{\text{new}}(s_i)}{P_i^{\text{old}}(s_i)}. \tag{4.33}$$

The belief is normalized after this update. This prior-ratio update preserves accumulated target evidence while allowing newly observed desk context to change the prior probability assigned to target states.

Table belief is updated from controller-provided desk observations. Positive table evidence is produced when a desk bounding box is projected into table-height cells. Negative table evidence is available for visible table-projection cells that did not receive positive desk evidence in the current processed perception frame. For a table observation, the binary likelihood is defined by a table false-alarm probability $p_{\text{fa},T}$ and a table hit probability $h_{T,i}$:

$$P(o_{T,i} = \text{detected} \mid T_i = 0) = p_{\text{fa},T}, \qquad P(o_{T,i} = \text{detected} \mid T_i = 1) = h_{T,i}. \tag{4.34}$$

Positive desk observations use detector confidence directly,

$$h_{T,i} = c_{\text{desk}}, \tag{4.35}$$

whereas no-table observations use coverage-dependent negative evidence,

$$h_{T,i} = \max(p_{\text{fa},T},\, w_i \eta_T). \tag{4.36}$$

The implemented table no-detection sensitivity is $\eta_T = 0.50$. Thus, absence of a desk detection in a weakly covered cell provides weaker evidence against table presence than absence in a well-covered cell.

Target observations are represented by four labels,

$$\mathcal{O}_{\text{tar}} = \{\varnothing, G, O_B, O_C\}, \tag{4.37}$$

where $\varnothing$ denotes no target detection, G denotes a generic target detection, O_B denotes a bottle observation, and O_C denotes a can observation.

The policy uses two target likelihood models, shown in Tables 4.6 and 4.7. The likelihood applied to a visible cell depends on whether the cell is treated as a fixation-level evidence cell. Let

$$\theta_{\text{cls}} = 0.55 \tag{4.38}$$

be the classification-coverage threshold. A cell i is updated with the fixation likelihood if it is the current fixation cell, or if its coverage satisfies

$$w_i \geq \theta_{\text{cls}}. \tag{4.39}$$

Cells that do not satisfy this condition are updated with the peripheral likelihood. In those cells, bottle and can detections are collapsed into the generic target observation G. The implemented peripheral class weight is 0.0, so class-specific evidence is not applied outside fixation-level observations. Peripheral detections therefore attract the policy toward possible targets, while class resolution is left to views in which the target cell is either being fixated or sufficiently covered.

Let $p_f = p_{\text{fa}}$ and $p_h = p_{\text{hit},i}$. The peripheral likelihood is given in Table 4.6.

Table 4.6: Peripheral likelihood for the zone-of-interest policy.

	Hidden state			
Observation	E	U	B	C
$\varnothing$	$1 - p_f$	$1 - p_h$	$1 - p_h$	$1 - p_h$
G	p_f	p_h	p_h	p_h

At fixation, the likelihood allows the observation to resolve the target class. Let γ_U denote the probability that an unclassified target produces a class-specific report, r the probability that a class-known target is reported with a class label, and α the class-report accuracy. The implemented configuration uses $\gamma_U = 0.25$, $r = 1.00$, and $\alpha = 0.92$. The fixation likelihood is given in Table 4.7.

Table 4.7: Fixation likelihood for the zone-of-interest policy.

	Hidden state			
Observation	E	U	B	C
$\varnothing$	$1-p_f$	$1-p_h$	$1-p_h$	$1-p_h$
G	$0.90p_f$	$(1-\gamma_U)p_h$	$(1-r)p_h$	$(1-r)p_h$
O_B	$0.05p_f$	$0.5\gamma_U p_h$	$\alpha r p_h$	$(1-\alpha)rp_h$
O_C	$0.05p_f$	$0.5\gamma_U p_h$	$(1-\alpha)rp_h$	$\alpha r p_h$

The hit probability depends on the observation type. Positive target detections use detector confidence directly, while no-detection observations use coverage-dependent negative evidence:

$$p_{\text{hit},i} = \begin{cases} c_i, & o_i \neq \varnothing, \\ \max(p_{\text{fa}},\, w_i \eta_{\text{nd}}), & o_i = \varnothing. \end{cases} \tag{4.40}$$

Here, c_i is the confidence of the mapped detection, w_i is the current coverage of the cell, and η_{nd} is the target no-detection sensitivity. The reported configuration uses $p_{\text{fa}} = 0.04$ and $\eta_{\text{nd}} = 0.50$. This differs from the simple binary policy: in the zone-of-interest policy, coverage is built into the no-detection likelihood itself, while positive detections retain the detector confidence as their evidence strength.

For a processed target observation, the categorical posterior update is

$$q_i^{\text{new}}(s_i) = \frac{P(o_i \mid s_i) q_i(s_i)}{\sum_{s_i'} P(o_i \mid s_i') q_i(s_i')}. \tag{4.41}$$

Cells outside the visible-cell mask are not updated. Target detections update q_i, while desk detections update the table belief $q_{T,i}$ through the table-observation model above.

The policy prioritizes cells that are likely to contain a target and still need classification. For cell i, the target-present probability is

$$p_i^{\text{present}} = q_i(U) + q_i(B) + q_i(C). \tag{4.42}$$

The class confidence is defined as

$$\chi_i = \frac{\max(q_i(B), q_i(C))}{\max(p_i^{\text{present}}, \epsilon)}, \tag{4.43}$$

where ϵ is a small numerical constant. This value measures how concentrated the target-present belief is on the most likely class, conditional on the cell containing a target.

The classification-need term is

$$\kappa_i = p_i^{\text{present}}(1 - \chi_i). \tag{4.44}$$

It is high when a cell is likely to contain a target but the posterior has not yet resolved whether the target is a bottle or a can. It becomes small when the cell is likely empty, and it also becomes small after repeated class-consistent observations concentrate the belief on one class.

For action selection, the zone-of-interest policy uses the shared score from Equation 4.20, with the task-cost term defined as negative utility:

$$R(a) = -U_{\text{zoi}}(a). \tag{4.45}$$

For a candidate action $a = (j, \psi)$, the classification utility is based on the selected cell j. Let

$$p_j^{\text{class}} = P(O_B \mid q_j, A_{\text{fix}}) + P(O_C \mid q_j, A_{\text{fix}}), \tag{4.46}$$

where A_{fix} is the fixation likelihood. The utility contribution of the candidate fixation cell is

$$U_{\text{zoi}}(a) = \lambda_c \kappa_j p_j^{\text{class}}, \tag{4.47}$$

where λ_c is the classification-preference strength. The reported configuration uses $\lambda_c = 2.0$. A candidate viewpoint is therefore valuable when it is expected to view a likely target whose class remains uncertain and for which fixation is likely to produce a class-specific observation.

The expected information-gain term is accumulated over the cells predicted to be visible from the candidate viewpoint. For each visible cell, the fixation likelihood is used when the cell satisfies the fixation-level evidence condition from Equation 4.39; otherwise, the peripheral likelihood is used. This makes candidate viewpoints valuable when they reduce uncertainty over the surrounding scene and, when coverage or fixation is sufficient, offer the possibility of resolving the class of a likely target.

Unlike the simple binary policy, the zone-of-interest policy does not remove a detected cell from the candidate set after a fixed number of detections. Once a bottle or can has been classified with high confidence, χ_i increases and κ_i decreases, so the cell becomes less attractive in the action score. The policy can then return to exploration or move toward another uncertain target-like region. This produces the intended fixate, classify, and return-to-search behaviour through the belief state itself.

The main differences between the two AIF-inspired policies are summarized in Table 4.8.

Table 4.8: Comparison of simple and zone-of-interest AIF policies.

Component	Simple binary policy	Zone-of-interest policy
Target state	Absent or present	Empty, unclassified target, bottle, or can
Peripheral evidence	Binary target evidence	Generic target evidence that attracts fixation
Fixation evidence	Same binary target update	Class-specific bottle and can update
Table context	Not represented	Table belief conditions target priors; detected table cells select surface-aware coverage
After detection	Cell exclusion after repeated detections	Reduced priority after classification through posterior belief and information gain

4.6 Experimental Protocol and Metrics

The evaluation protocol is designed to compare policies under matched closed-loop conditions. Each policy is tested on the same sequence of randomised scenarios within an environment, with the same timeout, warm-up period, detector-processing cadence, and claim-validation rules. The metrics then describe complementary aspects of the resulting behaviour: whether the task is completed, how many targets are recovered, how quickly successful runs finish, how much of the grid is physically visited, and how reliable the submitted claims are. This section defines the shared protocol, the three evaluation environments, and the task-level metrics reported in Chapter 5.

4.6.1 Shared Evaluation Protocol

Each evaluation condition is repeated for 100 runs. The run seeds are the consecutive integers from 1 to 100, allowing the same sequence of randomised scenarios to be reused across policies and detector conditions within an environment. This makes comparisons depend on policy and detector behaviour rather than on different sampled scene sequences.

At the beginning of each episode, the supervisor spawns the environment and the controller remains in a warm-up phase for

$$T_{\text{warmup}} = 15\,\text{s}. \tag{4.48}$$

Table 4.9: Shared evaluation settings.

Setting	Value
Runs per condition	100
Seeds	$1, \ldots, 100$
Warm-up duration	15 s
Autonomous timeout	180 s after warm-up
Detector processing interval	0.625 s
Nominal detector evidence rate	1.6 Hz
Claim confidence threshold	0.45
Claim streak requirement	1 processed frame

The warm-up period allows the drone to take off and settle into a stable flight condition before the autonomous search episode is evaluated. After warm-up, the autonomous episode is allowed to run for at most

$$T_{\text{timeout}} = 180\,\text{s}. \tag{4.49}$$

A run terminates earlier if the supervisor validates all required target objects.

All detector conditions are constrained to the same processed-frame interval,

$$\Delta t_{\text{det}} = 0.625\,\text{s}, \qquad f_{\text{det}} = 1.6\,\text{Hz}. \tag{4.50}$$

This cadence is applied to learned detectors and to the GT detector condition. The interval is chosen with reference to the onboard detector rate reported by Lamberti et al. [1], and is used here to equalize the rate at which policies receive new visual evidence.

Target claims are produced directly from processed target detections. The evaluation configuration uses a minimum confidence threshold of 0.45 and a one-frame streak requirement, so any target detection emitted by the active detector backend is immediately mapped to a grid cell and submitted as a claim. The supervisor then determines whether the claim is correct, incorrect, duplicated, or completes the episode.

The fixed settings used by every evaluation condition are summarized in Table 4.9.

4.6.2 Floor-Only Mixed Environment: `mixed_easy`

The `mixed_easy` environment is the floor-only search setting. Each run contains six target objects, with three bottles and three cans placed on the room floor. No desks or distractor objects are spawned. The environment therefore uses a single surface plane for coverage, target mapping, and belief updates.

This setting isolates the floor-level object-search problem. Since all targets lie on the same surface plane and no contextual objects are present, differences in performance can be attributed to search behaviour, detector quality, and claim validation rather than to table-surface reasoning or distractor clutter. It therefore provides the baseline environment for evaluating the simple AIF saccade policy before introducing the table-aware extension.

The detector conditions for this environment are the GT detector, MobileNetV2-SSD, YOLO11-nano, and YOLO11-small, as summarized in Table 4.3. The ground-truth condition provides the best-case perception reference for the same search problem, while the learned detector conditions show how downstream task performance changes when navigation depends on imperfect visual detections.

The role of `mixed_easy` is therefore twofold. It tests whether the simple AIF-inspired policy improves object search over geometric exploration in a floor-only setting, and it measures how strongly the same task depends on detector quality when the environment does not yet include raised surfaces or contextual table evidence.

4.6.3 Table and Zone-of-Interest Environment: `mixed_tables`

The `mixed_tables` environment introduces raised desk surfaces while preserving the same target-search objective as the floor-only setting. Each run contains three desks and six target objects, with three bottles and three cans. Target spawning is biased toward desk surfaces, with a table-spawn preference of 0.90, although floor placement remains possible when a table placement is not selected or cannot be sampled.

The purpose of this setting is to evaluate the same search-and-classification task in scenes where non-target structure is relevant to the search process. Desks are not claimable objects, but they affect the task because target placement is biased toward them and because detected desk surfaces change how the controller interprets camera coverage.

The evaluated detector conditions are the GT detector and a YOLO11-small detector trained on the `mixed_tables` dataset. Both conditions expose bottle, can, and desk events through the common detector interface defined in Section 4.3.2. Bottle and can events support target claims and target-belief updates, while desk events support table-surface mapping and table-conditioned target priors.

Three navigation policies are evaluated in this environment: the zone-of-interest AIF saccade policy, the simple binary AIF saccade policy, and the spiral policy. The spiral policy provides the detector-agnostic reference, while the simple binary policy separates the effect of perception-aware target search from the additional table-aware and class-specific mechanisms introduced by the zone-of-interest policy.

Figure 4.6: Example scene from the `mixed_hard` environment. The hard setting keeps the same bottle, can, and desk task semantics as the table environment, while adding distractor objects and a visually more complex floor texture.

4.6.4 Hard Mixed Environment: `mixed_hard`

The `mixed_hard` environment preserves the six-target search objective but increases visual clutter and scene complexity. Each run contains three bottles, three cans, and three desks. The environment also includes non-target distractors: one soccer ball, two cereal boxes, three plates, and two flower pots. These objects are not part of the task label set and cannot produce valid target claims.

The distractors are deliberately treated as background rather than as additional

Figure 4.7: Distractor objects used in the `mixed_hard` environment. Soccer balls, cereal boxes, plates, and flower pots are included as non-target visual clutter. They are not claimable objects and are not labelled as detector output classes.

detector classes. This keeps the closed-loop task unchanged: the drone must still find and classify bottles and cans, while desks remain contextual surface evidence. The added objects instead make the visual input harder by introducing scene content that should be ignored by the detector and, consequently, by the claim-validation mechanism.

The evaluated detector conditions are the GT detector and a YOLO11-small detector trained on the `mixed_hard` dataset. The learned detector retains the same output classes as the table setting: bottle, can, and desk. Distractor objects and floor texture are left unlabelled during detector training, so they act as negative background examples rather than as explicit semantic categories.

The `mixed_hard` environment is used to evaluate whether the table-aware search pipeline remains effective when the same target and desk semantics are embedded

in a more visually cluttered scene. It therefore tests the combined effect of detector robustness, distractor rejection, surface-aware coverage, and policy action selection under the most difficult simulated setting considered in this thesis.

4.6.5 Evaluation Metrics

The metrics are defined at the level of supervisor-validated task outcomes. They measure whether the closed-loop system completes the search task, how much of the target set it recovers on average, how efficiently successful runs finish, how much of the task grid the drone physically visits, and how reliable its submitted claims are.

Let $N = 100$ be the number of runs in an evaluation condition, and let $M = 6$ be the number of required targets in each run. For run r, let $Y_r \in \{0,1\}$ indicate whether all targets were found before timeout, F_r the number of validated targets, and V_r the number of unique grid cells entered by the drone.

The success rate is

$$S = \frac{1}{N} \sum_{r=1}^{N} Y_r. \tag{4.51}$$

A run is successful only if the supervisor validates all required target objects before the timeout.

Target recovery measures the fraction of required targets validated across all runs:

$$R_{\text{tar}} = \frac{1}{NM} \sum_{r=1}^{N} F_r. \tag{4.52}$$

This metric includes both successful and failed runs. Successful runs contribute all M targets, while failed runs contribute the number of targets validated before timeout.

Search time is reported only over successful runs. Let $\mathcal{R}_{\text{succ}} = \{r : Y_r = 1\}$, and let T_r be the autonomous search time of successful run r, excluding the fixed warm-up period. The mean time to success is

$$\overline{T}_{\text{succ}} = \frac{1}{|\mathcal{R}_{\text{succ}}|} \sum_{r \in \mathcal{R}_{\text{succ}}} T_r. \tag{4.53}$$

Failed runs are not included in this average.

Spatial exploration is measured through the visited-cell ratio,

$$R_{\text{visit}} = \frac{1}{N} \sum_{r=1}^{N} \frac{V_r}{143}. \tag{4.54}$$

This is distinct from camera-footprint coverage: it measures how much of the grid was physically visited by the drone.

Table 4.10: Evaluation metrics used for closed-loop search.

Metric	Meaning	Reported as
Success rate	Fraction of runs in which all six targets are validated before timeout	Percentage of runs
Target recovery	Fraction of required targets validated across all runs	Percentage of targets
Mean time to success	Autonomous search duration for successful runs only	Seconds
Visited-cell ratio	Fraction of the 143-cell grid physically entered by the drone	Percentage of cells
Claim precision	Fraction of evaluated non-duplicate target claims validated by the supervisor	Percentage of claims

Finally, claim precision measures the reliability of target claims submitted to the supervisor. Let C_v be the number of validated target claims and C_i the number of incorrect non-duplicate target claims. Then

$$P_{\text{claim}} = \frac{C_v}{C_v + C_i}. \tag{4.55}$$

Incorrect claims include claims directed to empty cells and claims with the wrong target class. Duplicate claims ignored by the supervisor are not counted.

Table 4.10 summarizes the metrics used in the results chapter.

Chapter 5

Results

This chapter presents the policy evaluation results for the three environments defined in Chapter 4. The `mixed_easy` setting tests the simple AIF saccade policy in a floor-only mixed-object search task. The `mixed_tables` setting introduces raised surfaces and a probabilistic bias that makes table cells more likely to contain targets. The `mixed_hard` setting keeps the same target and table structure, but adds non-target clutter and a visually more complex floor texture. The main comparison is between baseline exploration policies, whose movement decisions do not depend on detector evidence, and policies that use perception to decide where to move next. In the floor-only environment, the relevant question is whether the simple AIF policy improves over the baseline policies. In the table-based environments, the relevant question becomes whether the zone-of-interest AIF policy can use table beliefs and class-specific target states to improve search reliability and spatial selectivity.

As described in Section 4.3.1, the learned detectors were trained on simulated datasets generated for the corresponding evaluation settings. Their held-out performance was high across the tested environments, so the following analysis does not treat detector performance as a separate benchmark. Instead, learned-detector runs are used to evaluate whether the policy-level results remain stable when the observations are produced by trained visual models. In the closed-loop evaluation, the most relevant detector-related quantity is claim precision, since it measures how often the claims emitted during autonomous flight are validated by the supervisor.

5.1 Interpreting the Aggregate Metrics

The results should be read as combinations of metrics rather than as isolated rankings. Success rate and target recovery describe whether the task is being solved reliably. Mean time to success and visited-cell ratio are meaningful only in relation

to those completion metrics. A policy that succeeds rarely can still appear fast on the few favourable runs in which its trajectory happens to observe all targets, and a policy can visit few cells simply because it spends time rotating or repeatedly follows a limited path.

This point is especially important for the baseline policies. Wall-following, pseudo-random motion, and rotate-and-measure do not select viewpoints according to target belief. Their trajectories are determined by geometry, range readings, or random direction choices, not by the current evidence about target locations. As a result, seed-specific target placement can strongly affect whether a run succeeds. Wall-following may complete a run quickly when all targets happen to lie along the followed path. Rotate-and-measure can have a low visited-cell ratio because it spends much of the episode rotating in place before moving. These cases are not evidence of better search behaviour unless they are accompanied by high success and recovery.

For the AIF policies, the central question is whether the belief model changes the search process in a useful way: not only by increasing the probability of completing the task, but by directing the drone toward informative regions instead of relying on a fixed traversal pattern. In the floor-only setting, the simple AIF policy can use target evidence to revisit uncertain or promising regions. In the table settings, this is no longer sufficient. The simple AIF policy receives height-aware coverage, but it does not represent raised surfaces as regions with higher target probability. Since the projected camera footprint is smaller on table-height cells, candidate views containing known table cells can produce lower coverage-weighted information gain unless target evidence has already accumulated there. This is not an explicit penalty on tables, but it creates a bias against table regions in an environment where those regions are more likely to contain targets.

The zone-of-interest AIF policy is designed to address this mismatch. Its table belief and table-conditioned target prior make raised surfaces informative for planning, rather than treating them only as regions with reduced projected coverage.

5.2 Floor-Only Mixed Environment

Table 5.1 reports the results for the `mixed_easy` environment. The simple AIF saccade policy is the only policy that reaches 100% success across all detector conditions.

Spiral exploration is the closest baseline, with 97% success and approximately 99% target recovery. However, it requires substantially broader traversal: its visited-cell ratio is about 48%–49%, compared with about 31%–32% for AIF Saccade. Since both policies recover almost all targets, this difference indicates that the AIF

Table 5.1: Results in the floor-only mixed environment. Recovery, time, and visited-cell ratio are reported as mean ± standard deviation. Mean time to success is computed only over successful runs. YOLOn and YOLOs denote YOLO11-nano and YOLO11-small, respectively; MBv2-SSD denotes MobileNetV2-SSD.

Policy	Detector	Success (%)	Recovery (%)	Time (s)	Visited (%)	Precision (%)
AIF Saccade	GT	100.0	100.0 ± 0.0	89.3 ± 19.9	30.9 ± 6.7	100.0
AIF Saccade	YOLOs	100.0	100.0 ± 0.0	92.4 ± 19.3	31.7 ± 6.6	90.8
AIF Saccade	YOLOn	100.0	100.0 ± 0.0	92.5 ± 21.4	31.7 ± 7.3	91.1
AIF Saccade	MBv2-SSD	100.0	100.0 ± 0.0	92.0 ± 19.7	32.0 ± 6.4	69.5
Spiral	GT	97.0	99.2 ± 5.5	97.0 ± 19.9	48.1 ± 8.0	100.0
Spiral	YOLOs	97.0	99.0 ± 6.2	97.2 ± 20.0	48.2 ± 8.0	95.2
Spiral	YOLOn	97.0	99.0 ± 6.2	97.2 ± 20.0	48.3 ± 8.0	95.7
Spiral	MBv2-SSD	97.0	99.0 ± 6.2	99.3 ± 19.3	49.1 ± 7.6	75.3
Wall Following	GT	13.0	72.7 ± 16.5	81.1 ± 14.0	28.8 ± 2.9	100.0
Wall Following	YOLOs	11.0	72.2 ± 16.4	82.7 ± 14.2	28.9 ± 2.8	97.1
Wall Following	YOLOn	11.0	72.2 ± 16.4	89.7 ± 28.8	28.9 ± 2.8	97.3
Wall Following	MBv2-SSD	8.0	71.5 ± 15.8	85.3 ± 15.2	29.0 ± 2.6	63.5
Pseudo-Random	GT	36.0	82.0 ± 17.4	125.6 ± 35.3	39.4 ± 6.6	100.0
Pseudo-Random	YOLOs	38.0	85.2 ± 15.0	117.3 ± 40.0	39.4 ± 7.7	94.8
Pseudo-Random	YOLOn	34.0	82.3 ± 15.7	124.3 ± 40.0	39.6 ± 7.0	97.6
Pseudo-Random	MBv2-SSD	42.0	85.0 ± 15.4	133.0 ± 31.3	40.0 ± 5.9	73.4
Rotate and Measure	GT	1.0	50.3 ± 20.5	125.2	16.3 ± 4.0	100.0
Rotate and Measure	YOLOs	1.0	48.5 ± 20.9	125.2	16.3 ± 4.0	96.0
Rotate and Measure	YOLOn	1.0	48.0 ± 20.8	125.2	16.3 ± 4.0	96.3
Rotate and Measure	MBv2-SSD	1.0	47.5 ± 21.0	124.6	16.3 ± 4.0	67.9

policy completes the task with less physical traversal.

The weaker baselines do not provide the same combination of reliability and selectivity. Wall-following has a visited-cell ratio close to that of AIF Saccade, but succeeds in only 8%–13% of runs. Rotate-and-measure has the lowest visited-cell ratio, about 16%, but succeeds in only 1% of runs. These results are consistent with the fact that both policies follow trajectories that are not guided by target evidence.

Figure 5.1 reports success rate and mean time to success for this environment.

The learned detectors do not change the main conclusion in this environment. With both YOLO variants, AIF Saccade remains close to the ground-truth condition across success, target recovery, completion time, and traversal. MobileNetV2-SSD reduces claim precision more noticeably, but AIF Saccade still completes all runs. This suggests that, in the floor-only setting, the simple AIF policy is robust to the detector variation considered here.

Overall, the floor-only results support the simple AIF saccade policy as an effective perception-driven search strategy. Spiral exploration remains a strong geometric baseline, but it does not reach full reliability and it requires much broader

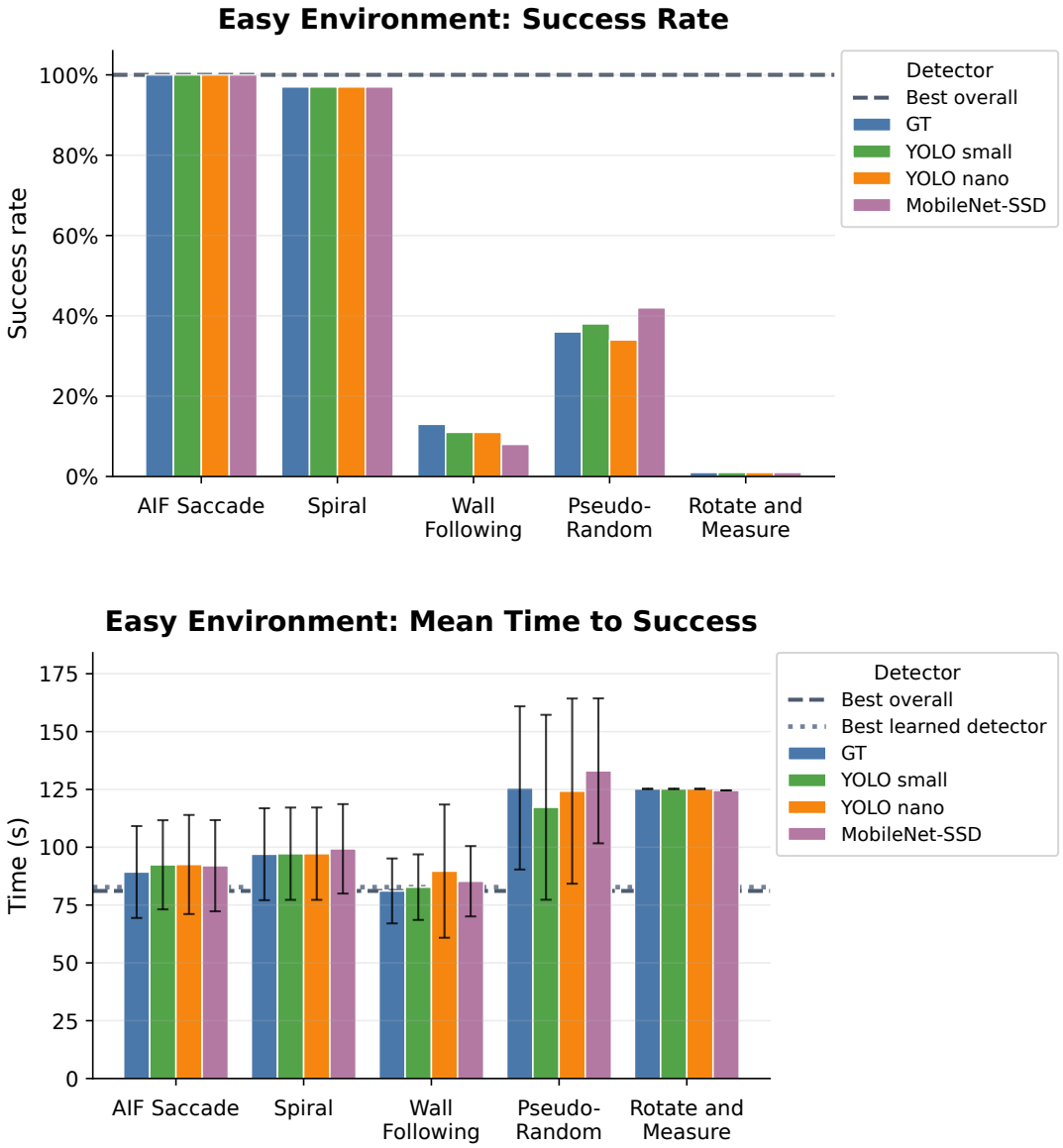

Figure 5.1: Success rate and completion time in the floor-only mixed environment. The plots summarize full-task completion and completion time for the floor-only mixed-object setting.

Table 5.2: Results in the table environment. Recovery, time, and visited-cell ratio are reported as mean ± standard deviation. Mean time to success is computed only over successful runs. YOLOs denotes YOLO11-small.

Policy	Detector	Success (%)	Recovery (%)	Time (s)	Visited (%)	Precision (%)
AIF Saccade	GT	49.0	87.2 ± 15.5	145.5 ± 29.1	52.7 ± 8.9	100.0
AIF Saccade	YOLOs	44.0	85.0 ± 16.5	147.3 ± 26.4	52.9 ± 7.5	97.7
AIF Saccade (ZOI)	GT	87.0	97.7 ± 6.3	118.6 ± 25.5	33.7 ± 9.4	100.0
AIF Saccade (ZOI)	YOLOs	81.0	96.8 ± 6.6	126.5 ± 30.8	36.2 ± 9.8	98.0
Spiral	GT	75.0	94.2 ± 11.9	112.5 ± 20.8	55.3 ± 7.6	100.0
Spiral	YOLOs	67.0	92.5 ± 12.6	113.5 ± 19.9	56.2 ± 7.4	98.4

traversal.

5.3 Table Environment

Table 5.2 reports the results for the `mixed_tables` environment. The zone-of-interest AIF policy gives the strongest result in this setting, reaching 87% success with ground-truth detections and 81% success with YOLOs. This is a large improvement over the simple AIF policy, which reaches 49% and 44%, and over Spiral, which reaches 75% and 67%.

The main difference from the floor-only setting is that the simple AIF policy no longer provides the strongest search behaviour. It still recovers many targets on average, but it fails to complete more than half of the runs. This is consistent with the limitation discussed in Section 5.1: the policy receives height-aware coverage, but its target belief is not conditioned on table evidence. The result is a poor match between the policy model and the environment structure, where table cells have higher target probability.

The zone-of-interest policy addresses this mismatch. Compared with the simple AIF policy, it increases success from 49% to 87% with ground-truth detections and from 44% to 81% with YOLOs. At the same time, it reduces the visited-cell ratio from about 53% to 34%–36%. The improvement is therefore not obtained by sweeping more of the room, but by changing which regions become attractive to the planner.

Spiral remains a strong baseline in this environment. Its successful runs are completed relatively quickly, with mean times of 112.5 s under ground-truth detections and 113.5 s under YOLOs. However, its success rate is lower than that of the zone-of-interest policy and its visited-cell ratio remains above 55%. The result is therefore not a simple time comparison: Spiral is fast when its trajectory is sufficient for the sampled layout, while the zone-of-interest policy completes more

Table 5.3: Results in the hard mixed environment. Recovery, time, and visited-cell ratio are reported as mean ± standard deviation. Mean time to success is computed only over successful runs. YOLOs denotes YOLO11-small.

Policy	Detector	Success (%)	Recovery (%)	Time (s)	Visited (%)	Precision (%)
AIF Saccade	GT	52.0	88.5 ± 14.0	137.9 ± 30.9	51.9 ± 9.8	100.0
AIF Saccade	YOLOs	47.0	88.2 ± 13.5	142.7 ± 30.2	52.3 ± 9.0	89.2
AIF Saccade (ZOI)	GT	89.0	98.2 ± 5.2	116.5 ± 23.3	32.5 ± 9.6	100.0
AIF Saccade (ZOI)	YOLOs	81.0	96.7 ± 7.1	124.1 ± 27.7	35.8 ± 9.8	85.7
Spiral	GT	67.0	92.2 ± 13.9	113.8 ± 16.7	56.6 ± 6.9	100.0
Spiral	YOLOs	61.0	91.0 ± 14.1	115.0 ± 17.2	57.2 ± 6.9	92.9

runs with substantially less traversal.

Figure 5.2 reports success rate and mean time to success for this environment.

Overall, the `mixed_tables` results suggest that, for the simple AIF policy, height-aware coverage alone is not enough to exploit the table setting reliably. The policy has access to the changed observation geometry, but not to the prior structure that makes table regions important. The zone-of-interest policy improves both reliability and spatial selectivity because its belief model represents the table context that is relevant for the task.

5.4 Hard Mixed Environment

Table 5.3 reports the results for the `mixed_hard` environment. The zone-of-interest AIF policy remains the strongest policy, with 89% success under ground-truth detections and 81% success under YOLOs. The simple AIF policy reaches 52% and 47%, while Spiral reaches 67% and 61%.

The comparison with the simple AIF policy remains similar to the table environment. The simple policy recovers many targets on average, but it does not reliably complete the full task. The zone-of-interest policy increases success by more than 35 percentage points in both detector conditions and reduces the visited-cell ratio from about 52% to about 32%–36%. This again indicates that the table-aware belief model changes the search behaviour, rather than improving completion by increasing traversal.

Spiral is still competitive in completion time among successful runs, but it has lower success and the highest visited-cell ratio in the hard setting. Under YOLOs, for example, Spiral succeeds in 61% of runs while visiting 57.2% of the grid, whereas the zone-of-interest policy succeeds in 81% of runs while visiting 35.8%. This comparison gives the clearest result in the hard setting: the ZOI policy keeps high recovery while remaining much more selective in where it flies.

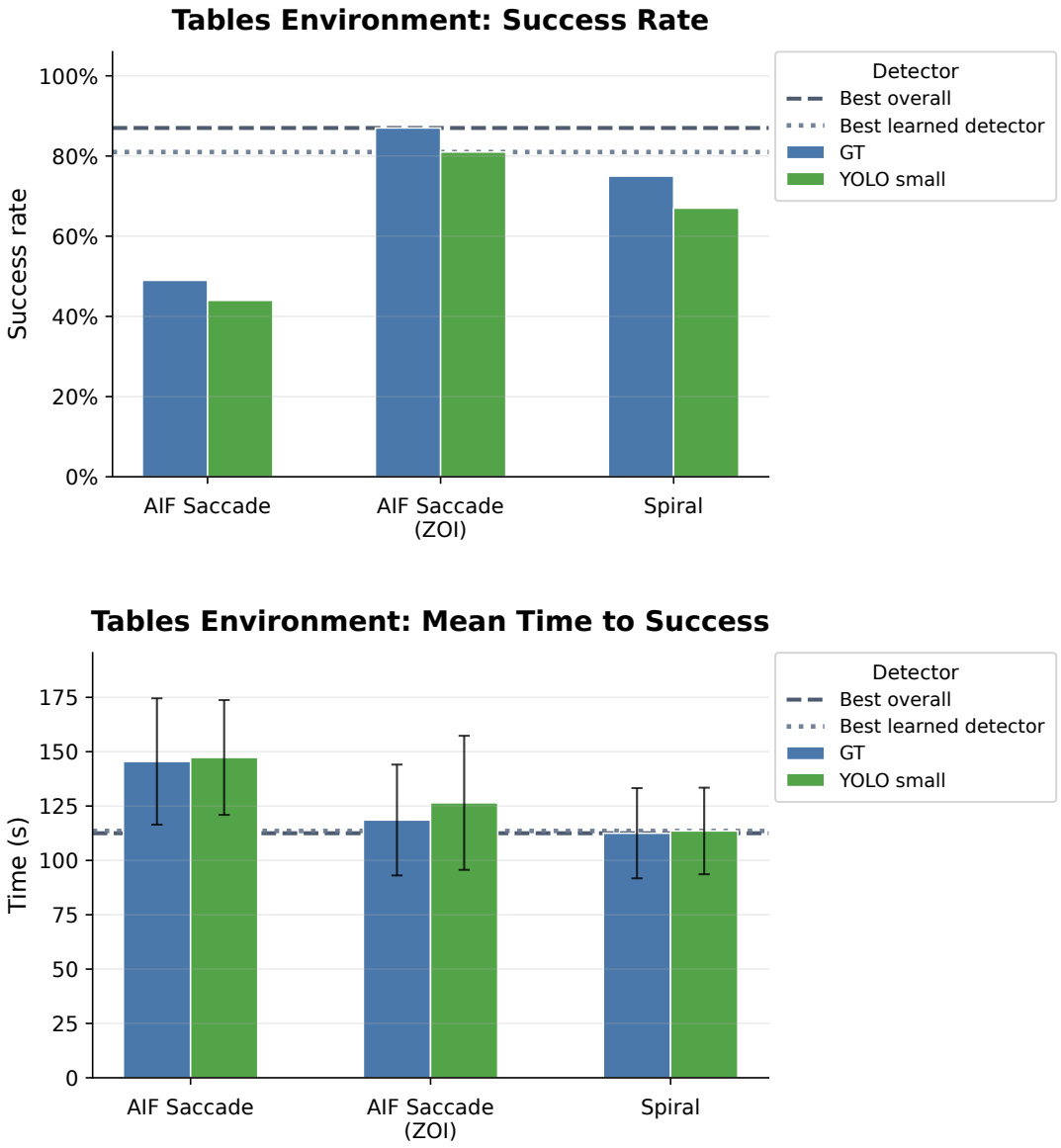

Figure 5.2: Success rate and completion time in the table environment. The plots summarize full-task completion and completion time in the table setting.

Figure 5.3 reports success rate and mean time to success for the hard setting.

Compared with the table setting, the YOLO detector produces a larger reduction in claim precision, especially for the zone-of-interest policy. Nevertheless, the YOLO run keeps the same success rate as in the table environment and maintains high target recovery.

The slightly higher ground-truth success of the zone-of-interest policy in `mixed_hard` than in `mixed_tables` should not be interpreted as evidence that the hard environment is easier. The difference is small and plausibly reflects the stochastic interaction between seeds, table placement, target placement, clutter placement, and trajectory evolution. The safer conclusion is that the table-aware policy remains robust when clutter is added.

Overall, the hard-environment results reinforce the table-setting conclusion. For the simple AIF policy, height-aware coverage does not provide enough structure to exploit raised target regions reliably. The zone-of-interest policy remains the most reliable and spatially selective policy because it uses table context as part of the planning model.

5.5 Qualitative Behaviour

The previous sections report aggregate performance over 100 runs. This section adds example runs to show the movement patterns behind those results. The four baseline policies and the simple AIF saccade policy are shown on the same seed from the `mixed_easy` environment, so those examples refer to the same floor-only target layout. The zone-of-interest AIF example is instead taken from a `mixed_tables` run, since that policy is defined for the table-enriched setting.

Figure 5.4 shows the trajectories of the four baseline policies. Spiral produces the most regular coverage pattern, with a broad sweep that progressively exposes large parts of the grid. Wall-following is much more constrained: it remains close to the room boundary, so completion depends strongly on whether targets are visible from that perimeter path. Pseudo-random motion produces long straight segments interrupted by direction changes, triggered when the distance sensors report a nearby obstacle. Rotate-and-measure is the most spatially limited example, since a considerable part of the episode is spent rotating and selecting a direction before moving again.

Figure 5.5 shows the corresponding occupancy heatmaps. The heatmaps make the traversal patterns visible at cell level: Spiral occupies a broad portion of the room, Wall Following leaves a perimeter trace, Pseudo-Random spreads occupancy irregularly across the grid, and Rotate and Measure concentrates activity in a compact region. These examples also show why low traversal is not automatically a positive result. A policy may visit few cells because it follows a narrow perimeter

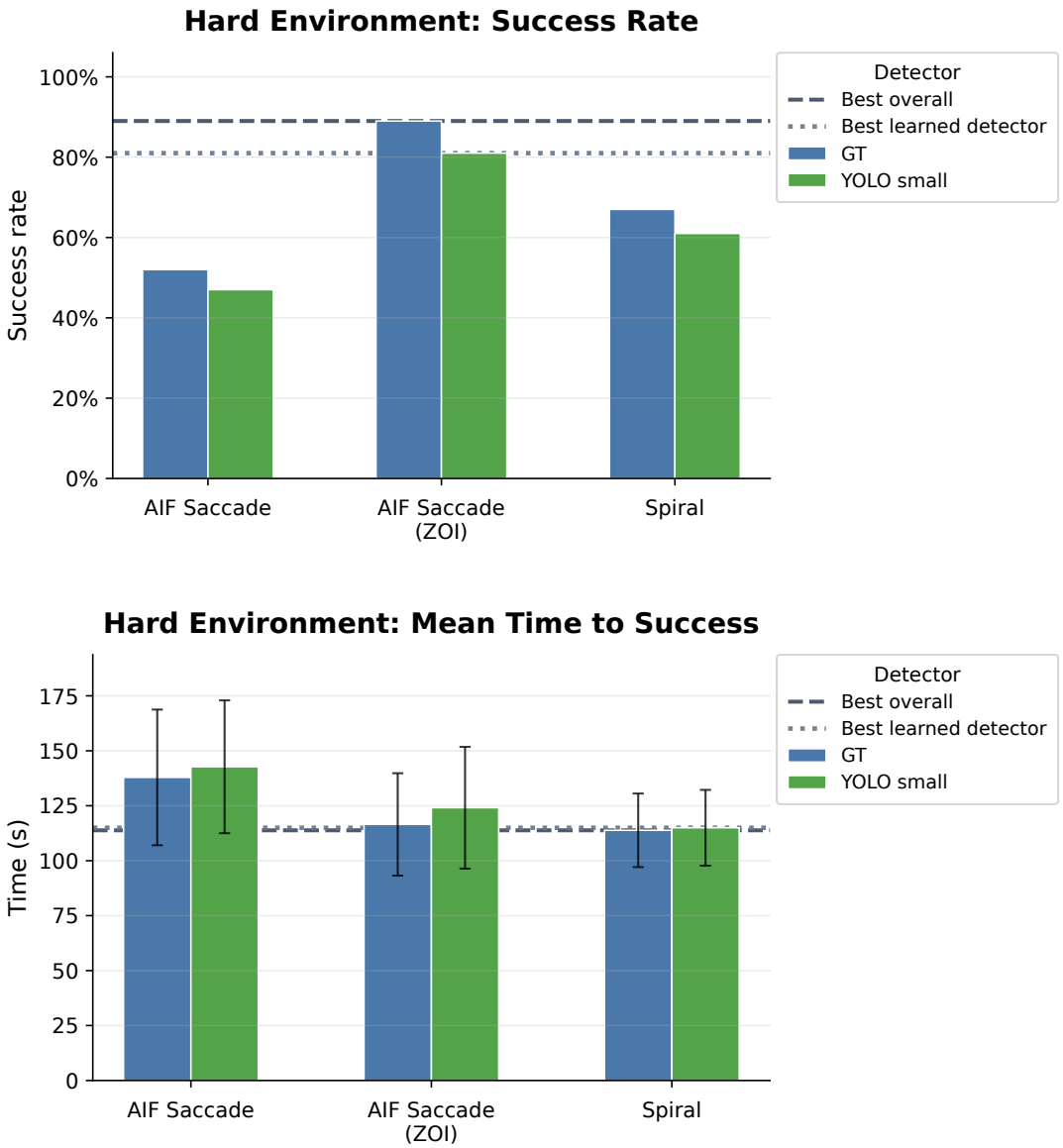

Figure 5.3: Success rate and completion time in the hard mixed environment. The plots summarize full-task completion and completion time when non-target clutter is added to the table setting.

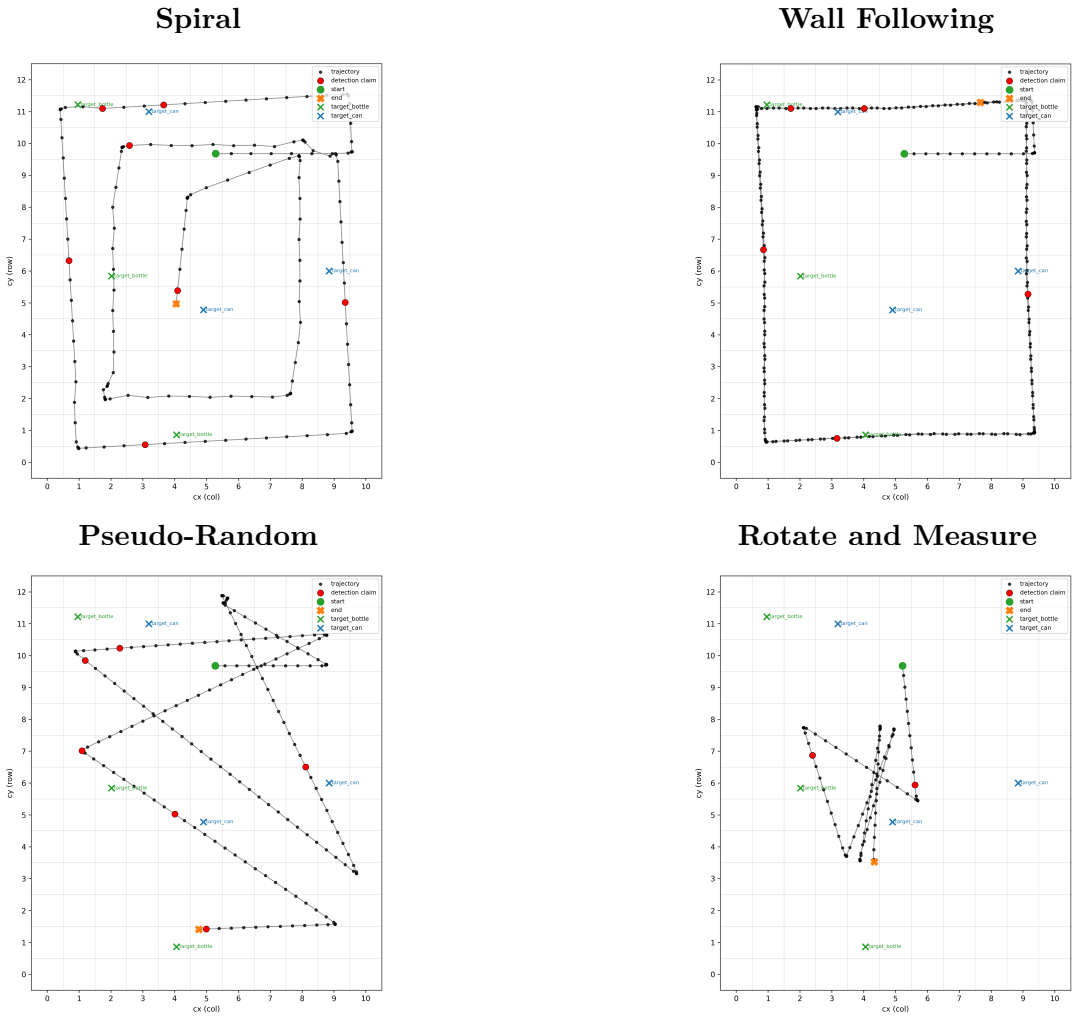

Figure 5.4: Baseline-policy trajectories. All plots refer to the same `mixed_easy` seed. The room is represented as an 11×13 grid of cells with side length 0.5 m.

path or because it remains confined to a small region, not because it has selected the most useful viewpoints. The visited-cell ratio is therefore meaningful only when interpreted together with success and target recovery.

Figure 5.6 compares the simple AIF saccade policy with the zone-of-interest AIF policy. The simple AIF example is from the same `mixed_easy` seed used in Figures 5.4 and 5.5. The zone-of-interest example is from a separate `mixed_tables` run, because the table-aware policy uses table observations and table-conditioned beliefs that are not present in the floor-only environment. In both cases, the trajectories are not fixed coverage patterns: the selected viewpoints change as the policy updates its internal state.

Figure 5.7 reports the final belief maps for a successful zone-of-interest run in the table-enriched setting. The upper map shows the table belief, while the lower map shows the final target-presence map derived from the categorical target belief. In the target-presence map, each cell is rendered using the highest posterior among the target-related states, so validated target cells appear with the same peak value even though the internal representation remains class-specific.

The maps also show that much of the upper-right part of the room remained unobserved: many cells in that region retain their base-prior values in both maps. Since any cell included in the visible mask of a processed observation would receive a belief update, unchanged priors identify regions that the policy did not need to inspect during this successful run. The target map contains the six high-confidence target cells that caused the episode to terminate, and the table map shows that all three table regions were discovered while the policy was searching for those targets.

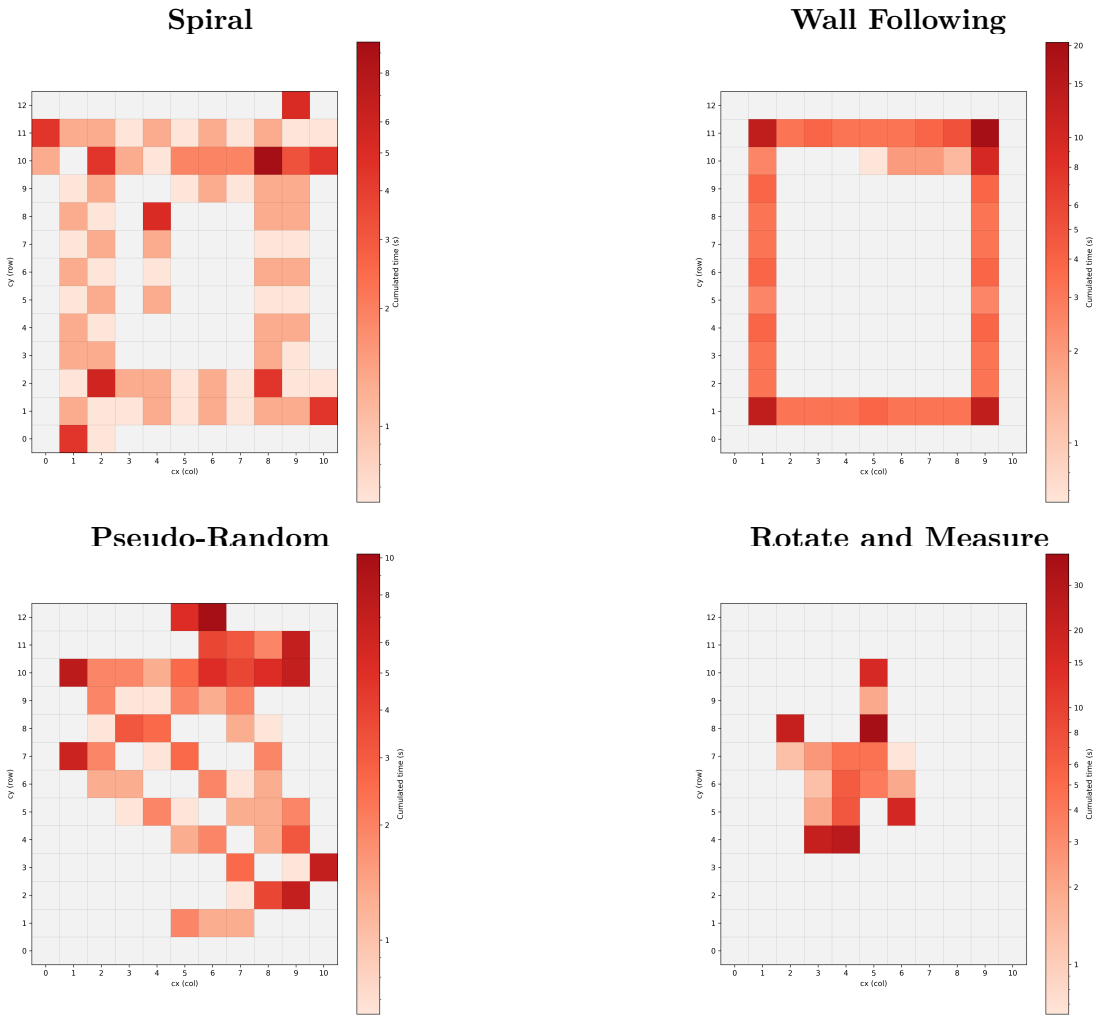

Figure 5.5: Baseline-policy occupancy heatmaps. All plots refer to the same `mixed_easy` seed and report accumulated cell-occupancy time.

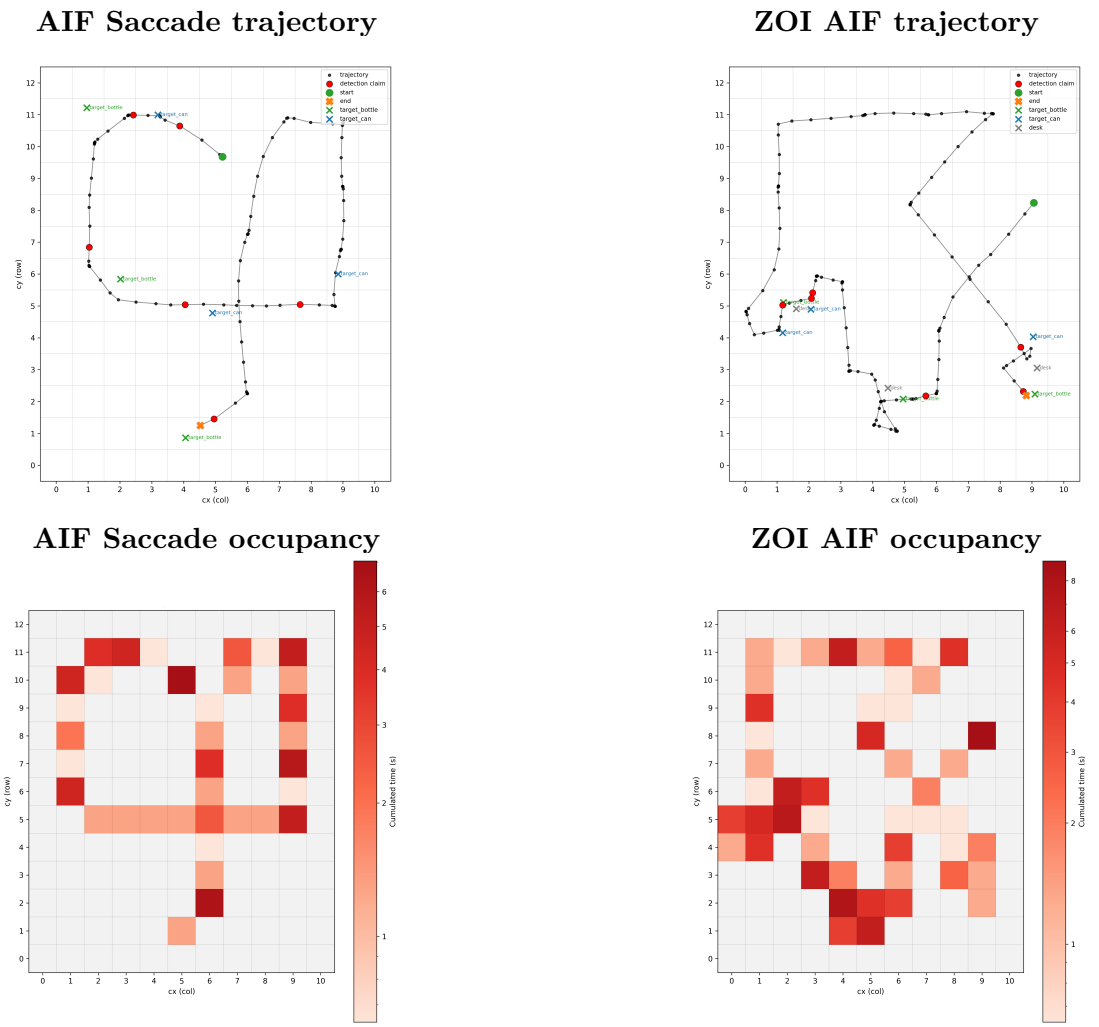

Figure 5.6: AIF trajectories and occupancy heatmaps. The simple AIF saccade plots on the left refer to the same `mixed_easy` seed used for the baseline examples. The zone-of-interest AIF plots on the right refer to a separate completed `mixed_tables` run.

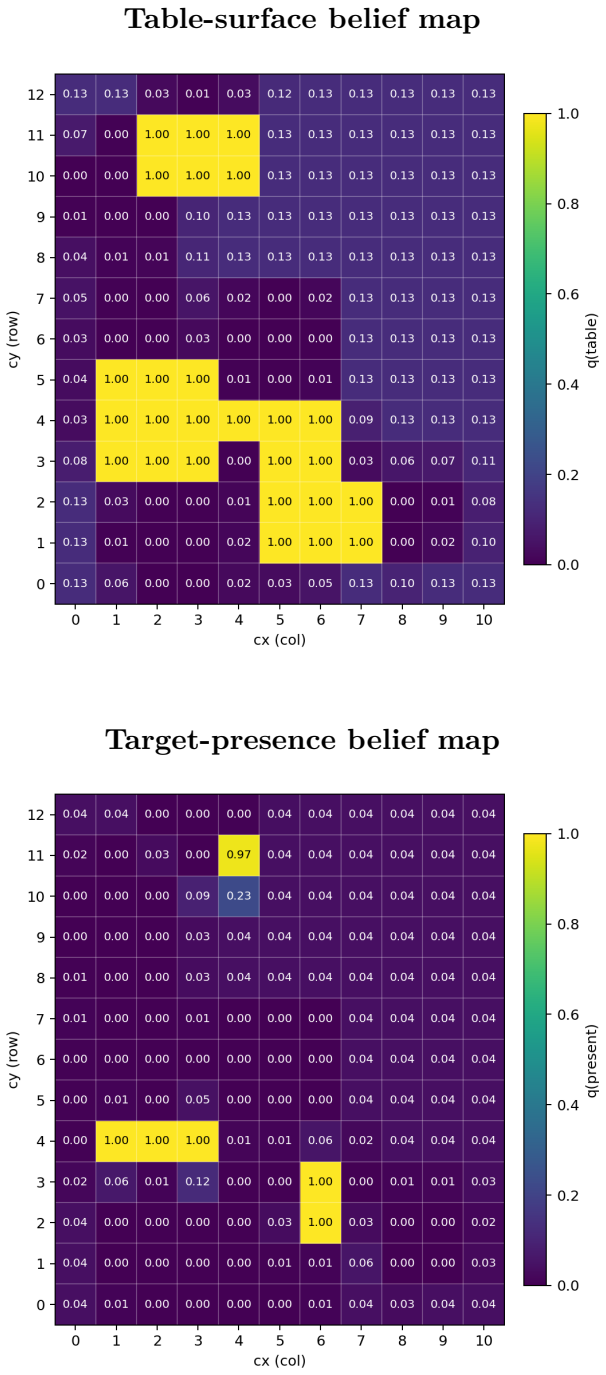

Figure 5.7: Final belief maps for the zone-of-interest AIF policy. The upper map shows the final table-surface belief, while the lower map shows the final target-presence belief for a successful `mixed_tables` run.

Chapter 6

Conclusion

This thesis investigated whether active-inference-inspired navigation policies can improve autonomous object search for a simulated nano-drone. The focus was on policies that use visual evidence during action selection while remaining lightweight and interpretable. Instead of learning an end-to-end mapping from images to actions, the policies developed in this work maintain explicit beliefs over task-relevant hidden states and use those beliefs to choose future sensing actions. This makes it possible to study how target evidence, uncertainty, contextual structure, and movement cost affect the search process.

The results show that this approach can improve search performance over detector-agnostic exploration. In the floor-only environment, the simple AIF saccade policy outperformed the baseline policies and reached full success across all detector conditions. Its advantage over Spiral is especially informative. Spiral is a strong geometric baseline and recovers almost all targets, but it does so by traversing a much larger portion of the room. The simple AIF policy instead uses detector evidence to redirect the search toward cells that are more relevant under its target-presence belief. The result is high task completion with less physical traversal. In this setting, a compact binary belief over target presence is enough to make the search behaviour more selective than fixed or range-based exploration rules.

The table environments show the limit of that simple representation. Once targets are more likely to appear on raised surfaces, the search problem is no longer defined only by whether a cell has produced target evidence. The environment contains contextual structure that changes which observations should matter. The simple AIF policy receives height-aware coverage, but it does not represent table cells as locations with higher target probability. Its reduced performance in the table and hard settings reflects a mismatch between the policy state and the task structure. The observation geometry is more accurate, but the policy still lacks the contextual belief needed to use that geometry effectively.

The zone-of-interest AIF policy addresses this limitation by adding the missing structure to the belief model. Table detections are used to maintain a probabilistic table belief, and that belief conditions the prior probability of target presence. At the same time, fixation-level observations allow the policy to resolve whether a likely target is a bottle or a can. Because the policy represents these class states explicitly, it can define a classification-need term that decreases as the posterior becomes more class-confident, replacing the manual exclusion mechanism used by the simple binary policy. This produces a more natural search pattern: the drone can be attracted by possible target evidence, inspect it more directly, classify it, and then return to search when the belief state no longer assigns high value to that fixation. In both table-based environments, this representation improves success and target recovery while reducing the visited-cell ratio relative to the simple AIF policy and Spiral.

The contrast between the two AIF policies is useful because it shows what the explicit probabilistic formulation makes visible. When the task is floor-only, a compact target-presence belief is enough to improve over fixed exploration rules. When the environment contains raised surfaces that change where targets are likely to appear, the policy must also represent that contextual structure. The contrast points to a practical design trade-off: the state representation should remain compact enough to preserve the lightweight character of the policy, while still representing the sources of uncertainty that actually affect action selection.

The detector results support the same interpretation. Learned detectors reduced claim precision in some cases, especially in the more visually complex setting, but they did not change the main policy ordering. The simple AIF policy remained robust in the floor-only environment, and the zone-of-interest policy remained the strongest policy in the table and hard environments under both ground-truth and learned perception. This suggests that, within the simulated conditions tested here, the design of the policy state and action-selection mechanism had a larger effect on task performance than the difference between ground-truth and trained detector observations.

The main limitation of the work is that the evaluation was conducted in simulation. Webots allowed the system to be tested under repeatable conditions, with controlled object placement, consistent detector interfaces, and supervisor-based validation. These advantages were necessary for isolating the effect of policy design, but they also limit what can be concluded about real-world deployment. The simulator provides ideal pose information, controlled sensing conditions, and a limited set of object categories. The learned detectors were also trained and evaluated within the same simulated visual domain. On a physical nano-drone, several assumptions made in simulation would become weaker. Pose estimates would be noisy rather than exact, camera observations would be affected by lighting and motion blur, and actuation errors could change the relation between planned and

executed viewpoints. Onboard computation would also impose a stricter constraint on detector inference, belief updates, and planning frequency.

The policies themselves also remain hand-designed. Their priors, likelihoods, preference terms, and movement costs were chosen to match the environments studied in this thesis. This makes the system transparent and useful for analysis, but it also means that generalization to richer scenes would require additional modelling work. A more realistic indoor environment might contain several kinds of support surfaces, partial occlusions, different target priors, and spatial layouts not captured by the current grid abstraction. Extending the method would require either designing richer contextual beliefs or learning parts of the generative model from data while preserving the interpretability and efficiency that motivate the approach.

Future work could therefore move in two complementary directions. The first is hardware-oriented: deploying the detector and policy pipeline under real onboard constraints, adding realistic localisation uncertainty, and testing whether the belief updates remain stable with noisier visual evidence. The second is model-oriented: replacing the fixed table-specific context with a more general mechanism for learning which scene structures are predictive of target locations. A longer-horizon planner could also be considered, but it would need to preserve the lightweight character of the current approach rather than turning the policy into a computationally expensive search procedure.

Overall, this thesis shows that active-inference-inspired policies can provide a useful middle ground for nano-drone object search. They are more adaptive than fixed exploration rules because perception influences action selection, but they remain more transparent than end-to-end learned navigation policies because the internal state and action score are explicitly defined. The strongest result is that a compact, interpretable belief model can improve search when it represents the uncertainties that matter for the environment. In this work, that meant target-presence belief for the floor-only task and target belief coupled with table-surface belief for the table-enriched tasks.

Bibliography

[1] Lorenzo Lamberti, Luca Bompani, Victor Javier Kartsch, Manuele Rusci, Daniele Palossi, and Luca Benini. «Bio-inspired Autonomous Exploration Policies with CNN-based Object Detection on Nano-drones». In: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137154. URL: https://doi.org/10.23919/DATE56975.2023.10137154 (cit. on pp. 1, 6, 8, 15, 16, 22, 40, 49).

[2] Mehdi Maboudi, Mohammadreza Homaei, Soohwan Song, Shirin Malihi, Mohammad Saadatseresht, and Markus Gerke. «A Review on Viewpoints and Path Planning for UAV-Based 3-D Reconstruction». In: IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 16 (2023), pp. 5026–5048. DOI: 10.1109/JSTARS.2023.3276427. URL: https://doi.org/10.1109/JSTARS.2023.3276427 (cit. on p. 1).

[3] Ruzena Bajcsy. «Active Perception». In: Proceedings of the IEEE 76.8 (1988), pp. 966–1005. DOI: 10.1109/5.5968. URL: https://doi.org/10.1109/5.5968 (cit. on pp. 2, 23).

[4] Frédéric Bourgault, Alexei A. Makarenko, Stefan B. Williams, Ben Grocholsky, and Hugh F. Durrant-Whyte. «Information Based Adaptive Robotic Exploration». In: Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems. Vol. 1. IEEE, 2002, pp. 540–545. DOI: 10.1109/IRDS.2002.1041446. URL: https://doi.org/10.1109/IRDS.2002.1041446 (cit. on p. 2).

[5] Lancelot Da Costa, Thomas Parr, Noor Sajid, Sebastijan Veselic, Victorita Neacsu, and Karl Friston. «Active Inference on Discrete State-Spaces: A Synthesis». In: Journal of Mathematical Psychology 99 (2020), p. 102447. DOI: 10.1016/j.jmp.2020.102447. URL: https://doi.org/10.1016/j.jmp.2020.102447 (cit. on pp. 2, 18–21).

[6] Devendra Vyas, Nikola Pižurica, Nikola Milović, Igor Jovančević, Miguel de Prado, and Tim Verbelen. «Towards Smart and Adaptive Agents for Active Sensing on Edge Devices». In: arXiv preprint arXiv:2501.06262 (2025). DOI:

10.48550/arXiv.2501.06262. URL: https://arxiv.org/abs/2501.06262 (cit. on pp. 2, 24).

[7] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. «Fully Onboard AI-Powered Human-Drone Pose Estimation on Ultralow-Power Autonomous Flying Nano-UAVs». In: IEEE Internet of Things Journal 9.3 (2022), pp. 1913–1929. DOI: 10.1109/JIOT.2021.3091643. URL: https://doi.org/10.1109/JIOT.2021.3091643 (cit. on p. 6).

[8] Hanna Müller, Vlad Niculescu, Tommaso Polonelli, Michele Magno, and Luca Benini. «Robust and Efficient Depth-Based Obstacle Avoidance for Autonomous Miniaturized UAVs». In: IEEE Transactions on Robotics 39.6 (2023), pp. 4935–4951. DOI: 10.1109/TRO.2023.3315710. URL: https://doi.org/10.1109/TRO.2023.3315710 (cit. on p. 6).

[9] Hanna Müller, Nicky Zimmerman, Tommaso Polonelli, Michele Magno, Jens Behley, Cyrill Stachniss, and Luca Benini. «Fully On-board Low-Power Localization with Multizone Time-of-Flight Sensors on Nano-UAVs». In: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137327. URL: https://doi.org/10.23919/DATE56975.2023.10137327 (cit. on p. 6).

[10] Alper Aydemir, Andrzej Pronobis, Moritz Göbelbecker, and Patric Jensfelt. «Active Visual Object Search in Unknown Environments Using Uncertain Semantics». In: IEEE Transactions on Robotics 29.4 (2013), pp. 986–1002. DOI: 10.1109/TRO.2013.2256686. URL: https://doi.org/10.1109/TRO.2013.2256686 (cit. on pp. 7, 23).

[11] Bitcraze AB. Crazyflie 2.1 Datasheet. Technical Datasheet Rev. 3. Accessed: 10 June 2026. Bitcraze AB, Nov. 2021. URL: https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf (cit. on pp. 7, 8).

[12] Bitcraze AB. AI-deck 1.1 Datasheet. Technical Datasheet Rev. 2. Accessed: 10 June 2026. Bitcraze AB, Apr. 2022. URL: https://www.bitcraze.io/documentation/hardware/ai_deck_1_1/ai_deck_1_1-datasheet.pdf (cit. on p. 7).

[13] Robert Mahony, Vijay Kumar, and Peter Corke. «Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor». In: IEEE Robotics & Automation Magazine 19.3 (2012), pp. 20–32. DOI: 10.1109/MRA.2012.2206474. URL: https://doi.org/10.1109/MRA.2012.2206474 (cit. on pp. 9, 10).

[14] Karl Johan Åström and Tore Hägglund. PID Controllers: Theory, Design, and Tuning. 2nd ed. Research Triangle Park, North Carolina: ISA – The Instrumentation, Systems, and Automation Society, 1995. ISBN: 1-55617-516-7 (cit. on p. 9).

[15] Samir Bouabdallah, André Noth, and Roland Siegwart. «PID vs. LQ Control Techniques Applied to an Indoor Micro Quadrotor». In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. Vol. 3. IEEE, 2004, pp. 2451–2456. DOI: 10.1109/IROS.2004.1389776. URL: https://doi.org/10.1109/IROS.2004.1389776 (cit. on p. 10).

[16] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. «A Review of Physics Simulators for Robotic Applications». In: IEEE Access 9 (2021), pp. 51416–51431. DOI: 10.1109/ACCESS.2021.3068769. URL: https://doi.org/10.1109/ACCESS.2021.3068769 (cit. on p. 10).

[17] Fabio Muratore, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger, and Jan Peters. «Robot Learning From Randomized Simulations: A Review». In: Frontiers in Robotics and AI 9 (2022), p. 799893. DOI: 10.3389/frobt.2022.799893. URL: https://doi.org/10.3389/frobt.2022.799893 (cit. on pp. 10, 11).

[18] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. «Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World». In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133. URL: https://doi.org/10.1109/IROS.2017.8202133 (cit. on p. 11).

[19] Olivier Michel. «Webots: Professional Mobile Robot Simulation». In: International Journal of Advanced Robotic Systems 1.1 (2004), pp. 39–42. DOI: 10.5772/5618. URL: https://doi.org/10.5772/5618 (cit. on p. 11).

[20] Cyberbotics Ltd. Webots Documentation. Version R2025a. Accessed: 5 July 2026. Cyberbotics Ltd. 2025. URL: https://cyberbotics.com/doc/guide/index (cit. on pp. 12, 13).

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. «Deep Learning». In: Nature 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539 (cit. on p. 14).

[22] Richard Szeliski. Computer Vision: Algorithms and Applications. 2nd ed. Cham, Switzerland: Springer, 2022. DOI: 10.1007/978-3-030-34372-9. URL: https://doi.org/10.1007/978-3-030-34372-9 (cit. on p. 14).

[23] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. «End-to-End Training of Deep Visuomotor Policies». In: Journal of Machine Learning Research 17.39 (2016), pp. 1–40. URL: https://jmlr.org/papers/v17/15-522.html (cit. on pp. 14, 25).

[24] Jens Kober, J. Andrew Bagnell, and Jan Peters. «Reinforcement Learning in Robotics: A Survey». In: The International Journal of Robotics Research 32.11 (2013), pp. 1238–1274. DOI: 10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721 (cit. on pp. 14, 25).

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». In: Advances in Neural Information Processing Systems. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html (cit. on p. 15).

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. «SSD: Single Shot MultiBox Detector». In: European Conference on Computer Vision. Springer, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. URL: https://doi.org/10.1007/978-3-319-46448-0_2 (cit. on pp. 15, 16).

[27] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: 2016 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91. URL: https://doi.org/10.1109/CVPR.2016.91 (cit. on pp. 15, 16).

[28] Carlo Marra, Beatrice Alessandra Motetti, Alessio Burrello, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. «BlankSkip: Early-exit Object Detection Onboard Nano-drones». In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). IEEE/CVF, 2026, pp. 3734–3743. URL: https://openaccess.thecvf.com/content/CVPR2026W/EVW/html/Marra_BlankSkip_Early-exit_Object_Detection_onboard_Nano-drones_CVPRW_2026_paper.html (cit. on pp. 15, 23).

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. «MobileNetV2: Inverted Residuals and Linear Bottlenecks». In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474. URL: https://doi.org/10.1109/CVPR.2018.00474 (cit. on p. 15).

[30] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications». In: arXiv preprint arXiv:1704.04861 (2017). DOI: 10.48550/arXiv.1704.04861. URL: https://arxiv.org/abs/1704.04861 (cit. on p. 16).

[31] Glenn Jocher and Jing Qiu. Ultralytics YOLO11. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics (cit. on p. 17).

[32] Karl J. Friston. «The Free-Energy Principle: A Unified Brain Theory?» In: Nature Reviews Neuroscience 11.2 (2010), pp. 127–138. DOI: 10.1038/nrn2787. URL: https://doi.org/10.1038/nrn2787 (cit. on p. 17).

[33] Karl J. Friston, Jean Daunizeau, James Kilner, and Stefan J. Kiebel. «Action and Behavior: A Free-Energy Formulation». In: Biological Cybernetics 102.3 (2010), pp. 227–260. DOI: 10.1007/s00422-010-0364-z. URL: https://doi.org/10.1007/s00422-010-0364-z (cit. on pp. 18, 19).

[34] Karl J. Friston, Francesco Rigoli, Dimitri Ognibene, Christoph Mathys, Thomas Fitzgerald, and Giovanni Pezzulo. «Active Inference and Epistemic Value». In: Cognitive Neuroscience 6.4 (2015), pp. 187–214. DOI: 10.1080/17588928.2015.1020053. URL: https://doi.org/10.1080/17588928.2015.1020053 (cit. on pp. 19–21).

[35] Francesco Giuliari, Alberto Castellini, Riccardo Berra, Alessio Del Bue, Alessandro Farinelli, Marco Cristani, Francesco Setti, and Yiming Wang. «POMP++: POMCP-Based Active Visual Search in Unknown Indoor Environments». In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021. DOI: 10.48550/arXiv.2107.00914. URL: https://arxiv.org/abs/2107.00914 (cit. on p. 24).

[36] Toon Van de Maele, Tim Verbelen, Ozan Çatal, Cedric De Boom, and Bart Dhoedt. «Active Vision for Robot Manipulators Using the Free Energy Principle». In: Frontiers in Neurorobotics 15 (2021), p. 642780. DOI: 10.3389/fnbot.2021.642780. URL: https://doi.org/10.3389/fnbot.2021.642780 (cit. on p. 24).

[37] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. «ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects». In: arXiv preprint arXiv:2006.13171 (2020). DOI: 10.48550/arXiv.2006.13171. URL: https://arxiv.org/abs/2006.13171 (cit. on p. 25).

[38] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. «Object Goal Navigation using Goal-Oriented Semantic Exploration». In: Advances in Neural Information Processing Systems. Vol. 33. Curran Associates, Inc., 2020, pp. 4247–4258. URL: https://proceedings.neurips.cc/paper/2020/hash/2c75cf2681788adaca63aa95ae028b22-Abstract.html (cit. on p. 25).