Direct Learning of Home Vector Direction

Incited by Existing Insect-Inspired Approaches for Local Navigation and Wayfinding

Michiel Vital M. Firlefyn



Delft University of Technology

TU Delft

Direct Learning of Home Vector Direction

Incited by Existing Insect-Inspired Approaches for Local Navigation and Wayfinding

by

Michiel Vital M. Firlefyn

Student Name	Student Number
M.V.M. Firlefyn	4558774

Supervisor:	Dr. G.C.H.E. de Croon
Instructor:	J.J. Hagenaars MSc
Project Duration:	September, 2022 - September, 2023
Faculty:	Faculty of Aerospace Engineering, Delft

Cover:	Brown and Yellow Bee Perched on Flower by Patrick Hendry (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

Bees are essential for all organic life that we enjoy around us. In recent years, much attention has been given to their foraging capabilities. “What can they do for us?” is asked frequently. It is generally acknowledged that these little wonders keep the hierarchical food chain in tact. A fragile equilibrium that has formed the basis for most organic life on Earth, so why not learn as much as possible from their exploits. One day, it may be too late to do so. Better understanding of the realm of insect navigation could provide these animals with environments more tailored to their specific needs and make us all profit in the process.

Readers who feel inspired by the first paragraph are invited to start reading Part II to learn more about the fascinating navigational behavior of bees, wasps, and ants. If technicalities spike your interest more, Part III might be more to your tastes. It describes the complete process on how to attain and implement a hypothetical bio-inspired navigation strategy.

Some lines need to be dedicated to a well-deserved expression of gratitude. Much praise to Prof. Dr. Guido de Croon for taking time out of his busy schedule to guide me along this journey. Also, many kudos to Jesse Hagenaars for patiently answering my many questions and providing helpful tips along the way. Without their support this project would never have been possible. At home, everybody got their fair share of interesting insect facts and confusing jargon. I am forever grateful to my girlfriend, parents and friends for keeping me and always supporting me the past few years.

Michiel Vital M. Firlefyn
Delft, August 2023

Summary

The insect brain is orders of magnitude smaller than their mammal counterparts, which brings forth many complications in studying them. Often, it is physically not possible to study an insect's brain activity on a neuronal level. On the other hand, because of the very limited cranal capacity, their navigational behavior provides the most parsimonious guidance solutions that are available anywhere. Thus, researchers are presented with a dilemma. Bee analogs, which are extraordinary navigators, are very desirable to be studied, but not a whole lot of means are available to study them.

Attempts have been made in the past to speculate how these foragers go about their exploits. A common strategy among researchers is to form a hypothesis on how insects navigate and verify whether analogous behavior is observed whenever the idea is implemented experimentally, often in a robotic substitute. At the moment, it can be concluded that insects do not have an innate ability to keep track of their surroundings. They learn the environment, starting from their initial emergence from the nest. Visual odometry is combined with orientation cues to form a mental track of the previously conducted movement. Over their lifetime, bees become less reliant on path integration and supplement their guidance with visual memories.

This thesis shows that learning home vectors with modern neural architectures is not only possible, but can be a preferred navigational method compared to existing bio-inspired alternatives. The developed bearing convolutional neural network solves many of the pitfalls of its predecessors and simplifies the process of learning the way back to the hive. Relative home bearing inference, meaning the angle between the agent gaze direction and the home vector, is conducted very successfully for various nest locations. Verification and validation is performed on multiple acquisition trajectories in simulated and real forest environments.

Contents

Preface i

Summary ii

Nomenclature iv

List of Figures vi

List of Tables viii

1 Introduction 1

I Scientific Paper 3

II Relevant Literature 62

2 Insect Intelligence: Findings from Biology 63
2.1 Understanding the Modeling Effort . . . 63
2.2 Embodiment and Sensorimotor Integration . . . 64
2.3 How Bees and Wasps Conduct Their Learning Flights . . . 66

3 Insect-Inspired Robotics 70
3.1 From Biological Observations to Navigational Algorithms . . . 70
3.2 Different Types of Navigational Algorithms . . . 71
3.2.1 Local Navigation Based on Snapshot Model . . . 72
3.2.2 Local Navigation through Scene Familiarity . . . 75
3.2.3 Way-finding Strategies . . . 76
3.3 Identifying the Research Gap . . . 77

4 Literature Synthesis 78
4.1 Biological Observations . . . 80
4.2 Robotic Implementations . . . 81

III Preliminary Experiments 83

5 Methods and Tools 84
5.1 Neural Architecture . . . 84
5.2 Simulation and Environment . . . 86

6 Home Bearing Learning and Inference Strategy 89

7 Experimental Results 92
7.1 Experiment 1: Impact of Changing the Nest Position . . . 92
7.2 Experiment 2: Sensitivity Analysis . . . 96

8 Conclusion 100

Bibliography 101

Nomenclature

Abbreviations

Abbreviation	Definition
3D	Three-Dimensional
ALV	Average Landmark Vector
ANN	Artificial Neural Network
ANN	Artificial Neural Network
BA	Bug Algorithm
CNN	Convolutional Neural Network
CPU	Central Processing Unit
FC	Fully Connected
FoV	Field of View
GPU	Graphics Processing Unit
MBON	Mushroom Body Output Neurons
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NN	Neural Network
PI	Path Integration
PIM	Pixel Index Map
POL	POLarization
POL-OP	POLarization-OPponent
PV	Proportional Vector
RAM	Random Access Memory
RCNN	Recurrent Convolutional Neural Network
RGB	Red, Green, Blue
RGBD	Red, Green, Blue, Depth
RL	Reinforcement Learning
ROI	Region Of Interest
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
SNN	Spiking Neural Network
SVD	Singular Value Decomposition
SVD	Singular Value Decomposition
TBL	Turn-Back-and-Look (behavior)

Symbols

Symbol	Definition	Unit
C_n	Camera pose	[-]
D_n	Straight line distance from subject to nest location	[m]
d	Degree of polarization	[-]
d	Distance catadioptric mirror to camera image plane	[m]
E	Essential matrix	[-]
$\overline{\mathbf{e_i}}$	Two-dimensional, position of i-th landmark edge	[m]
f_x, f_y	Focal lengths of camera image plane	[pixels]
H	Discrete PIM	[-]

Symbol	Definition	Unit
H_{plane}^{env}	Catadioptric projection matrix	[pixels·m]
H_c	Global calibration matrix	[pixels·m]
$\overline{\mathbf{H}}(\overline{\mathbf{x}})$	Two-dimensional home vector	[-]
$h(x)$	Non-linear catadioptric projection function	[-]
I	Total intensity	[W/m^2]
I_n	Image	[pixels]
K	Arbitrary constant gain	[s]
K_c	Perspective calibration matrix	[pixels]
$\overline{\mathbf{L_i}}(\overline{\mathbf{x}})$	Two-dimensional landmark vector of i-th landmark	[-]
lr	Latus rectum of catadioptric mirror	[m]
M_c	Catadioptric mirror behavioral matrix	[m]
n	Timestep	[s]
n	Total number of landmarks in scene	[-]
P_n	Step distance between two consecutive subject locations	[m]
$p(\phi)$	Output of POL-OP units	[-]
R_c	Global rotiation matrix	[-]
$R_{n,n-1}$	Rotation matrix	[-]
r, θ	Polar coordinates in camera image plane	[pixels]
s	Arbitrary scaling factor	[-]
$s(\phi)$	POL-sensor output	[-]
$\frac{\mathrm{d}s}{\mathrm{d}t}$	Subject linear speed	[m/s]
$T_{n,n-1}$	Rigid body transformation matrix	[-]
t	Time	[s]
$t_{n,n-1}$	Translational vector	[-]
u, v	Cartesian coordinates in camera image plane	[pixels]
u_0, v_0	Optical center of camera image plane	[pixels]
W_{in}, H_{in}	CNN input dimensions	[pixels]
W_{kernel}, H_{kernel}	Kernel dimensions	[pixels]
W_{out}, H_{out}	CNN output dimensions	[pixels]
X_n, Y_n	Cartesian coordinates of subject position	[m]
x_{gaze}, y_{gaze}	Subject gaze direction	[-]
x_{home}, y_{home}	Home vector, direction of nest	[m]
x, y, z	Cartesian coordinates of point in environment	[m]
$\hat{x}, \hat{y}, \hat{z}$	Rows of H_{plane}^{env}	[-]
$\overline{\mathbf{x}}$	Two-dimensional subject position	[m]
$\overline{\mathbf{x_i}}$	Two-dimensional position of landmark i	[m]
α_n	Subject direction change with respect to previous direction	[rad]
δ_n	Angle between next subject position and line connecting subject and home location	[rad]
θ_n	Subject compass bearing	[rad]
$\theta_i(\overline{\mathbf{x}})$	Two-dimensional sweep angle of i-th landmark	[rad]
$\frac{\mathrm{d}\theta}{\mathrm{d}t}$	Subject angular speed	[rad/s]
ν, ξ, p	Parametric geometry of catadioptric mirror	[m]
τ	Camera image plane FoV	[rad]
Φ_n	Home vector bearing with respect to nest location	[rad]
ϕ	POL-sensor orientation with respect to the solar meridian	[rad]
ω	Relative angle between home vector and gaze direction	[rad]
ω_n	Home vector bearing with respect to subject gaze direction	[rad]

List of Figures

2.1 Depiction of arbitrary path integration model variables, taken from [1]. Coarse dotted lines depict the trajectory of the specimen up till n-1 timesteps. Full lines show trajectories in between timesteps. 64
2.2 Schematic representation of learning flight and TBL time traces. (A) Learning flight is split into 2 components: (A.1) orientation flight or TBL and (A.2) exploration flight or circling. Schematics are an approximation of the observed behavior. Arrows indicate flight direction. (B) TBL time trace, side view, of foraging bees extracted from video recording. (B.1) First TBL of individual bee. (B.2) Departure when TBL is no longer required. (C) TBL time trace, top view, of ground-nesting solitary wasp (*Cerceris*) extracted from video recording. Circles and dashes represent head and body respectively. Numbers are chronologically ordered according to flight trajectory. The nest is indicated by a red box/cross. (B) and (C) are taken from [2]. 68

3.1 The snapshot model and its derivations. (A) The original snapshot model, home vector contributions are unit length. (B) The proportional vector model, home vector contributions are proportional. (C) The average landmark vector model, only rotational home vector contribution. For (A), (B), and (C): (*.1) horizontal view at goal position and (*.2) horizontal view at subject's current position. Nest position is indicated by the red cross. Teal, orange, and purple circles are landmarks (numbered). Teal, orange, and purple arrows are home vector contributions. Blue dashed arrows are home vectors. Black and red arrows are the average landmark vectors. (D) Horizontal views taken at different reference frames. Teal, orange, and purple circle areas are projections of the landmarks (numbered), not always to scale due to concentric view. Snapshot circles and rectangular boxes represent the subject's head and body respectively. Adapted from [3] and [4]. 74

5.1 CNN Architecture with example input and feature map sizes. Dimensions are given per layer. Kernel weights are portrayed at half opacity. Teal layers are convolutional and blue layers are fully connected. The last node has an output value in [-1,1], encoding the relative home bearing. 85
5.2 Summary of omni-directional image construction. (A) Unfolded cubemap of 6 perspective images with 90°FoV, image order from left to right: left, front, right, back. Dorsal image is on top and ventral image on the bottom. (B) Catadioptric projection of cubemap according to [5], green circles depict the region of interest with radii of 200 and 400 pixels. (C) Unfolded catadioptric image that is used as input for the CNN. 88

6.1 Schematic of an example trajectory for the online training and visual homing of an intelligent agent. Nest position is indicated by the red cross. The black solid line shows the flight trajectory of the learning flight (planar). The dashed black line displays an arbitrary foraging flight. The opaque line is the trajectory resultant from the PI home vector. The dotted blue line is the flight path associated with visual homing. Arrows indicate the flight direction. 91

7.1 MSE error between network output and true angle for nest location (0,0) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location. 93
7.2 MSE error between network output and true angle for nest location (0,1) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location. 94

7.3 Relative home vector bearing and MSE error for nest location (4,4) on a 10x10 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle. 95
7.4 MSE error between network output and true angle, scaled with distance, for nest location (0,0) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location. 95
7.5 The two environments used in the preliminary simulation experiments. 96
7.6 MSE error between network output and true angle for nest location (4,4) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location. 97
7.7 MSE error between network output and true angle for nest location (4,4) on a 100x100 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location. 97
7.8 Relative home vector bearing and MSE error for nest location (4,4) on a 200x200 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle. 98
7.9 Relative home vector bearing , only trained on every other grid point, and MSE error for nest location (4,4) on a 200x200 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle. 98

List of Tables

4.1 Overview of literature relating to how biologists and roboticists interpret insect intelligence. 79

1

Introduction

Imagine yourself visiting an unfamiliar city for the first time. Halfway to your destination, disaster strikes. Your phone is out of battery. In order to walk back to your hotel, a direction must be chosen at every previously encountered intersection. How far can you get relying on stored memories alone? What is your decision strategy? How long does it take for you to ask for directions? Logically, you will pick a direction that feels the most “familiar” whenever triggered to do so. Insects like bees and wasps may as well use a similar decision strategy. Relying solely on their judgement of the experienced view in order to return home. Not a straight-forward task in an environment where every tree or patch of grass may seem ambiguous.

As the brain primarily evolved for the purpose of navigation, studying these small creatures’ cognitive abilities can give more insight in the inner workings of the human brain [6, 7, 8]. In other words, understanding the underlying principles of insect intelligence is the first step in understanding humans’ navigational abilities.

Bees have been known to forage in areas as far as 13.5km from their hive [9]. It is widely believed among the research community that feats such as these are performed by computational mechanisms such as path integration and comparing visual snapshots of the current environment with ”stored” snapshots in memory. Path integration is used extensively during the animal’s early life to build foraging experience and discover her surroundings in general. Relevant navigational computations are performed by neurons in the central complex of the insect. The related sensory memories are stored in the mushroom body. A maturing bee or wasp will rely more and more on visual means in order to get to the food source and back. The concrete implementation of these visual navigation efforts remains a subject of debate.

The learning flights of bees and wasps constitute 2 distinct phases, namely: the *orientation* flight and the *exploration* flight. The orientation flight is initiated when the animal gets a visual cue from dark to light in an unknown environment, e.g. the first emergence from the nest entrance. Immediately the insect will orient her body 180 degrees away from the emergence direction, such that her gaze is fixed on the nest entrance. At this point, she will start a series of flight maneuvers designated the turn-back-and-look behavior (TBL), which are saccadic turns centered around the nest location [2]. TBL transitions into circling, the exploration flight, constituting of a spiraling trajectory upwards. This categorization typifies a fine to coarse trajectory, where the bee or wasp discovers her surroundings in decreasing level of detail. First examining the nest entrance itself (orientation), before learning how to return to the broader nest location (exploration). This thesis focuses on the learning flight as a whole and attempts to achieve the homing performance of bees and wasps by utilizing the same visual input as if observed in their natural environment. Note that it is assumed that insects maintain a continuous notion of the home vector through the sensory input they acquire.

Several researchers have contributed to the topic of insect navigation. The first known model to consider insect-inspired navigation is the snapshot model by B.A. Cartwright and T.S. Collett [3]. They

argue that: (1) the snapshots must be stored explicitly; (2) recalled at the exact location of where the image was taken; and (3) subsequently matched such that the remembered view corresponds with the one currently experienced. This might be feasible for relatively small environments. However, the storage requirements may become substantial for larger routes. Several solutions such as using Fourier components have been suggested in order to minimize the memory strain. A more favorable strategy was suggested in [4], where instead of storing the images themselves, only vectors need to be stored and compared. The average landmark vector (ALV) idea takes the average vector based on surrounding landmark vectors in order to home towards the nest location. Be aware that it may produce faulty results in cases where the animal is located very close to particular landmarks or whenever a landmark is occluded in a dynamic environment.

To deal with larger areas and more variability in the surroundings, a familiarity score could be assigned to the current snapshot. A high score entices a continuation of the trajectory, whereas a low score will prompt the insect to rotate itself in order to find a more familiar trajectory. This is much like the initial idea of making your way through an unfamiliar city. The rotation to search for more familiar views is cumbersome however, it takes a lot of time and is far from the observed natural behavior. Building further on this familiarity concept, the proposition of an attractive and repulsive familiarity score was formed [10]. This way both scores could be combined to give an indication of the right direction towards the nest.

We propose to directly learn the home vector bearing towards the nest from visual percepts in a simulated and real forest environment. A compact convolutional neural network (CNN) architecture takes rectified catadioptric omni-directional images as input and returns a single output indicating the relative bearing angle, i.e. the difference between the global gaze angle of the insect and the global home direction angle. Note that the single output is replaced by two home vector coordinate outputs in Part I. Learning a particular direction vector has been done before [11], but never by exploiting the strength of modern neural architectures. Preliminary studies include a model that is trained offline and one trained online on simulated visual data will not be far behind. Extra attention is given to the variability of different scenes and the amount of considered gaze orientations. We encourage the reader to proceed and discover more about the first insect-inspired visual homing algorithm that learns the home vector orientation directly before finding its way back to the nest, after foraging, from a currently experienced omni-directional view.

Research question
Can visual homing using learned home vector directions serve as a successful insect-inspired navigation algorithm?

Sub-questions

- What is the algorithm's performance?
- What are the algorithm's limitations?
- How is the home vector learned most effectively?

A successful and effective algorithm is defined as having an absolute error less than 45° on the inferred home vector direction.

This thesis work ranges from insect navigation background information to simulated and real experimentation. More detailed descriptions of biological observations and robotic implementations can be found in Chapter 2 and Chapter 3 respectively. Afterwards, the discoveries associated with these subjects are summarized in a literature synthesis (Chapter 4). Part III starts off by presenting the methods and tools utilized for preliminary experimentation (Chapter 5), whereas current and future bearing inference strategies are layed out in Chapter 6. Finally, Chapter 7 shows the various experimental results and Chapter 8 presents our associated conclusions.

Part I

Scientific Paper

Direct Learning of Home Vector Direction for Insect-inspired Robot Navigation.

M.V.M. Firlefyn[1]*, J.J. Hagenaars[1]**, G.C.H.E. de Croon[1]**

***Abstract*—Insects have long been recognized for their ability to navigate and return home using visual cues from their nest's environment. However, the precise mechanism underlying this remarkable homing skill remains a subject of ongoing investigation. Drawing inspiration from the learning flights of honey bees and wasps, we propose a robot navigation method based on directly learning the home vector directions from visual percepts during the learning flight. Subsequently, the robot will travel away from the nest, come back by odometric means, and eliminate the resultant drift by inferring the home vector orientation from the currently experienced view. In this study, a convolutional neural network is employed as learning mechanism in both simulated and real forest environments. Additionally, a comprehensive performance analysis reveals that the network's homing abilities closely resemble those observed in real insects, all while only utilizing visual and odometric senses. If all images contain sufficient texture and illumination, the average errors of the inferred home vectors remain below 24°. Moreover, our investigation reveals a noteworthy insight: the trajectory followed during the initial learning flight, for sample image acquisition, exerts a pronounced impact on the network's output. For instance, a higher density of sample points in proximity to the nest results in a more consistent return.**

***Index Terms*—visual homing, convolutional neural network, insect-inspired, robot navigation**

I. Introduction

Imagine yourself in the middle of a forest. How accurately can you point to the direction of your home without external aids? Insects of the order Hymenoptera have evolved into experts regarding such high-level navigation tasks. They solely rely on their senses and judgement to return home in an environment where every tree or patch of grass may seem ambiguous.

Zoologists have been studying Hymenopteran behavior for more than a century. In the early Jurassic, ancestors of the honey bee adapted to a parasitoidy lifestyle [1]. In order to keep track of the nests of potential hosts, to lay their eggs, these animals needed to possess an exceptionally precise and accurate navigational sense. As such, inspiration for this work is drawn from their descendants: western honey bees (*Apis mellifera*) for their long-distance wayfinding capabilities and ground-nesting solitary wasps (*Cerceris*), which learn to remember the location of their nest in a single learning flight.

[1] Micro Air Vehicle Laboratory (MAVLab), Delft University of Technology, Delft, The Netherlands,
* MSc student.
** Supervisor.
Software packages will be made available once this work is published.

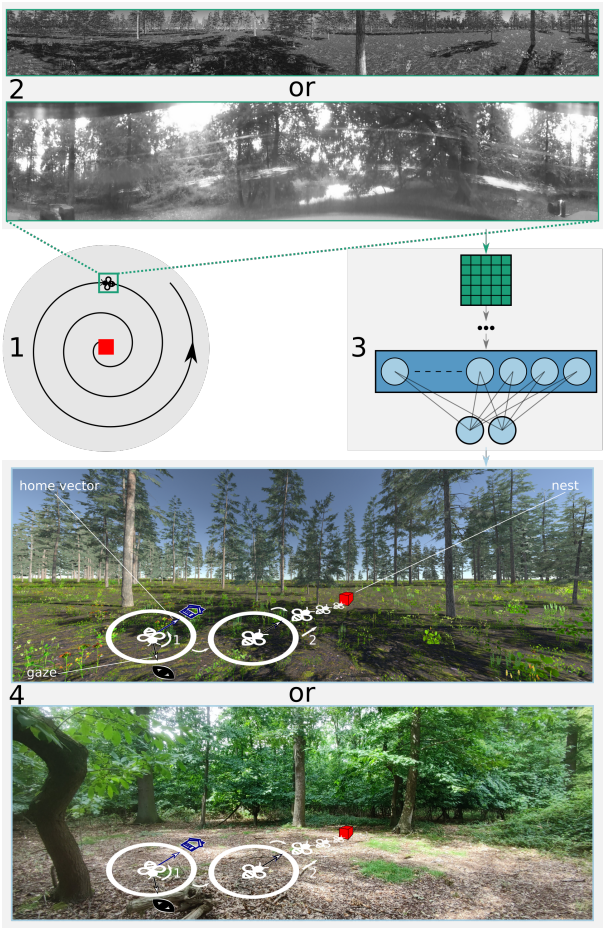


Fig. 1. Direct learning of home vector direction in a simulated or real forest environment. The experimental phases are as follows: (1) acquisition, (2) rectification, (3) learning, and (4) evaluate-to-home. The black numbers indicate the corresponding phase numbers, whereas the white numbers and arrows show evaluation actions. The red cuboid represents the nest location. Please note that all depictions are for illustrative purpose only and are not present during actual experiments.

Insect-inspired robotics aims to replicate the behavior or thought process of insects in their natural habitat [2]. This approach complements experimenting directly with the animals. By starting with a 'clean' slate, researchers can deploy, adjust, and assess the software as needed, providing great flexibility. If the design yields a robot that exhibits behavior with a comparable level of accuracy to that of the virtuoso, i.e. example, animal, then the design can be deemed successful.

The first known model to consider Hymenopteran-inspired navigation is the snapshot model by Cartwright and Collett [3]. Based on their own experiments, which involved differing arrays of landmarks, they observed that visual homing in honey bees relies on visual memories acquired at the nest location. They theorized a framework where the currently perceived horizontal view is compared to the remembered one, resulting in a home vector that is oriented towards the hive. Other variants of this algorithm include: the proportional vector (PV) model, where the home vector is proportional to its radial and tangential components, and the average landmark vector (ALV), which only takes the bearing of the landmark into account, not its apparent size [4]. Distant landmarks provide information on orientation, whereas near landmarks function more like positional cues. Due to cases where those landmarks are occluded or mismatched, all snapshot variants experience decreased performance when distant landmarks are considered. When occluded, the 'pull' of the distant landmark on the home vector orientation is altered, resulting in less consistent home vectors. Additionally, the snapshot and PV model have the limitation that the hive location must be retained in memory. Hafner and Möller realized that the snapshot could be fed to an artificial neural network along with the currently perceived view to achieve more consistent results [5]. To address larger areas and more variability in the surroundings, Baddeley et al. [6] assigned a familiarity score to the snapshots during training. A high score encourages the continuation of the trajectory, whereas a low score prompts the insect to rotate itself in order to find a more familiar trajectory. Since arbitrary rotations are sub-optimal, Le Möel and Wystrach [7] made them unnecessary by applying a repulsive score along with the original attractive one. We propose to directly learn the home vector orientation from training images instead.

When considering how honey bees encode vectors in their brains, we believe they map visual percepts of a site to an internal representation of direction and possibly distance. This idea is reinforced by the fact that honey bee convey this information through waggle dances [1]. Before starting on longer foraging bouts, honey bees emerging from the nest for the first time make initial learning flights around the hive, often in increasing loops. When satisfied, the bee would eventually venture further away in search of nectar. We hypothesize that during her learning flight, the honey bee learns to map her visual percepts directly to the nest direction. Our approach employs a compact supervised convolutional neural network (CNN) architecture that takes rectified catadioptric omnidirectional images around the nest as input. It returns a unit vector representing the relative bearing angle, i.e. the difference between the global gaze angle of the insect and the global home direction angle, expressed through its horizontal and vertical components. After a robot learns the home vector directions, it can travel far outside the range of its learning flights and return based on odometry. Upon return, a large offset between the actual nest and the presumed nest location could have been developed due to odometric drift. Nevertheless, the robot will be able to return home as long as the offset falls within the region explored during the learning flights. We demonstrate our approach within both a simulated and real forest environment, highlighting the successful implementation of bearing-based homing in an online context. Consequently, this work presents the first local navigation algorithm that learns to home directly using only the currently experienced omnidirectional view.

II. Related Work

A. Hymenoptera Ontology

Bees and wasps heavily rely on visual stimuli for navigation. They possess multifaceted eyes called ommatidia, where each ommatidium is associated with a 1-4° horizontal field of view (FoV) [8]. Small specialized ommatidia are located at the dorsal rim of the eye, facilitating the perception of polarized light [1]. Throughout the foraging trip, a sense of the home vector is retained through path integration (PI) [9]. The velocity vector is continuously integrated over time, utilizing the directions of Sun polarization (e-vectors) and visual odometry based on the optic flow across the retinal image [10].

The tasks performed by a worker bee throughout her life primarily include foraging for nectar, constructing, and maintaining the honeycomb structure within the hive. Our focus lies on the activities she engages in while outside the nest. The outgoing trips are guided by PI along the route, covering distances of up to 13.5 km [11]. Upon returning to the nest, a significant odometric drift could have been accumulated over the journey due to wind. This discrepancy would have to be corrected to ensure consistent returns to the colony. The bee's exceptional navigation skills are not innate but rather a result of a complex learning process [12]. She learns to navigate back to the hive by interpreting visual memories acquired during the initial learning flight, which is triggered by the transition from a dark to an illuminated environment [13].

From observations and harmonic radar tracking, it can be deduced that the initial learning flight of honey bees and ground-nesting wasps consists of two distinct phases: the orientation flight phase (Fig. 2 A) and the exploration flight phase (Fig. 2 B). The orientation flight, also known as the turn-back-and-look (TBL) behavior, involves a 180° turn to orient the gaze towards the nest entrance. The animal then briefly hovers in place before performing saccadic turns while alternately moving away from and converging towards the nest entrance [14]. Subsequently, the exploration flight phase, also referred to as 'circling' behavior, is initiated [15]. Although this phase has been examined by Zeil, a trajectory trace remains to be documented [16]. Thus, to infer its spatial geometry, we rely on our own impressions from video material and personal anecdotes. We assume that the trajectory forms a conical spiral that converges at the hive and exhibits a linear relationship between radius and height, as depicted in Fig. 2 B. In essence, the TBL behavior appears to reflect the gathering of detailed features to identify the nest entrance (very close to the nest, smaller than $\sim$1 meter), while circling suggests the collection of coarser features for visual homing (in the vicinity of the nest, $\sim$1 meter or above).

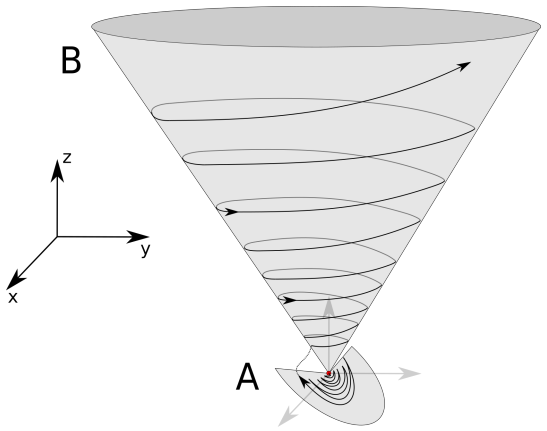


Fig. 2. Schematic representation of learning flight, divided into two components: (A) orientation flight or TBL and (B) exploration flight or circling. The schematic is an approximation of the observed behavior. The nest is depicted as a red point. Arrows indicate flight direction.

B. Insect-Inspired Homing

Robotic platforms have been developed to replicate the homing behavior observed in hymenopteran insects. In a series of feeding experiments, Collett and Cartwright [3] trained bees on either a single landmark or an array of landmarks in various configurations. Their findings indicated that bees are primarily concerned with the apparent size of the landmark and its relative bearing on their retina. This suggests that bees form a mental snapshot of the surrounding objects and terrain at the goal location. A unit home vector is then derived by applying radial and tangential transformations to align the observed retinal image with the stored snapshot.

Lambrinos et al. [4] conducted a robotic experiment of the snapshot hypothesis using a wheeled robot equipped with omnidirectional image acquisition. Their set-up consisted of black cylinders positioned on a flat 40×40 meter desert terrain. The test included the proportional vector (PV) variant of the snapshot model, which encodes the goal condition as vector length. Additionally, they proposed the average landmark vector (ALV) as an alternative representation instead of directly storing the nest image in memory. To prevent unnecessary rotations associated with aligning reference frames, the image could be parameterized using Fourier decomposition [17]. Although snapshot variants address many limitations of the original model, they still face challenges when visual noise or distant landmarks are introduced to the testing area [3]. Cues that have a large impact on home vector direction, such as landmark occlusion or mismatches, appear to be the most common obstacles.

Hafner and Möller [5] demonstrated the efficiency and simplicity of learning the home vector by visually guiding a wheeled robot to its target location. They employed a snapshot-inspired approach, where a multi-layer perceptron (MLP) was trained to produce a population encoding, representing 8 possible homing directions, based on concatenating the current and home views. A model utilizing a perceptron with delta rule learning the vector coordinates of a normalized homing vector proved to be more sensitive to noise compared to the MLP. Hafner and Möller proposed a target-driven visual homing approach, taking the concatenation of the home view and the current view as input. Their simulation environment was small and limited to a white background with black cylinders as landmarks, undoubtedly due to the sparser computational resources at the time. Building upon modern neural architectures and simulation tools, we have developed an algorithm that learns to return home, inspired by naive bees emerging from the hive for the first time.

A framework was developed by Baddeley et al. [6] that abstracts the concept of the snapshot and instead learns a familiarity score based on acquired images along a route. By following the images with a relatively high score, the route can be retraced. While scene familiarity can account for changing landscapes and distant landmarks, it alone cannot provide an accurate indication of the nest direction without a continual stop-scan-go routine. This could be addressed incorporating a repulsive score, which, when combined with the attractive score, yields an angular error indicating the most familiar viewing direction [7]. Van Dalen et al. [18] presented a validation of scene familiarity using data obtained from a micro air vehicle (MAV). Although the concept of attractive/repulsive scene familiarity appears to resolve previous limitations of homing algorithms, one could question the necessity of inferring two abstract measures from an image and subsequently integrating them into a relative angular difference that indicates the local home vector bearing. Alternatively, the same outcome could be achieved by directly learning the bearing from the input image. Consequently, we present the first visual homing algorithm that allows a robot to learn the home directions directly during its learning flight, while only requiring the currently experienced omnidirectional view and odometry to return to its starting location.

III. Methodology

A. Assumptions

The performed experiments encompass an offline experiment conducted in both simulated and real environments and an online experiment, only in simulation. To provide clarity and establish the limitations of the algorithm, it is essential to outline the underlying assumptions. For a comprehensive understanding of each experiment's intricacies, a more detailed explanation is provided in section III-C.

Online and offline experiment assumptions:

1) Planar movement: The experiments assume two-dimensional movement, which simplifies rotations and avoids introducing tilt into the acquired image.
2) Compass available: A compass or a similar source of compass information, e.g. based on sensing polarized light or magnetic fields, is assumed for PI and training labels.
3) No collisions: The cameras are permited to pass through objects as our focus is purely on navigation, for

now. Note that some images are captured in between branches, leading to occlusion of some of the surrounding objects.

4) Catadioptric imaging system: The omnidirectional image construction involves projecting six images from a cubemap to a single image, simulating the view obtained through a parabolic mirror with a latus rectum of 0.1 meter and a distance to the camera image plane of 0.1 meter. The catadioptric projection was performed according to the method prescribed by Berenguel-Baeta et al [19].
5) Polar-to-Cartesian Rectification: The process of rectifying the omnidirectional images involves selecting the pixel area enclosed between two concentric circles and then transforming the pixel coordinates from a polar reference frame to a Cartesian reference frame.

Online experiment assumptions:

1) Predefined goal condition: The inherent algorithm lacks a built-in goal condition, meaning it would continue homing even after passing the nest multiple times, similar to most discussed homing algorithms. To address this, a predefined goal condition can be implemented using a quick response (QR) code or colored square placed on a flat surface, easily detectable by a downward-facing camera.
2) Path integration drift model: Existing literature does not provide a probability distribution that precisely matches the drift experienced by insects during foraging trips. Hence, the discrepancies in PI caused by imperfect sensing or wind are modeled as a normal distribution.

B. Neural Architecture

We employ a compact CNN consisting of two 5×5 convolutional kernels, each with a stride of 4, and one fully connected (FC) layer. The input image is grayscale with a shape of $1\times1\times201\times1800$ (batch size$\times$channels$\times$height$\times$width) following PyTorch convention. It is expanded into a $1\times2\times50\times449$ feature map using a $2\times1\times5\times5$ (output channels$\times$input channels$\times$height$\times$width) filter with a stride of 4. This map is convoluted further into a $1\times4\times12\times112$ feature map using a $4\times2\times5\times5$ filter with a stride of 4. Furthermore, the layer's output is flattened into a 1×5376 (batch size$\times$feature dimension) tensor, which is then passed through an FC layer, converging into two output nodes. The network output nodes hold the x- and y-coordinate of the egocentric home vector relative to the subject's gaze on the unit circle. Moreover, the training loss is chosen to be the mean squared error (MSE) loss. All layers utilize the hyperbolic tangent ($\tanh$) activation function in their respective intermediate outputs. The resulting CNN consists of $11,002$ weights and 8 bias terms.

The format of the home vector, as two separate coordinates, is not chosen arbitrarily. Initial versions of the evaluation error directly encoded the home vector direction to the nest as training labels, resulting in larger than anticipated errors during training. Some angles were very similar in convention but spanned large intervals mathematically, e.g. -175° and 175°. To mitigate this issue, a decomposition of the relative angle into its two-dimensional components on the unit circle was implemented as evaluation error. Note that the angle formed by the home vector coordinates is relative to the subject's gaze, which can be seen in Fig. 3. While all the training labels are defined in the same local reference frame, they represent angles relative to different global reference points. Further explanation of this convention can be found in section IV-A. We define the unit home vector with respect to the subject's gaze angle, i.e. the middle of the currently experienced omnidirectional view. This approach allows the network output to be passed to a robot control system directly.

C. Flight Phases

Photo-realistic simulated data provide a cost-effective means to verify our navigation scheme. The visuals in the simulated scene should resemble a real world foraging area, while the learning flight should include bee-like maneuvers. Before the flight phases of the online experiment could be determined, offline simulations were performed to assess whether the network could successfully infer homing directions. The impact of the network architecture, simulation environment, number of gaze angles, acquisition trajectory, and nest location were studied accordingly.

The online experiment comprises several flight phases that simplify specific aspects of the biological learning flight. These phases are as follows: (1) training image acquisition (exploration flight), (2) outbound trip, (3) inbound trip, and (4) evaluate-to-home. To clarify, the evaluate-to-home phase consists of vision-based homing with the previously learned home directions. An offline simulation differs from its online counterpart due to the exclusion of the outbound/inbound PI trip. Additionally, the network output is used as a control input in the online experiment. The offline simulation tests training performance as the same samples that were used during training are evaluated. To acquire training images, the simulated bee performs an Archimedean spiral while maintaining a constant altitude, ~1.89 units above the terrain. Overall, 100 training samples are taken over four gradually increasing circles around the colony. In a true online implementation, the network would be trained after every image acquisition. However, since a single pass of the true online implementation ensued in 'catastrophically forgetting', we approached this as close as possible by training the network in a single pass over all images in a random order. Once the subject is trained on the acquired data, the simulated foraging trip can be initiated, as if guided by PI. The user controls the movement along the outbound trajectory using a keyboard, while the inbound trajectory involves a straight line towards a perturbed nest location. This perturbation is modeled as zero-mean Gaussian noise, where the variance of the error in the PI trajectory endpoint is approximately equal to the sum of the variances of the directional and rotational errors per step [20]. To ensure accurate results, the perturbed nest location must be situated within the boundaries of the learning flight. In the evaluate-

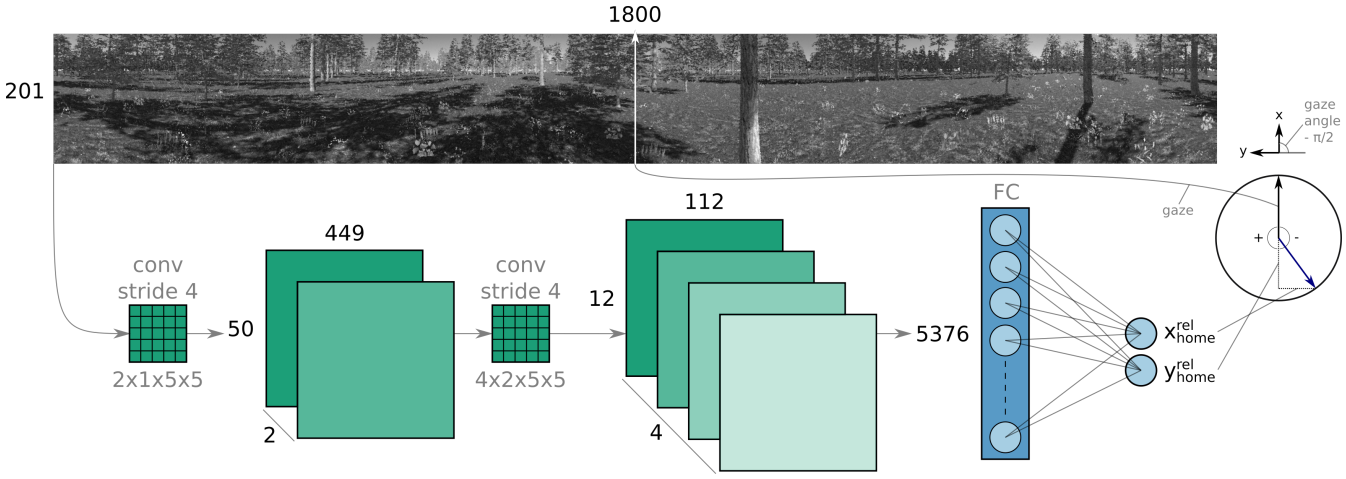


Fig. 3. CNN Architecture with feature map sizes and example input. Dimensions are provided per layer. Kernel specifications are displayed at half opacity. Teal layers represent convolutions and blue layers indicate full connections. The gaze is indicated as a black or white arrow and the home vector as a blue arrow. The last nodes have an output value in the range of $[-1, 1]$, encoding the coordinates of the relative home vector. Note that the presented gaze angle $0°$ corresponds to North/upwards.

to-home phase, the trained CNN is employed to infer the home vector every 0.25 units. These discrete inference steps constitute the visual homing mechanism, which guides the subject from the presumed nest to the true nest location.

IV. Experiments

A. Set-up

A modified version of Flightmare serves as the backbone for the simulation [21]. Flightmare consists of a ROS (C/C++) client[1] and a Unity renderer[2] on the server side. Communication between the client and the server is facilitated by the ZMQ messaging protocol[3]. Unity (C#) provides an editor[4] to rapidly sculpt photorealistic virtual environments. Its main benefit is the asset store with countless object meshes and terrain tools that are readily available. Furthermore, the CNN is implemented using the PyTorch (Python) library[5]. All experiments were performed on an Intel®Core™ i7-8550U CPU @ 1.80GHz. Catadioptric image projection, according to the method described by Berenguel-Baeta et al. [19], was performed in simulation using perspective images with an FoV of $90°$ and a dimension of $1024{\times}1024$ pixels. They were captured from the subject's front, back, left, right, dorsal, and ventral regions.

We considered two acquisition trips, which are both centered around the nest location, throughout the simulated experiments: a square $10{\times}10$ regular grid and an Archimedean spiral with radius $r = 0.25 \cdot \theta$, where θ runs from 0 to 8π. To clarify, the regular grid occupies an area of 100, whereas the spiral encompasses approximately 113. For network training, the chosen hyperparameters include the following: a batch size of 1 and a learning rate of $9 \cdot 10^{-4}$ over 1 epoch. Additionally, every omnidirectional sample image is rectified into 360 distinct sample images to serve as network input. The training labels are computed from geometric relations along one of the predefined two-dimensional trajectories. The associated relative angle ω between the estimated home bearing and gaze direction is derived from two vectors situated at the experimental subject's current location, where the gaze direction is represented by a unit vector (e.g. North is $[0, 1]$). Let us consider two arbitrary vectors with the same origin: $\vec{x}_{\text{home}} = [x_{\text{home}}, y_{\text{home}}]^T$ and $\vec{x}_{\text{gaze}} = [x_{\text{gaze}}, y_{\text{gaze}}]^T$, which indicate the direction of the nest location and the gaze direction respectively. Then, the relative angle between the two vectors can be calculated by taking the two-argument arctan (arctan2) of the determinant of the vectors' augmented matrix divided by their dot product:

$$\omega = \arctan2\left(\frac{x_{\text{gaze}} \cdot y_{\text{home}} - y_{\text{gaze}} \cdot x_{\text{home}}}{x_{\text{gaze}} \cdot x_{\text{home}} + y_{\text{gaze}} \cdot y_{\text{home}}}\right), \quad (1)$$

where the coordinates of the relative home vector on the unit circle $[x^{\text{rel}}_{\text{home}}, y^{\text{rel}}_{\text{home}}]^T$ can be defined as:

$$\begin{bmatrix} x^{\text{rel}}_{\text{home}} \\ y^{\text{rel}}_{\text{home}} \end{bmatrix} = R_z\left(-\omega_{\text{gaze}}\right) \begin{bmatrix} x_{\text{home}} \\ y_{\text{home}} \end{bmatrix}, \quad (2)$$

$$\text{with} \quad R_z\left(-\omega_{\text{gaze}}\right) = \begin{bmatrix} \cos\left(\omega_{\text{gaze}}\right) & \sin\left(\omega_{\text{gaze}}\right) \\ -\sin\left(\omega_{\text{gaze}}\right) & \cos\left(\omega_{\text{gaze}}\right) \end{bmatrix}, \quad (3)$$

where $\omega_{\text{gaze}} = \angle\vec{x}_{\text{gaze}}$ and the coordinates of the relative home vector are used as training labels. Note that Unity does not explicitly state dimensional units. In addition, a coordinate frame conversion must be performed to translate the results of (1) from a right-handed coordinate frame to Unity.

During online evaluation, ω can be used directly to control the amount of rotation so that the subject moves towards the direction of the estimated home vector. Hence, no additional

[1]Flightmare client Github: https://github.com/flightmare
[2]Flightmare server Github: https://github.com/flightmare_unity
[3]ZMQ protocol main website: https://zeromq.org/
[4]Unity editor main website: https://unity.com/
[5]PyTorch library main website: https://pytorch.org/

rotations of the output are required. The goal condition is set to be 0.25 units from the nest. This is assumed to correspond to a condition in which a honey bee or robot would be able to reliably detect and enter the nest by itself. Thus, the number of discrete inference steps to reach the goal condition is utilized in section IV-F as a performance metric for online homing.

B. Offline Simulation

Before initiating the online simulation, it is essential to analyze the network's performance during training. Thus, a simulation environment was created, consisting of a 1000×1000 terrain with small elevations generated by Perlin noise, along with grasses, tussocks, bushes, and flowers. Fig. 4 illustrates the two acquisition trajectories in simulation, where each MAV indicates an acquisition location. Notably, the spiral configuration covers a slightly larger area. The position of the nest is consistently set at $[538, 573]^T$, serving as the global reference point for all presented measurements. Several nest locations and terrains were tested to arrive at the final configuration, which is utilized for all simulated experiments. A bearing map depiction of the home vectors at specific locations is provided at the top right of each figure pane. The unit length deviation of the home vector could be interpreted as the 'confidence' of the network. If the home vector length is not precisely one, it suggests that the network has not converged to an exact solution, even if the relative angle is deemed correct. The angle deviation $\Delta\omega$, i.e. angular error, is defined as the angle between the estimated relative home vector direction $\hat{\omega}$ and the real one ω. Its absolute error can be computed in the same fashion as ω in (1), substituting the Cartesian components of ω and $\hat{\omega}$ and taking the absolute value of the resultant angle. Notably, Unity employs a left-handed coordinate frame and unit-less distance measures, which are reflected in the figure. Additionally, it is important to mention that the supplied performance plots are only relevant for a single gaze angle, equivalent to the positive x-direction (North) as in the example of Fig. 3.

Training the CNN on images captured on a rectangular grid yields a better overall performance compared to the spiral case. This conclusion is drawn from the MSE, which is equal to 3361.3777 for the spiral variant and 2537.6865 for the grid variant, during training on 360 gaze angles (batch size 1 for $35,640$ training samples). However, it is worth noting that the subject should be able to return to the vicinity of the hive in both scenarios, since the angular errors of all locations are below $45°$. Further performance analysis can be found in section IV-C. Remarkably, areas near trees, surrounded by foliage, do not show a significant decline in performance. This can be determined qualitatively from the shaded green areas in Fig. 4. Thus, important landscape features remain interpretable to the subject, even when they are partly occluded. Note that in the already shown and subsequent performance plots the location at which the home vectors are inferred is equivalent to its origin.

Next, it is important to mention that the nest location sample itself was excluded from training because it lacks an associated home vector. This has some implications on the homing performance, which is explained further on in section IV-F. A higher density of sample locations can be observed in the spiral case. Consequently, both trajectories present some advantages. A spiral trajectory resembles the bee-like learning flight most accurately according to our findings. Whereas, the grid provides a consistent pattern that is easily implementable and can be scaled in resolution to evaluate the network's generalization.

C. Offline Verification

Evaluating the generalization of the network needs to be performed on several locations where the subject did not already experience the view. First and foremost, the regular grid utilized during training is repurposed for evaluation, with an enhanced resolution. Both the horizontal and vertical resolution of the grid is made four times smaller, with an intermittent distance of 0.25 instead of 1, resulting in a 40×40 evaluation sites, including the original 100. In contrast, none of the evaluation sites correspond to the training locations used for the spiral configuration. Fig. 5 demonstrates the evaluation performance for four gaze angles on both acquisition trajectories. Once again, training on the grid exhibits superior performance, as can be seen in Table I, but an intriguing phenomenon occurs near the hive location. When the home vectors are interpreted as velocity vectors and integrated over the evaluation grid, a stream plot can be generated. These homing streams converge toward a specific point that does not precisely coincide with the nest location. Interestingly, in the figure for the spiral case (Fig. 5 B), this convergence point lies closer to the hive in all cases. This difference is a result of the higher training location density. Moreover, a circular convergence area takes shape instead of a single point. Such an area creates less of an 'artificial' nest location, as can be seen in the singular convergence point of the grid case. The consequences of sample density near the hive become more evident in the evaluate-to-home step plots of section IV-F.

In the performance graphs for training (Fig. 4), no significant decline in performance could be attributed to regions with abundant foliage. However, the trees do impact the performance of the generalization. From the bearing map and error plots can be deduced that whenever the network evaluates a particular view occluded by tree foliage (home vectors located in green shading), which is not encountered during training, the error will be larger. Regions with high error in the plots of Fig. 5 can therefore be attributed to the presence of occluding trees.

The stream plots offer insights into the potential trajectory of the subject within the experimental environment. As such, their endpoints are a rendition of the algorithm's catchment area, delineating the boundaries for successful homing. It is important to note that these streams represent the home vector as a velocity vector, although this interpretation is not valid for the actual experiment. We specifically employ the home vectors as directional cues, excluding any direct representation of velocity. As previously stated, the length of the home

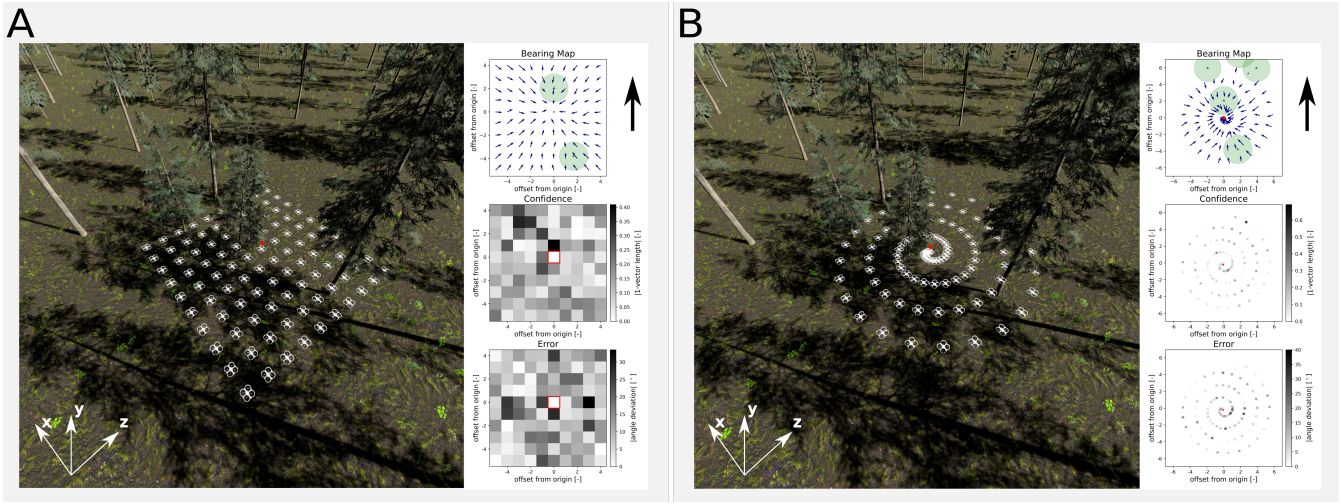


Fig. 4. Simulation environment and training performance for two acquisition trajectories: (A) a 10×10 regular grid and (B) an Archimedean spiral. Each pane presents an aerial view on the left and the corresponding training performance plots on the right. Trees are illustrated as green markers with foliage shaded translucent green. The nest location is depicted as a red point or square. The gaze angle is a black arrow and home vectors are blue arrows.

vector does not signify velocity, as distance data has not been incorporated into the analysis.

D. Network Activation Analysis

Many exisitng insect-inspired visual homing approaches explicitly define which landmarks or features to use. Whereas, we employ a learning approach that will extract the most informative features given the robot's specific environment. This prompts us to question what the network truly learns as landmarks in order to get more insight in the inference process. To address this inquiry, we constructed a flat textured test terrain adorned with a landmark array comprising of three trees of identical height and orientation. This set-up was chosen over a more intricate simulation environment due to the simplicity it offered in the visual analysis. By omitting many elements from the scene, we achieved a clear representation of the features that hold the greatest significance for the network's inference of the home vector.

In our analysis depicted in Fig. 6, we adopted the Grad-CAM method introduced by Selvaraju et al. [22]. The figure presents heatmap-colored training samples, wherein areas of higher gradients of the CNN output with respect to the second convolutional layer activations are denoted by red shading. The landmark rotation has been modeled in accordance with the experiments conducted by Collett and Cartwright [3]. While it is indeed true that bees predominantly regard trees as landmarks for navigation, an intriguing observation emerges. The most critical features that the CNN takes into account for home vector inference are not solely the tree stems, as one might expect. From the grid heatmaps (Fig. 6 A), it becomes evident that the dominant contributor to the forward/backward x-component of the home vector is a horizon cue. Some mild contributions are discernable in the leftmost tree stem, foliage, and shadows. For the left/right y-component of the home vector, the highlighted features are not tree stems as anticipated. Instead, the focus is on specific segments of the foliage and shadows. An essential characteristic of these features is their clear contrast against the sky.

In Collett and Cartwright's experiments, actual bees would anticipate the reward to be situated in a location corresponding with the shifted landmark array. However, our simulated bee appears to deduce that the nest location is in the same location with respect to the central tree, which is situated 10 units in front of the nest location. Our simulated landmark array consists of three trees. It seems that the central tree is interpreted as the dominant landmark. This leads the simulated bee to infer the central tree as a reference point. Once the trees are rotated, the nest location is again inferred as being 10 units below the dominant landmark, but this landmark is not the same tree as before the rotation. The first tree that is encountered almost straight in front of the subject after tree rotation was the right-most tree before the rotation. Dominant landmarks after and before tree rotation are marked with an asterisk in the green and teal foliage shading respectively. Notably, the wrong dominant landmark interpretation persists, despite the 90° CCW rotation of the trees intended to account for the corresponding shift.

Interestingly, the heatmap generated for the spiral trajectory (Fig. 6 B) reveals distinct areas of interest. The horizontal lines within the tree foliage and shadows seem to show interest in the movement of visual percepts compared to a main reference point or object, e.g. a tree and the horizon or two trees. Certain features in trees are highlighted more than others, such as tree foliage. This suggests that intricate patterns of tree foliage set against a clear blue sky take precedence over other tree features in directly learning home vector orientations in this simplified environment.

Furthermore, we emphasize the significance of having objects extend beyond the actual horizon line. When we explored environments featuring only small plants and bushes instead of

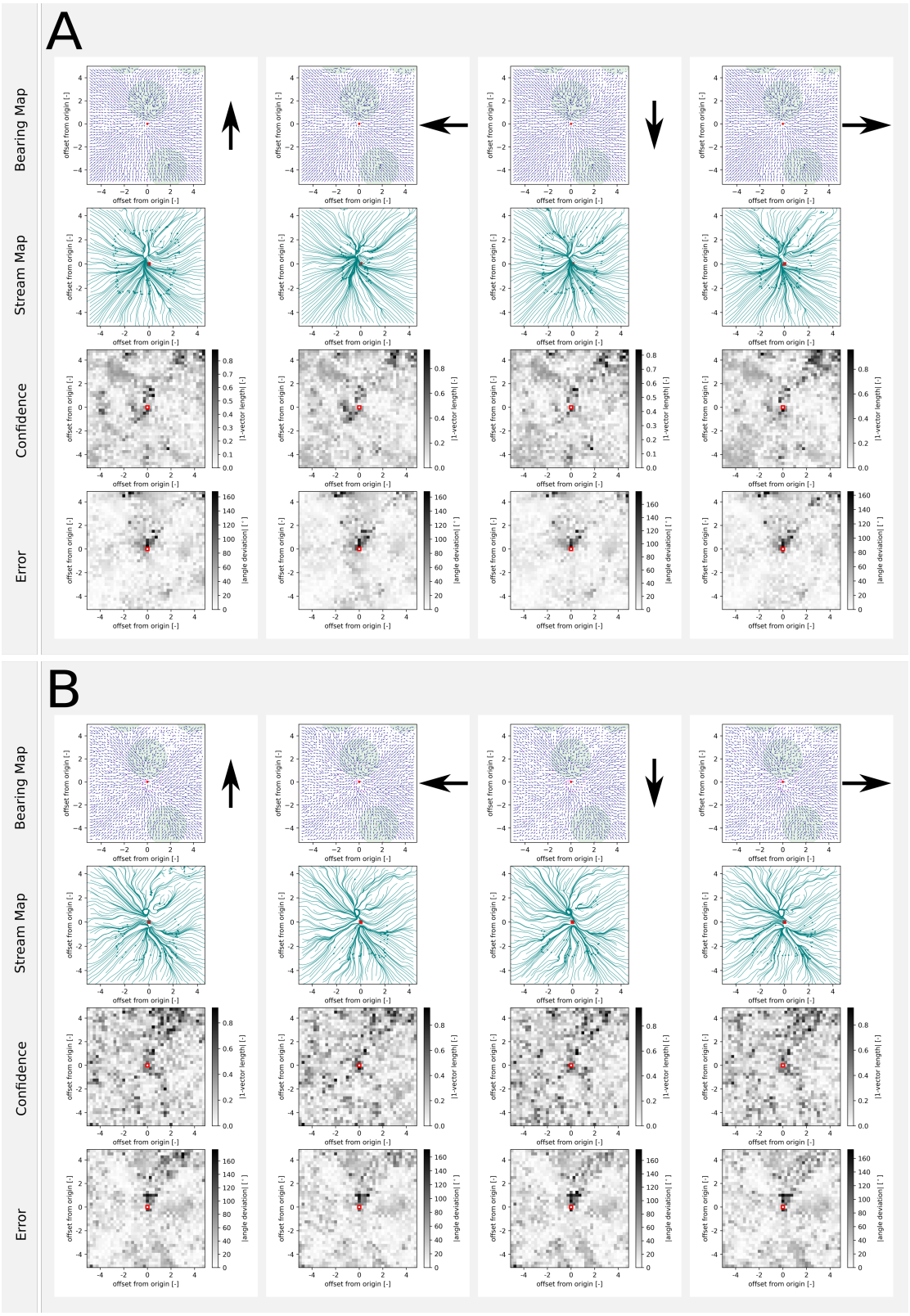


Fig. 5. Generalization performance for two acquisition trajectories: (A) a regular grid and (B) an Archimedean spiral. Each pane includes a bearing map, stream plot, length deviation, and angle deviation, for a subset of gaze angles. Trees are illustrated as green markers with foliage shaded translucent green. Training locations are denoted by translucent magenta points. The nest location is represented as a red point or square. Gaze angles are depicted as black arrows, stream directions as teal arrows, and home vectors as blue arrows.

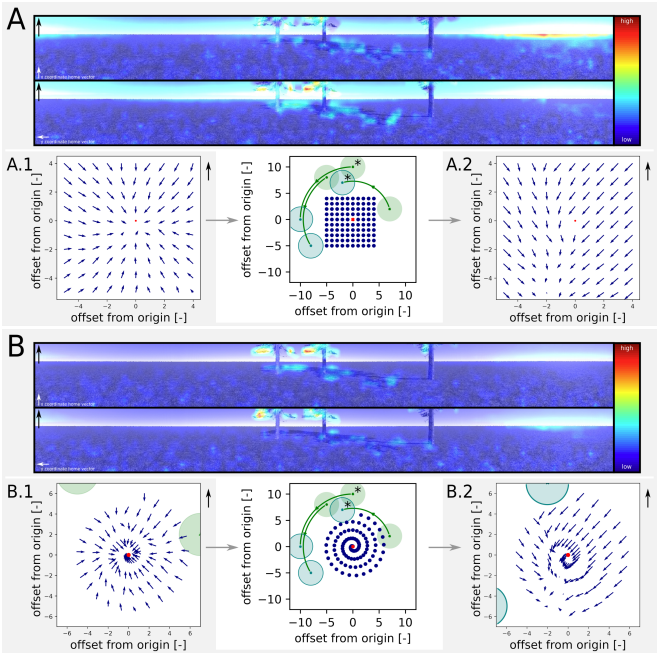


Fig. 6. Analysis of the highest gradients of the CNN output with respect to the second convolutional layer activations per home vector coordinate and bearing maps for two acquisition trajectories: (A) a regular grid and (B) an Archimedean spiral. Each pane comprises an image of a landmark array overlaid with a heatmap and two bearing maps: (A.1/B.1) training performance on original landmark array (green) and (A.2/B.2) evaluation performance, with original training, on rotated landmark array (teal). The original landmark array is rotated counterclockwise $90°$, as indicated. Trees are depicted as green or teal markers with foliage shaded translucent green or teal. Dominant landmarks are marked with an asterisk. Training locations are blue points. The nest location is presented as either a red point or cross. Gaze angles are indicated by black arrows, landmark rotations by green arrows, home vector coordinate directions by white arrows, transitions to rotated landmark performance by grey arrows, and home vectors by blue arrows. The analysis is conducted according to the Grad-CAM method [22].

trees, the CNN exhibited more erratic learning behavior. Plants and bushes tend to blend into the terrain, posing challenges in distinguishing them from the background. This phenomenon might be attributed, in part, to the utilization of low-resolution grayscale input images. Note that the horizon is not entirely uniform, with some protruding pixels discernible against the background. This discrepancy arises due to the implementation of the straightforward Cartesian-to-polar rectification method.

Beyond the assessment of landmarks, our investigation extends to the network's capacity to retain knowledge of the nest location amid evolving surroundings. The presented figure further includes training performance depicted as bearing maps (Fig. 6 A.1/B.1), illustrating instances where the network is trained and evaluated along the same respective acquisition trajectory. Subsequently, an alteration was introduced by rotating the trees counterclockwise (CCW) by $90°$, without subjecting the network to retraining. Following this adjustment, evaluation (Fig. 6 A.2/B.2) led the CNN to infer a downward and leftward shift in the nest location—an unexpected outcome. If our homing approach was purely image-based, the nest would be inferred in the expected location compared to the shifted landmark array. We implicitly encode the gaze angle of the subject, possibly leading to an incorrect generalization of the network to use landmarks in the middle of its visual field as a reference point.

To examine the impact of noise on the algorithm, the generalization performance analysis was performed for 360 gaze angles on all 1600 locations. The mean and standard deviations of the error associated with the two acquisition trajectories are presented in Table I. To demonstrate the robustness of the visual homing approach to unanticipated disturbances, the training labels have been perturbed with $\mathcal{N}(0,\ 100)$ [°] Gaussian noise. One can observe that the generalization performance of the perturbed network is comparable to the unperturbed version.

E. Offline Validation

All the experiments conducted so far have been carried out in a simulated environment. While the performance exhibited successful homing behavior with simulated images, it is imperative to acknowledge that such success does not necessarily ensure comparable outcomes in real-world situations. To bridge this gap, the offline experiment was replicated using a physical 10×10 meter regular acquisition trajectory.

For real-world validation, two distinct forest locations were selected, each characterized by differing features. The first location displayed minimal vegetation within the sampled grid and benefited from substantial sunlight, which effectively illuminated the surroundings. In contrast, the second location presented a scenario that exhibited deeper shadows, due to reduced sunlight penetration of the denser tree canopy. Furthermore, this site included a higher density of vegetation throughout the sample area.

Both data acquisitions took place on two sunny summer days interspersed with small rain showers. The sampling process at the first location commenced at 14:19; however, due to incomplete data, the final 20 samples were retaken the subsequent day at 17:50. For the second location, the sampling began at 11:08 on the second day.

The experimental set-up encompassed the following components: (1) a Oneplus One smartphone, (2) a Kogeto Dot lens, (3) a tripod equipped with an integrated leveler, (4) marker sticks, and (5) a weight attached to a string. The acquisition height was set to $\sim$1.44 meter, in contrast to the $\sim$1.89 meter employed in the simulation environment. The Kogeto Dot lens, for which no specifications are available, offers a cost-effective approach for obtaining omnidirectional data. However, it is likely that this lens does not conform to the assumption of polar-to-Cartesian rectification.

The real world images include various visual artefacts that are not present in their simulated counterparts. Lens flare as well as inconsistent illumination make the real pictures unclear. Fig. 7 presents the rectified samples. It can be observed that due to the uneven terrain, the two-dimensional assumption was not adhered to and the rectification returned a sinusoidal horizon. Also, the texture in location 2 (Fig. 7 B) is completely lost since the grid was located in the shadows of tree foliage.

TABLE I
EVALUATION ERROR ON 575,640 SAMPLES

Trained	**Without label noise**		**With label noise**	
Trajectory	μ^{m} [°]	σ^{m} [°]	μ^{m} [°]	σ^{m} [°]
Grid	16.8176	18.9834	17.6635	18.4317
Spiral	23.3427	21.7665	22.7724	21.9405

$^{\mathrm{m}}\mu$ is mean, σ is standard deviation.

Next to the rectified image per location, an image of the test area is depicted as well as some test equipment.

As a cost-effective and flexible acquisition method, the sample data were manually captured using a tripod. It allowed for the evaluation of the CNN's robustness to adapt to changes beyond the confines of simulation, enabling validation on real-world data from forest environments. Despite the fact that the quality of the sample images falls short of that of the simulated ones, and certain assumptions were not upheld, we can conclude that our approach transfers very well to real world data.

The inferred home vectors following training at location 1 (Fig. 7 A) displayed a comparable error as observed during simulation. In the case of location 2, the performance slightly decreased due to the presence of more vegetation and shadows falling from the dense tree canopy on the forest floor. Nonetheless, the achieved results remained remarkable, particularly when considering the demanding conditions the system had to contend with. It can be concluded that the network's confidence diminishes with decreasing image quality and texture. However, the CNN still manages to achieve an MSE during training of 2866.1219 m^2 for location 1 and 7239.1858 m^2 for location 2 (batch size 1 for $35,640$ training samples). This is a similar performance compared to the simulated grid case if all data were collected under favorable circumstances, i.e. for location 1.

F. Online Simulation

The culmination of all preceding experiments is manifested in an online simulation experiment. The data acquisition and training of the CNN is performed exactly the same as in the offline simulation. The core difference between the two simulations is the utilization of the network output as a control input in the online experiment. Specifically, omnidirectional images are acquired at 100 locations along an acquisition trajectory, each image is rectified at 1° intervals, and the CNN is trained using the resultant dataset encompassing $35,640$ samples in a random order within a single epoch. Subsequently, the test subject initiates an arbitrary outbound trajectory, emulating a PI-based foraging journey. Upon its return to the area around the nest, the subject's integrated home vector zeros out within the catchment area of the bearing inferring network. In sequence, new image data are acquired and assessed at 0.25 unit intervals in the evaluate-to-home phase. After each evaluation, the simulated bee advances a step in alignment with the inferred home vector until a step concludes within a radius of 0.25 around the nest position.

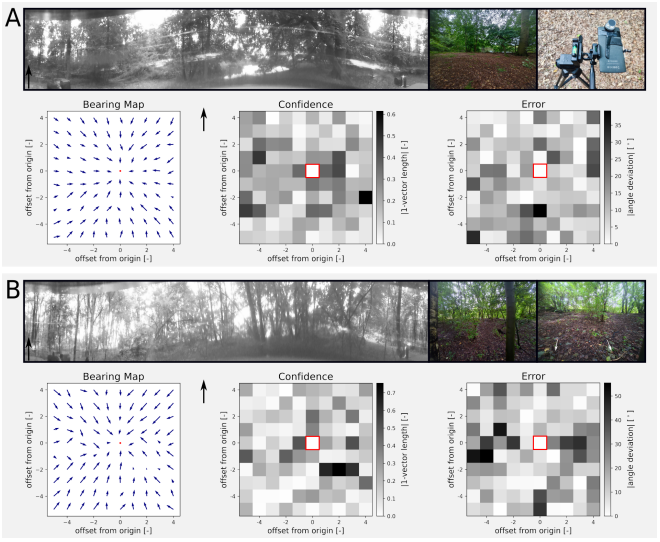


Fig. 7. Training performance in two real forest environments: (A) open and bright and (B) covered and dark. The provided images encompass sample images, the experimental area, the equipment used, and bearing maps. A OnePlus One smartphone equipped with a 13MP Sony Exmor IMX 214 (f/2.0 aperture) sensor and an additional Kogeto Dot omnidirectional lens was utilized. Gaze angles are indicated by black arrows, sample location markers by white arrows, and home vectors by blue arrows.

Consequently, the metric employed to ascertain the relative success of each acquisition trajectory is the count of steps needed to reach the goal condition. Note that the test subject is allowed to continue for a few steps after reaching the goal condition in order to visualize the convergence point of its trajectory around the hive.

The mean resultant inbound trajectory, coupled with the network confidence per step, is depicted in Fig. 8. Opaque lines and points delineate three homing attempts, each re-trained and initiated from an edge of an acquisition trajectory. An immediately conspicuous feature is the convergence point, previously discussed in section IV-C, positioned above the nest location for the grid acquisition trajectory (Fig. 8 A). The CNN trained on a spiral trajectory (Fig. 8 B) significantly enhances the consistency of homing compared to the grid trajectory. In fact, the grid variant does not attain the goal condition even once. In contrast, the spiral variant achieves the goal in 10 out of 12 attempts, with an average step count of 33.5. This outcome should be interpreted in the context of our decision to exclude nest data during the training process. Nevertheless, a denser sample distribution around the nest area can enhance convergence to the nest location. The findings underscore the profound impact that modifying the acquisition trajectory can have on homing performance. Therefore, fine-tuning of the acquisition trajectory can yield a more precise match of the convergence point with the nest location.

For every attempt per starting point, the network is retrained with random weight initialization and a random order of the sample dataset (batch size 1 for $35,640$ training samples). Interestingly, most of the evaluate-to-home trajectories in Fig. 8

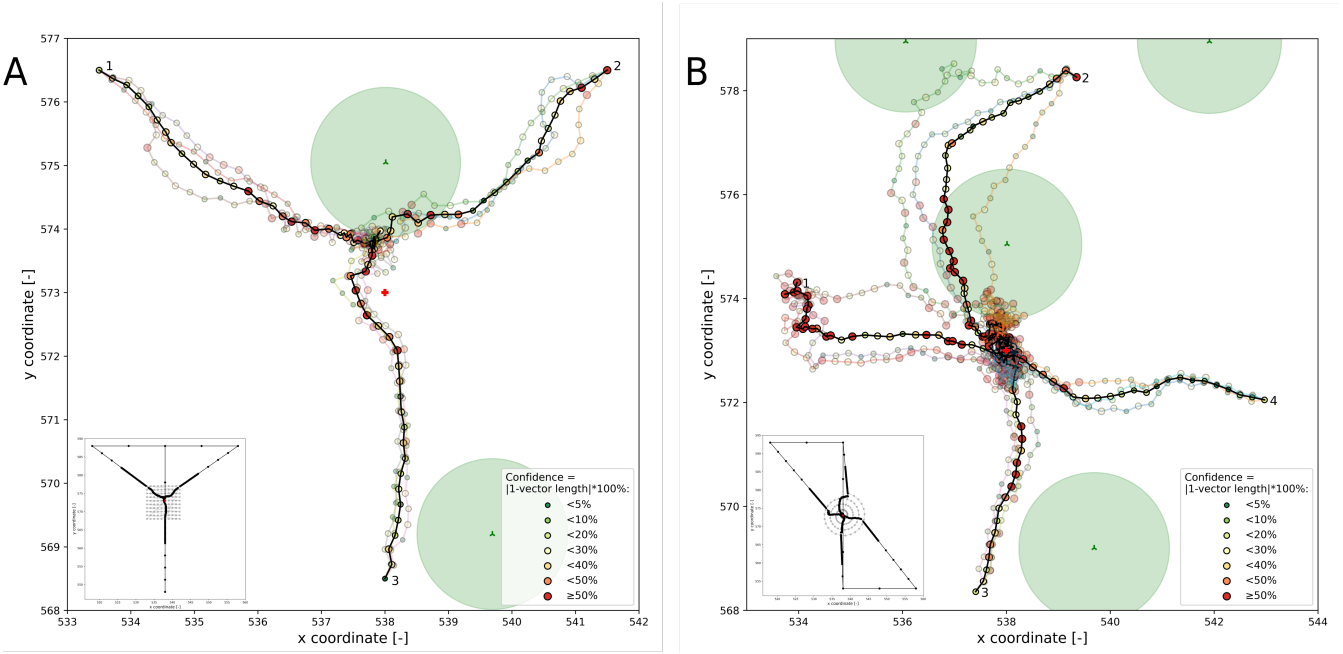


Fig. 8. Evaluate-to-home trajectories starting close to an edge of the respective acquisition trajectory: (A) a regular grid and (B) an Archimedean spiral. Trees are depicted as green markers with foliage shaded translucent green. The nest is marked with a red cross at location $[538, 573]^T$. Trajectory numbers are given adjacent to their starting points. Smaller confidence markers indicate higher confidence value. An inset, situated in the lower left corner of each pane, shows an example of all online flight phases per evaluate-to-home trajectory, where the acquisition trajectory is marked with opaque dots. The dimensions of the acquisition trajectory roughly match the overall figure size.

B originating from starting point 2 appear to demonstrate a certain tendency to inherently avoid objects, circumventing tree foliage to evade the occlusion stemming from flying through it. In the attempt that the subject did not reach the nest, it got stuck in areas with abundant tree foliage (green shaded areas).

V. Discussion

The size of the algorithm's catchment area is corresponds to the area encompassing the learning flight of the robot. It is tempting to increase this area by stretching out the learning flight to facilitate homing in a much larger area. However, this expansion can reach a point where the network's generalization performance or size becomes undesirable. A straightforward method to increase the effective area of the network is to repeat local training several times in sequence for different waypoints along a route. Ensuring consistent convergence within the catchment area of the respective goal waypoint would be vital in this case. The probability of successful homing would diminish significantly outside the designated training regions.

We aspire to conduct further research on Hymenopteran navigation, thereby assisting zoologists and biologist to explore its intricate mechanisms. Future efforts would involve extra tuning of the sample acquisition trajectories and other network hyperparameters. Verification of the offline experiment led to generalization performance plots illustrating the impact of training on alternate data acquisition trajectories. Although the utilized spiral trajectory already includes a dense sample distribution above the nest, the introduction of more uniform concentric circles could potentially enhance precise nest convergence even further.

Moreover, we envision the potential integration of supplemental flight time training labels as a distance cue. Real bees convey distance information as well as directional information to their peers through waggle dances [1]. By including flight time information as labels during CNN training, the goal condition could be directly implied by the network. With no distance information, an extra nest detection algorithm should be present to facilitate goal condition recognition, i.e. finding the exact nest location. Thus, enhancing the network's ability to detect some form of distance to the nest would be advantageous.

Additionally, we plan to address the sequential training limitation. Several mitigation strategies are available to address catastrophic forgetting in sequential learning methods, as presented by Kirkpatrick et al. [23].

Finally, the revalidation of the algorithm could be performed on a MAV whenever the network's size is tailored for embedded deployment. This step would conclude this series of experiments and would establish the direct learning of home vector directions as a preferred method for visual homing in robotics.

VI. Conclusion

We proposed an insect-inspired visual homing approach that relies exclusively on inferred home vector bearings from single

rectified catadioptric images. The main hypothesis posits that bees retain directional cues to the nest following their initial learning flight. Subsequently, we have demonstrated the feasibility of training an artificial neural network in either a simulated or actual forest environment to effectively execute the homing procedure in a bee-like manner. Moreover, this has been realized by utilizing comparable resources as those employed by the actual bee, i.e. visual and odometric senses.

An extensive performance analysis has been conducted on two acquisition trajectories centered around the nest location: a regular grid and an Archimedean spiral. The network's performance can be summarized by its inference of the home vector direction on newly experienced data. The associated absolute errors are distributed according to $\mathcal{N}(16.8176,\ 360.3695)$ [°] and $\mathcal{N}(23.3427,\ 473.7805)$ for grid and spiral variants respectively.

One main limitation of the algorithm in its current state has been identified: All training samples have to be retained in memory until training is complete. This is due to the 'catastrophic forgetting' property, which plagues sequential learning in general. Despite this, we believe the algorithm could be employed quickly and auspiciously on autonomous vehicles in arbitrary environments. This, in turn, would facilitate these vehicles to return to their starting point or charging port without any external navigational aids.

Future work will aim to address the sequential learning limitation as well as fine-tune homing performance in order to efficiently deploy direct learning on a variety of robotic platforms.

REFERENCES

[1] Chittka, L., *The Mind of a Bee*, 1st ed., Princeton University Press, Princeton, NJ, USA, Jul. 2022.

[2] de Croon, G. C. H. E., Dupeyroux, J. J. G., Fuller, S. B., and Marshall, J. A. R., "Insect-inspired AI for autonomous robots," *Science Robotics*, vol. 7, no. 67, 2022.

[3] Cartwright, B. A., and Collett, T. S., "Landmark learning in bees," *Journal of Comparative Physiology*, vol. 151, 1983, pp. 521–543.

[4] Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., and Wehner, R., "A mobile robot employing insect strategies for navigation," *Robotics and Autonomous Systems*, vol. 30, 2000, pp. 39–64.

[5] Hafner, V., and Möller, R., "Learning of visual navigation strategies," *Proceedings of the European Workshop on Learning Robots*, vol. 9, 2001, pp. 47–56.

[6] Baddeley, B., Graham, P., Husbands, P., and Philippides, A., "A model of ant route navigation driven by scene familiarity," *PLoS Computational Biology*, vol. 8, 2012, pp. 1–16.

[7] Möel, F. L., and Wystrach, A., "Opponent processes in visual memories: A model of attraction and repulsion in navigating insects' mushroom bodies," *PLoS Computational Biology*, vol. 16, no. 2, 2020, pp. 1–24.

[8] Hafner, V. V., "Adaptive homing – Robotic exploration tours," *Adaptive Behavior*, bol. 9, no. 3-4, 2001, pp. 131–141.

[9] Haferlach, T., Wessnitzer, J., Mangan, M., and Webb, B., "Evolving a neural model of insect path integration," *Adaptive Behavior*, vol. 15, no. 3, 2007, pp. 273–287.

[10] Srinivasan, M. V., Zhang, S. W., Lehrer, M., and Collett, T. S., "Honeybee navigation en route to the goal: Visual flight control and odometry," *The Journal of Experimental Biology*, vol. 199, no. 1, 1996, pp. 237–244.

[11] von Frisch, K., *The Dance Language and Orientation of Bees*, 1st ed., Harvard University Press, Cambridge, MA., USA, 1967.

[12] Capaldi, E. A., Smith, A. D., Osborne, J. L., Fahrbach, S. E., Farris, S. M., Reynolds, D. R., Edwards, A. S., Martin, A., Robinson, G. E., Poppy, G. M., and Riley, J. R., "Ontogeny and orientation flight in the honeybee revealed by harmonic radar," *Nature*, vol. 403, 2000, pp. 537–540.

[13] Zeil, J., "Visual homing: an insect perspective," *Current Opinion in Neurobiology*, vol. 22, no. 2, 2012, pp. 285–293.

[14] Lehrer, M., and Bianco, G., "The turn-back-and-look behaviour: bee versus robot," *Biological Cybernetics*, vol. 83, 2000, pp. 211–229.

[15] Lehrer, M., "Why do bees turn back and look," *Journal of Comparative Physiology*, vol. 172, 1993, pp. 549–563.

[16] Zeil, J., "Orientation flights of solitary wasps (Cerceris; Sphecidae; Hymenoptera)," *Journal of Comparative Physiology*, vol. 172, 1993, pp. 189–205.

[17] Stürzl, W., and Mallot, H. A., "Efficient visual homing based on Fourier transformed panoramic images," *Robotics and Autonomous Systems*, vol. 54, no. 4, 2006, pp. 300–313.

[18] van Dalen, G. J. J., McGuire, K., and de Croon, G. C. H. E., "Visual homing for micro aerial vehicles using scene familiarity," *Unmanned Systems*, vol. 6, no. 2, 2018, pp. 119-130.

[19] Berenguel-Baeta, B., Bernudez-Cameo, J., and Guerrero, J. J., "OmniSCV: an omnidirectional synthetic image generator for computer vision," *Sensors*, vol. 20, no. 7, 2020.

[20] Maurer, R., and Séguinot, V., "What is modelling for? A critical review of the models of path integration," *Journal of Theoretical Biology*, vol. 175, 1995, pp. 457-475.

[21] Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D., "Flightmare: a flexible quadrotor simulator," *Proceedings of the 2020 Conference on Robot Learning*, vol. 155, 2021, pp. 1147-1157.

[22] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra D., "Grad-CAM: visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336-359.

[23] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521-3526.

APPENDIX

APPENDIX A
SIMULATED REGULAR GRID DATASET FOR 360 GAZE ANGLES



(a) Offset from nest: $(-5, -4)$, Gaze angle: 0°



(b) Offset from nest: $(-5, -3)$, Gaze angle: 0°



(c) Offset from nest: $(-5, -2)$, Gaze angle: 0°



(d) Offset from nest: $(-5, -1)$, Gaze angle: 0°



(e) Offset from nest: $(-5, 0)$, Gaze angle: 0°



(f) Offset from nest: $(-5, 1)$, Gaze angle: 0°



(g) Offset from nest: $(-5, 2)$, Gaze angle: 0°



(h) Offset from nest: $(-5, 3)$, Gaze angle: 0°



(i) Offset from nest: $(-5, 4)$, Gaze angle: 0°

Fig. 1: Selection of sample locations on the square 10×10 regular **grid** used for training.



(a) Offset from nest: $(-4, -5)$, Gaze angle: 0°



(b) Offset from nest: $(-3, -5)$, Gaze angle: 0°



(c) Offset from nest: $(-2, -5)$, Gaze angle: 0°



(d) Offset from nest: $(-1, -5)$, Gaze angle: 0°



(e) Offset from nest: $(0, -5)$, Gaze angle: 0°



(f) Offset from nest: $(1, -5)$, Gaze angle: 0°



(g) Offset from nest: $(2, -5)$, Gaze angle: 0°

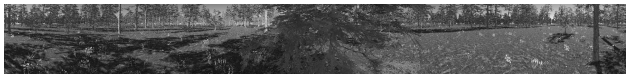


(h) Offset from nest: $(3, -5)$, Gaze angle: 0°

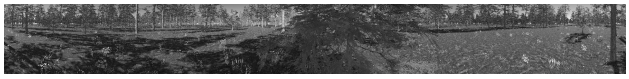


(i) Offset from nest: $(4, -5)$, Gaze angle: 0°

Fig. 2: Selection of sample locations on the square 10×10 regular **grid** used for training.



(a) Offset from nest: (0, 0), Gaze angle: 10°



(b) Offset from nest: (0, 0), Gaze angle: 20°



(c) Offset from nest: (0, 0), Gaze angle: 30°



(d) Offset from nest: (0, 0), Gaze angle: 40°



(e) Offset from nest: (0, 0), Gaze angle: 50°



(f) Offset from nest: (0, 0), Gaze angle: 60°



(g) Offset from nest: (0, 0), Gaze angle: 70°



(h) Offset from nest: (0, 0), Gaze angle: 80°



(i) Offset from nest: (0, 0), Gaze angle: 90°



(j) Offset from nest: (0, 0), Gaze angle: 100°



(k) Offset from nest: (0, 0), Gaze angle: 110°



(l) Offset from nest: (0, 0), Gaze angle: 120°



(m) Offset from nest: (0, 0), Gaze angle: 130°



(n) Offset from nest: (0, 0), Gaze angle: 140°



(o) Offset from nest: (0, 0), Gaze angle: 150°



(p) Offset from nest: (0, 0), Gaze angle: 160°



(q) Offset from nest: (0, 0), Gaze angle: 170°



(r) Offset from nest: (0, 0), Gaze angle: 180°



(s) Offset from nest: (0, 0), Gaze angle: 190°



(t) Offset from nest: (0, 0), Gaze angle: 200°



(u) Offset from nest: (0, 0), Gaze angle: 210°



(v) Offset from nest: (0, 0), Gaze angle: 220°



(w) Offset from nest: (0, 0), Gaze angle: 230°



(x) Offset from nest: (0, 0), Gaze angle: 240°



(y) Offset from nest: (0, 0), Gaze angle: 250°



(z) Offset from nest: (0, 0), Gaze angle: 260°

Fig. 3: Selection of sample gaze angles on the square 10×10 regular **grid** used for training.



(a) Offset from nest: (0, 0), Gaze angle: 270°



(b) Offset from nest: (0, 0), Gaze angle: 280°



(c) Offset from nest: (0, 0), Gaze angle: 290°



(d) Offset from nest: (0, 0), Gaze angle: 300°



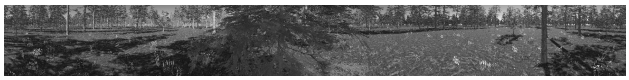
(e) Offset from nest: (0, 0), Gaze angle: 310°



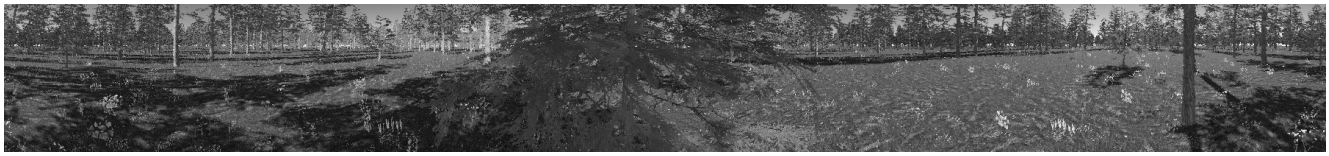
(f) Offset from nest: (0, 0), Gaze angle: 320°



(g) Offset from nest: (0, 0), Gaze angle: 330°



(h) Offset from nest: (0, 0), Gaze angle: 340°



(i) Offset from nest: (0, 0), Gaze angle: 350°

Fig. 4: Selection of sample gaze angles on the square 10×10 regular **grid** used for training.

APPENDIX B
SIMULATED SPIRAL DATASET FOR 360 GAZE ANGLES



(a) Offset from nest: $(-0.01562, 0.0609)$, Gaze angle: 0°



(b) Offset from nest: $(-0.0605, 0.1101)$, Gaze angle: 0°



(c) Offset from nest: $(-0.1290, 0.1374)$, Gaze angle: 0°



(d) Offset from nest: $(-0.2122, 0.1347)$, Gaze angle: 0°



(e) Offset from nest: $(-0.2988, 0.0971)$, Gaze angle: 0°



(f) Offset from nest: $(-0.3762, 0.0237)$, Gaze angle: 0°



(g) Offset from nest: $(-0.432, -0.0824)$, Gaze angle: 0°



(h) Offset from nest: $(-0.4548, -0.214)$, Gaze angle: 0°



(i) Offset from nest: $(-0.4357, -0.3605)$, Gaze angle: 0°



(j) Offset from nest: $(-0.3693, -0.5083)$, Gaze angle: 0°



(k) Offset from nest: $(-0.2544, -0.6426)$, Gaze angle: 0°



(l) Offset from nest: $(-0.0945, -0.748)$, Gaze angle: 0°



(m) Offset from nest: $(0.1024, -0.8104)$, Gaze angle: 0°



(n) Offset from nest: $(0.3238, -0.8179)$, Gaze angle: 0°



(o) Offset from nest: $(0.554, -0.7625)$, Gaze angle: 0°



(p) Offset from nest: $(0.7746, -0.6408)$, Gaze angle: 0°



(q) Offset from nest: $(0.9665, -0.4548)$, Gaze angle: 0°



(r) Offset from nest: $(1.1109, -0.2119)$, Gaze angle: 0°



(s) Offset from nest: $(1.1914, 0.075)$, Gaze angle: 0°



(t) Offset from nest: $(1.1951, 0.3883)$, Gaze angle: 0°

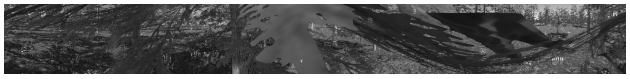


(u) Offset from nest: $(1.1141, 0.707)$, Gaze angle: 0°



(v) Offset from nest: $(0.9462, 1.0077)$, Gaze angle: 0°

Fig. 5: Selection of sample locations on the Archimedean **spiral** used for training.



(a) Offset from nest: (0.6962, 1.2664), Gaze angle: 0°

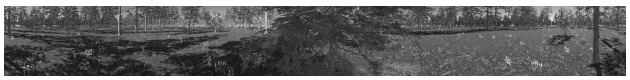


(b) Offset from nest: (0.375, 1.4606), Gaze angle: 0°



(c) Offset from nest: (0, 1.5708), Gaze angle: 0°

Fig. 6: Selection of sample locations on the Archimedean **spiral** used for training.



(a) Offset from nest: (0, 0), Gaze angle: 10°



(b) Offset from nest: (0, 0), Gaze angle: 20°



(c) Offset from nest: (0, 0), Gaze angle: 30°



(d) Offset from nest: (0, 0), Gaze angle: 40°



(e) Offset from nest: (0, 0), Gaze angle: 50°



(f) Offset from nest: (0, 0), Gaze angle: 60°



(g) Offset from nest: (0, 0), Gaze angle: 70°



(h) Offset from nest: (0, 0), Gaze angle: 80°



(i) Offset from nest: (0, 0), Gaze angle: 90°



(j) Offset from nest: (0, 0), Gaze angle: 100°



(k) Offset from nest: (0, 0), Gaze angle: 110°



(l) Offset from nest: (0, 0), Gaze angle: 120°



(m) Offset from nest: (0, 0), Gaze angle: 130°



(n) Offset from nest: (0, 0), Gaze angle: 140°



(o) Offset from nest: (0, 0), Gaze angle: 150°



(p) Offset from nest: (0, 0), Gaze angle: 160°



(q) Offset from nest: (0, 0), Gaze angle: 170°



(r) Offset from nest: (0, 0), Gaze angle: 180°

Fig. 7: Selection of sample gaze angles on the Archimedean **spiral** used for training.



(a) Offset from nest: (0, 0), Gaze angle: 190°



(b) Offset from nest: (0, 0), Gaze angle: 200°



(c) Offset from nest: (0, 0), Gaze angle: 210°



(d) Offset from nest: (0, 0), Gaze angle: 220°



(e) Offset from nest: (0, 0), Gaze angle: 230°



(f) Offset from nest: (0, 0), Gaze angle: 240°



(g) Offset from nest: (0, 0), Gaze angle: 250°



(h) Offset from nest: (0, 0), Gaze angle: 260°



(i) Offset from nest: (0, 0), Gaze angle: 270°



(j) Offset from nest: (0, 0), Gaze angle: 280°



(k) Offset from nest: (0, 0), Gaze angle: 290°



(l) Offset from nest: (0, 0), Gaze angle: 300°



(m) Offset from nest: (0, 0), Gaze angle: 310°



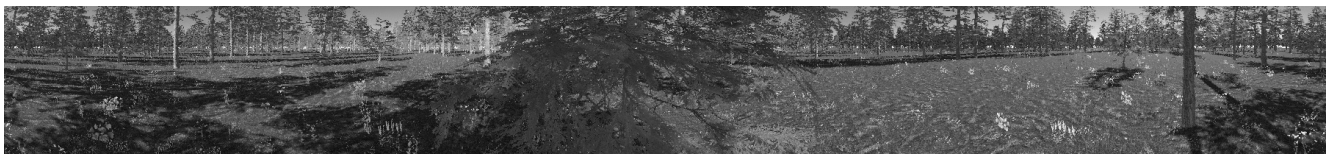
(n) Offset from nest: (0, 0), Gaze angle: 320°



(o) Offset from nest: (0, 0), Gaze angle: 330°



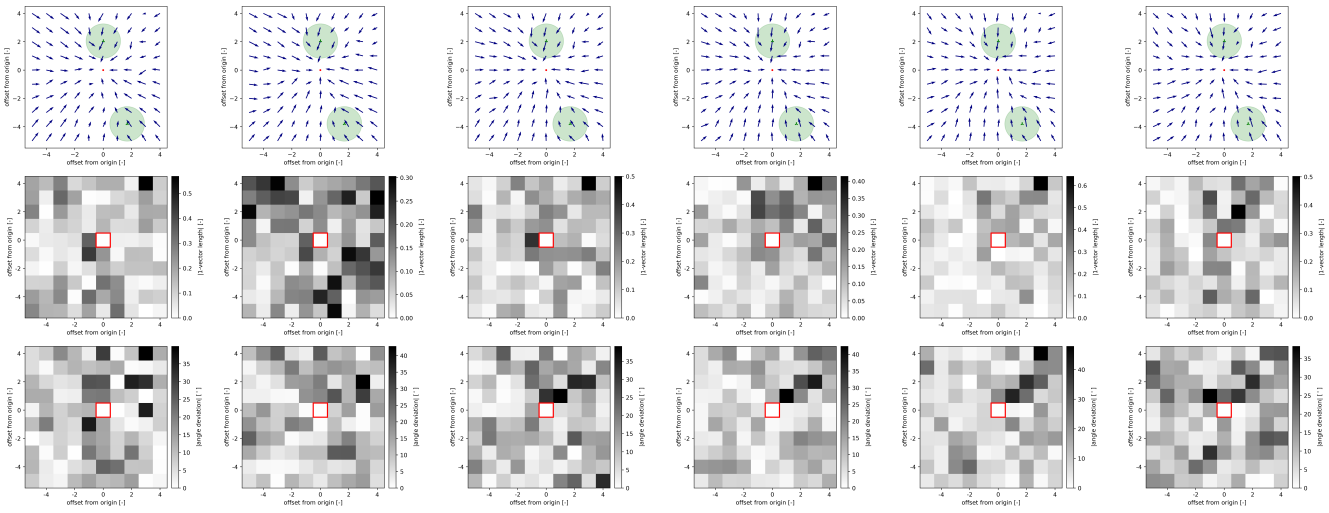
(p) Offset from nest: (0, 0), Gaze angle: 340°



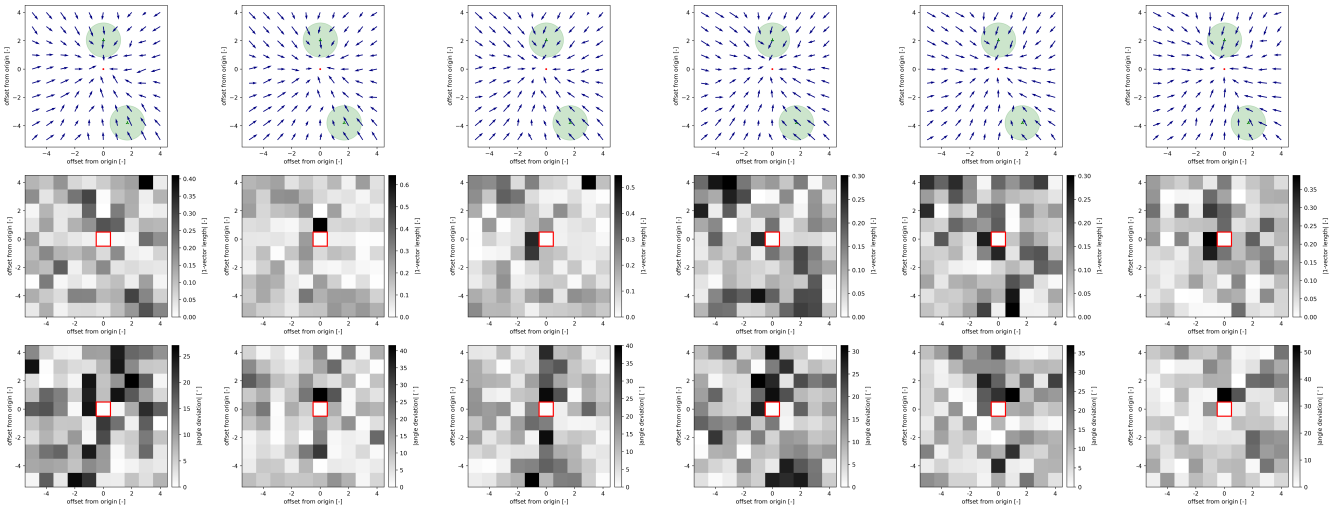
(q) Offset from nest: (0, 0), Gaze angle: 350°

Fig. 8: Selection of sample gaze angles on the Archimedean **spiral** used for training.

APPENDIX C
SIMULATED TRAINING PERFORMANCE FOR 360 GAZE ANGLES

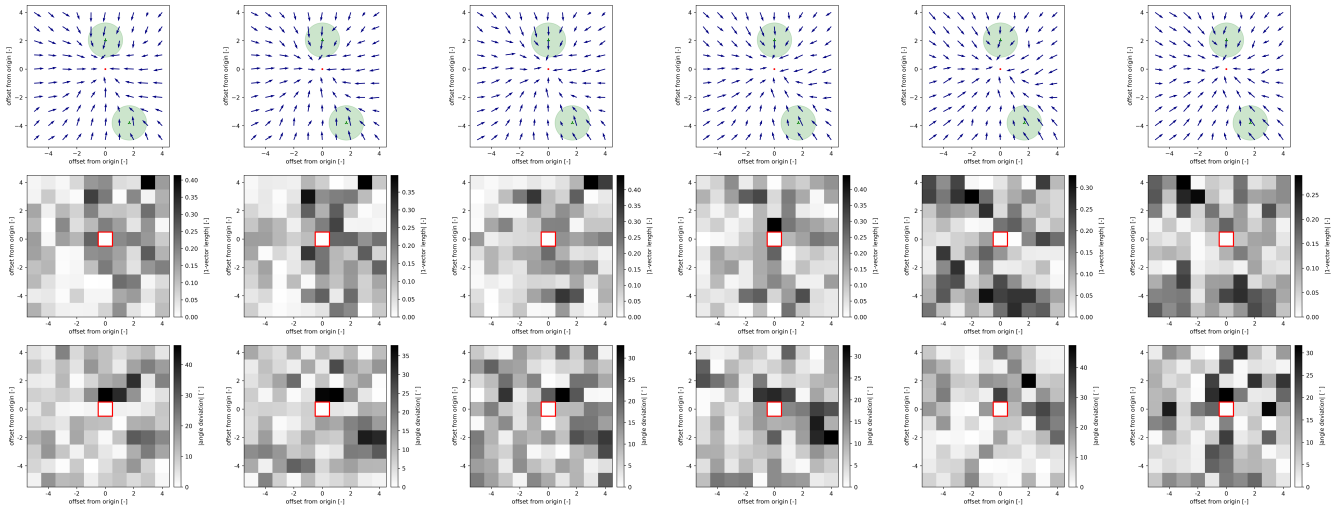


(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40° (e) Gaze angle: 50° (f) Gaze angle: 60°

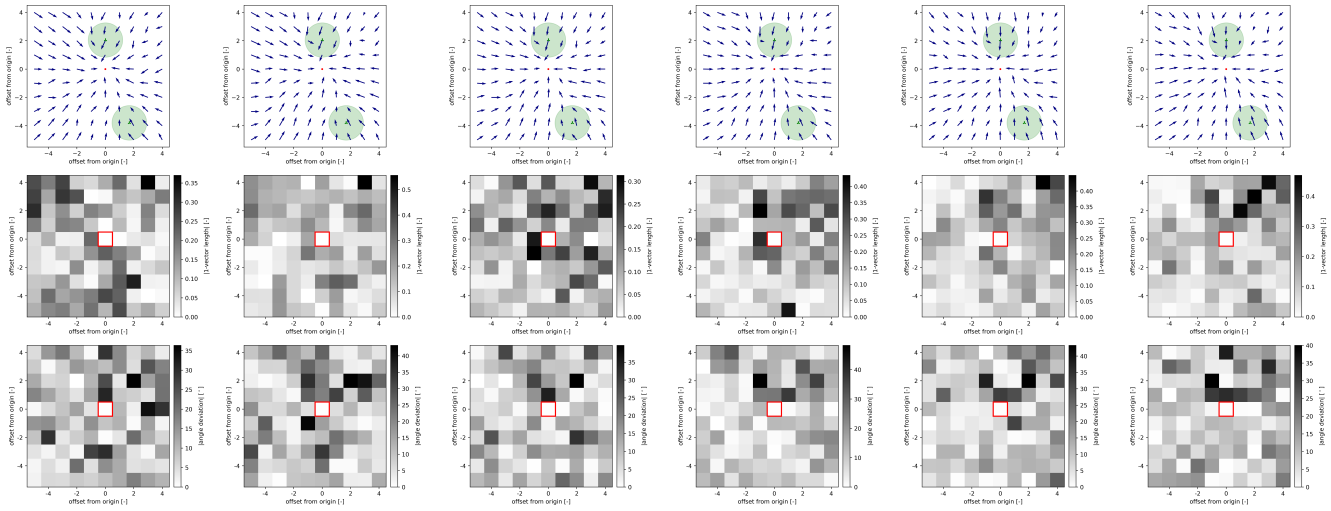


(g) Gaze angle: 70° (h) Gaze angle: 80° (i) Gaze angle: 90° (j) Gaze angle: 100° (k) Gaze angle: 110° (l) Gaze angle: 120°

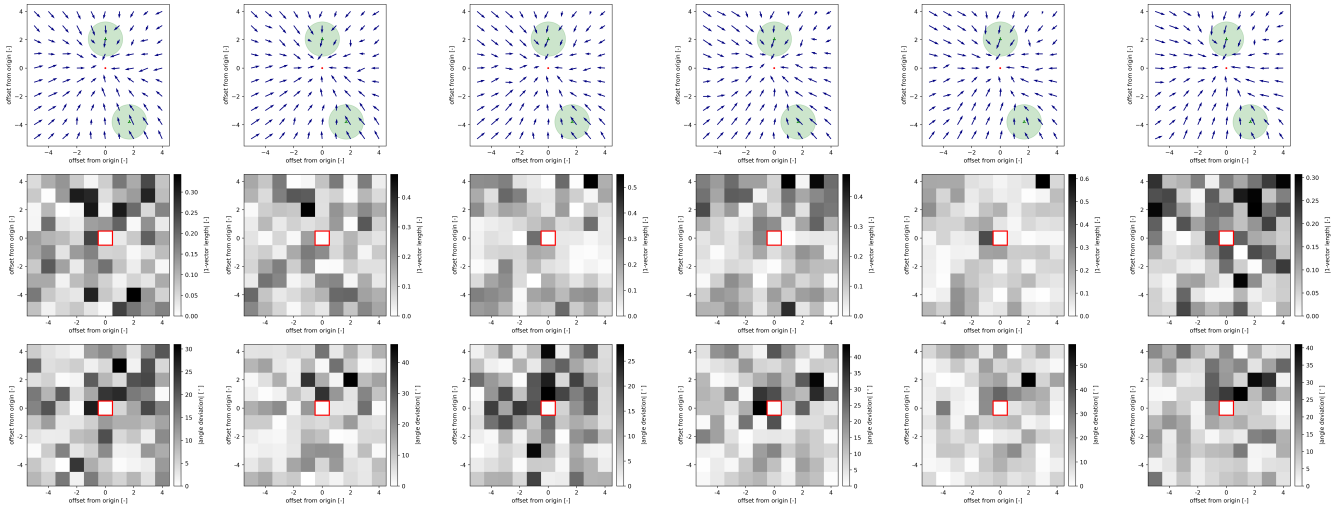
Fig. 9: Performance of network trained on **grid** acquisition trajectory. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 130° (b) Gaze angle: 140° (c) Gaze angle: 150° (d) Gaze angle: 160° (e) Gaze angle: 170° (f) Gaze angle: 180°

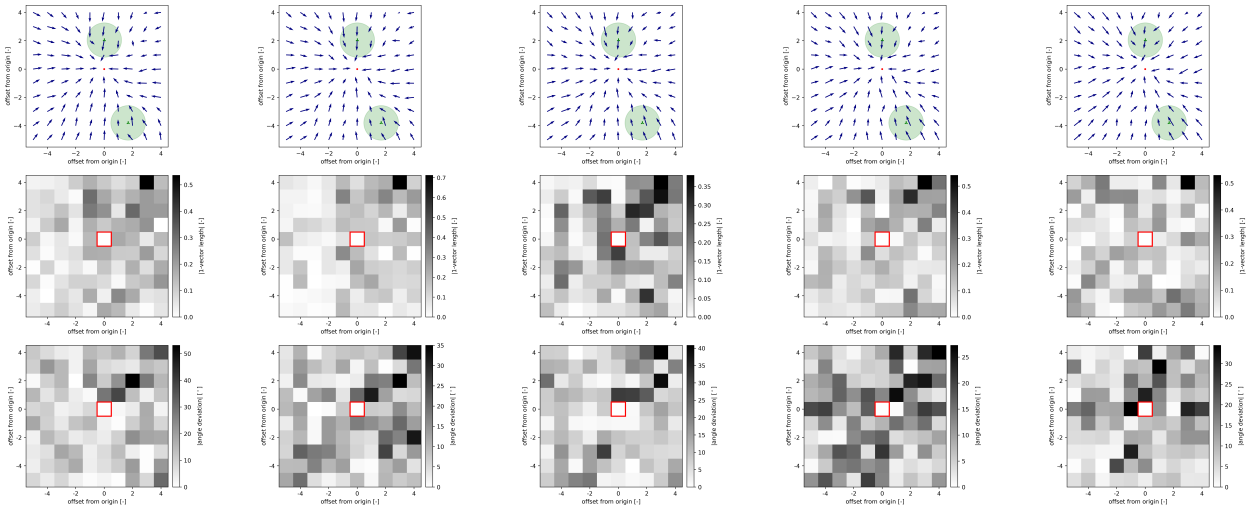


(g) Gaze angle: 190° (h) Gaze angle: 200° (i) Gaze angle: 210° (j) Gaze angle: 220° (k) Gaze angle: 230° (l) Gaze angle: 240°



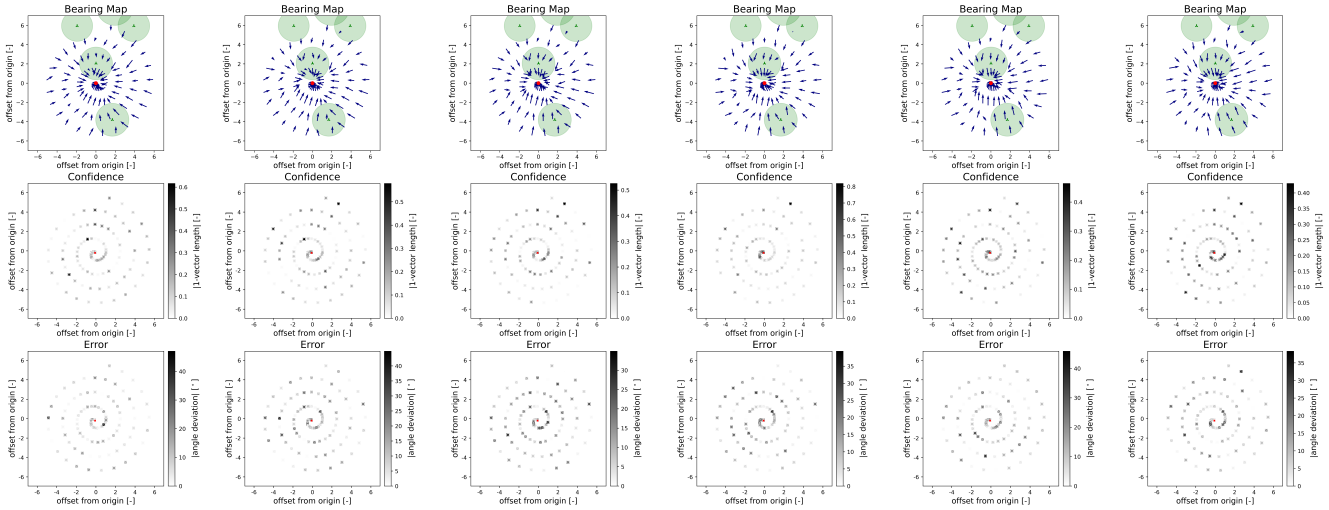
(m) Gaze angle: 250° (n) Gaze angle: 260° (o) Gaze angle: 270° (p) Gaze angle: 280° (q) Gaze angle: 290° (r) Gaze angle: 300°

Fig. 10: Performance of network trained on **grid** acquisition trajectory. Gaze angle is relative to vector pointing North/upwards.

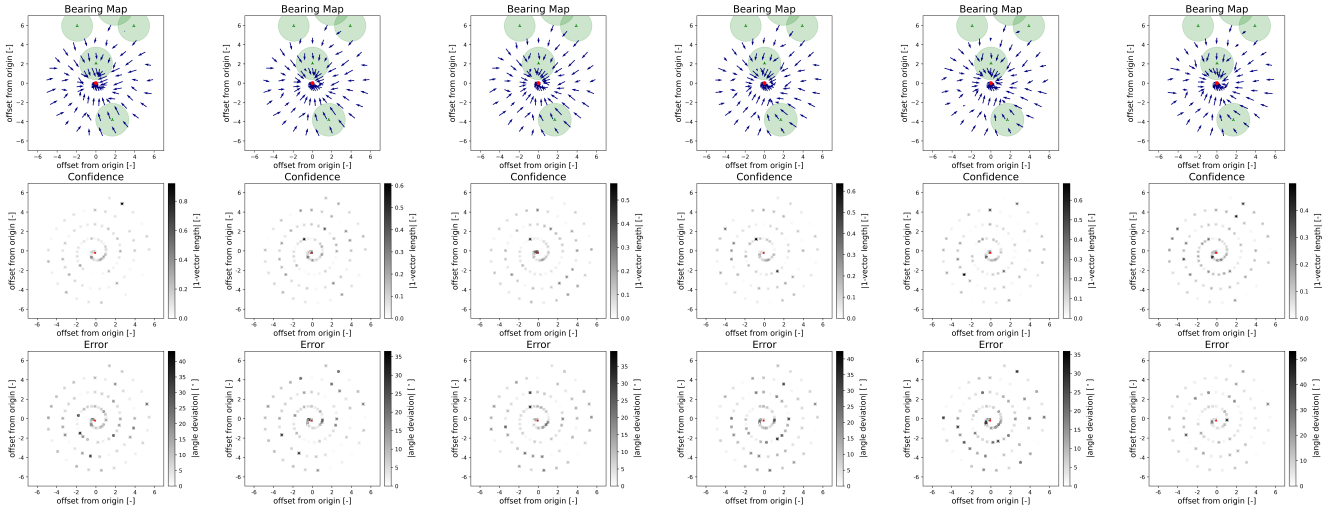


(a) Gaze angle: 310° (b) Gaze angle: 320° (c) Gaze angle: 330° (d) Gaze angle: 340° (e) Gaze angle: 350°

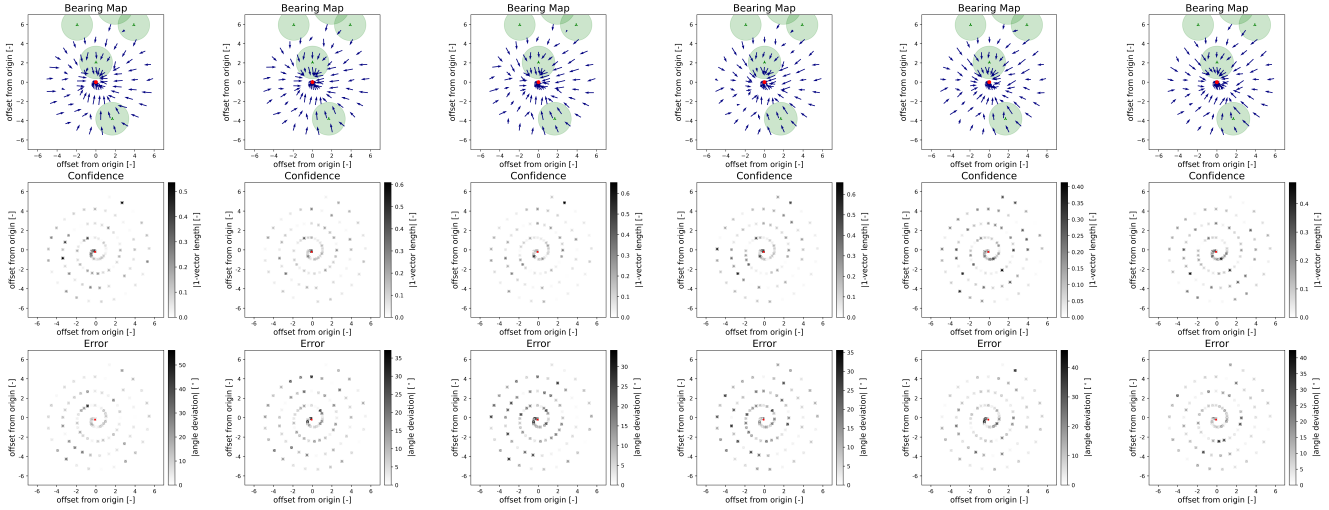
Fig. 11: Performance of network trained on **grid** acquisition trajectory. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40° (e) Gaze angle: 50° (f) Gaze angle: 60°

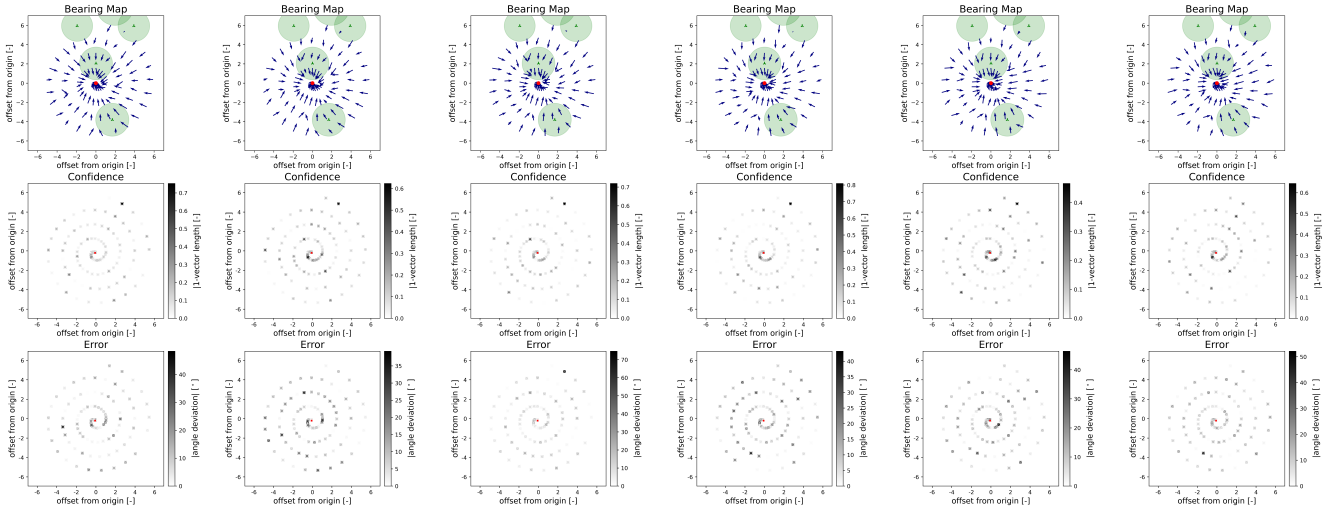


(g) Gaze angle: 70° (h) Gaze angle: 80° (i) Gaze angle: 90° (j) Gaze angle: 100° (k) Gaze angle: 110° (l) Gaze angle: 120°

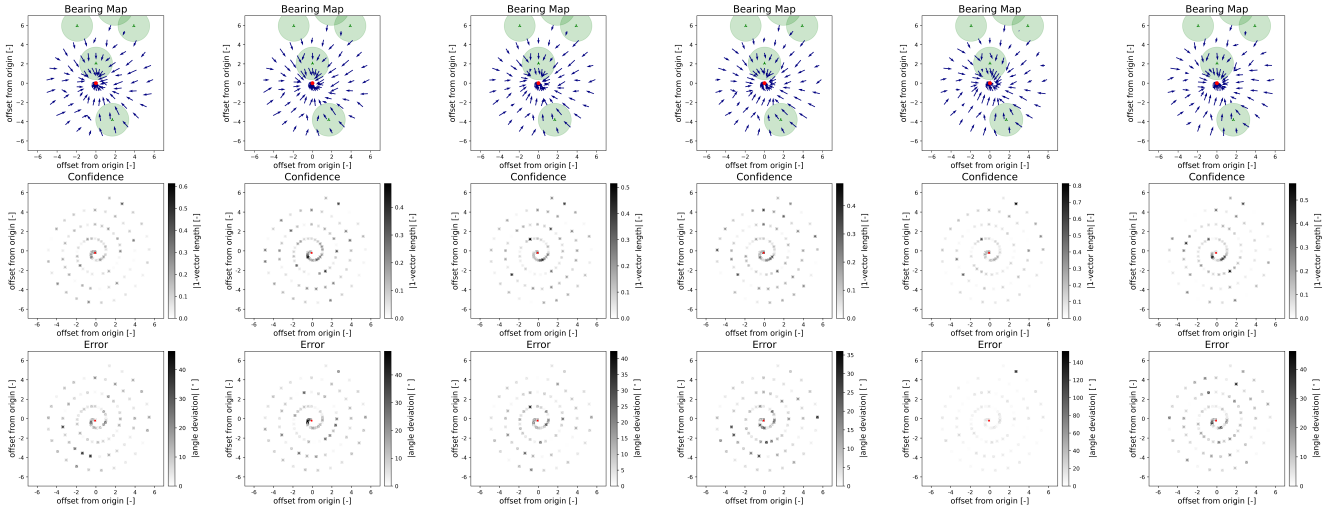


(m) Gaze angle: 130° (n) Gaze angle: 140° (o) Gaze angle: 150° (p) Gaze angle: 160° (q) Gaze angle: 170° (r) Gaze angle: 180°

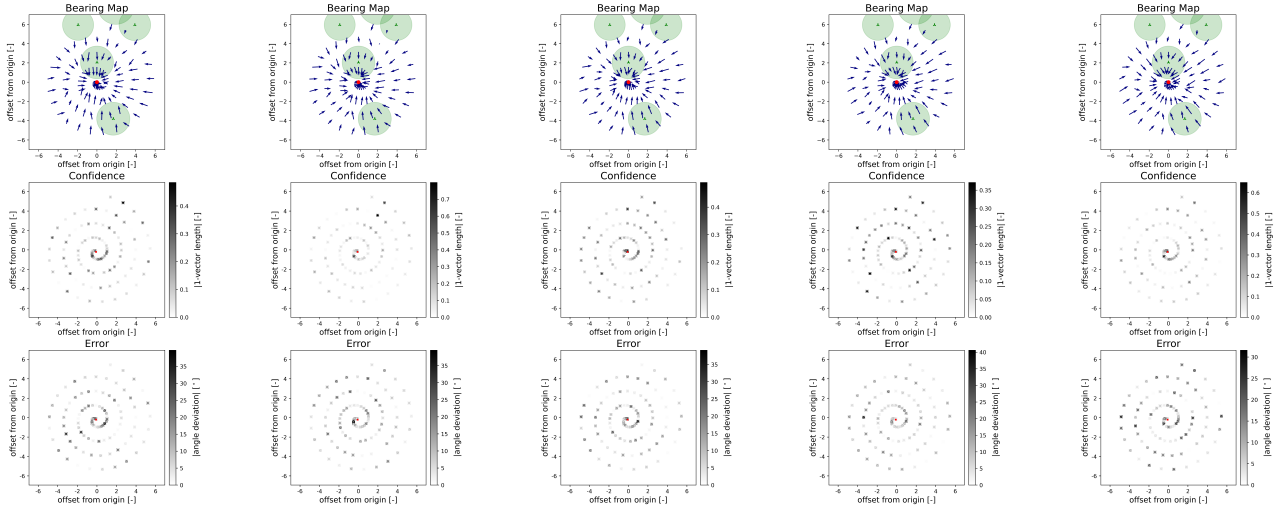
Fig. 12: Performance of network trained on an Archimedean **spiral** acquisition trajectory. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 190° (b) Gaze angle: 200° (c) Gaze angle: 210° (d) Gaze angle: 220° (e) Gaze angle: 230° (f) Gaze angle: 240°



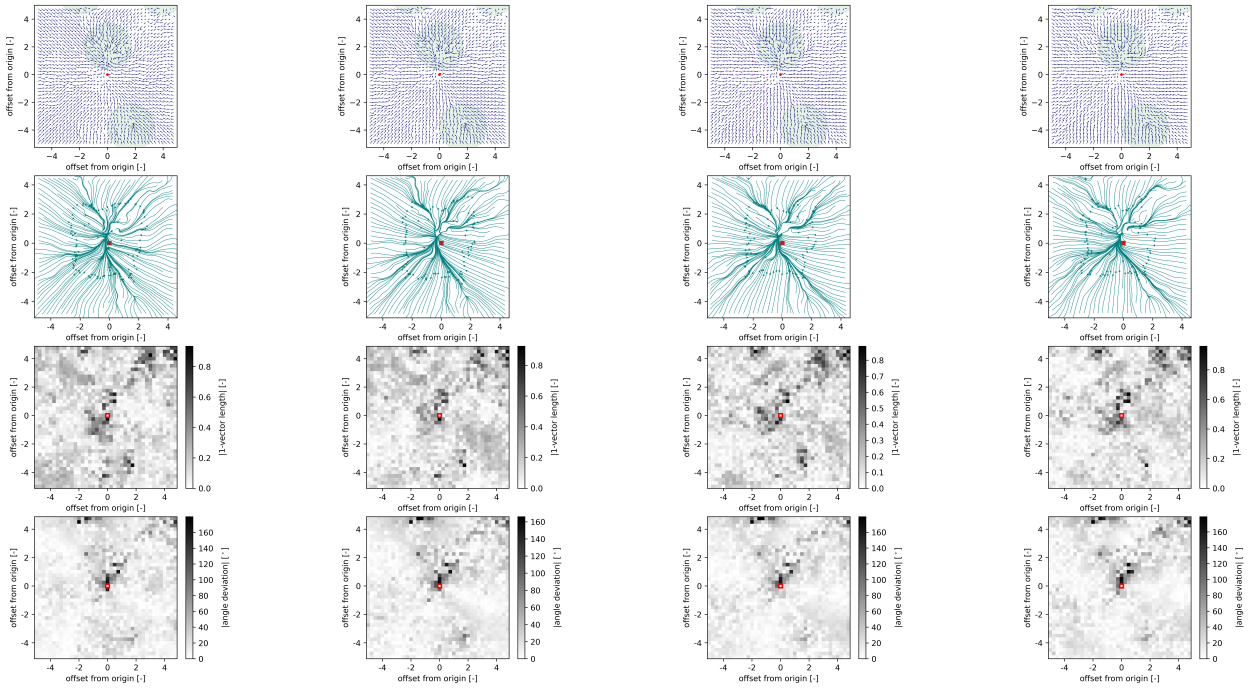
(g) Gaze angle: 250° (h) Gaze angle: 260° (i) Gaze angle: 270° (j) Gaze angle: 280° (k) Gaze angle: 290° (l) Gaze angle: 300°



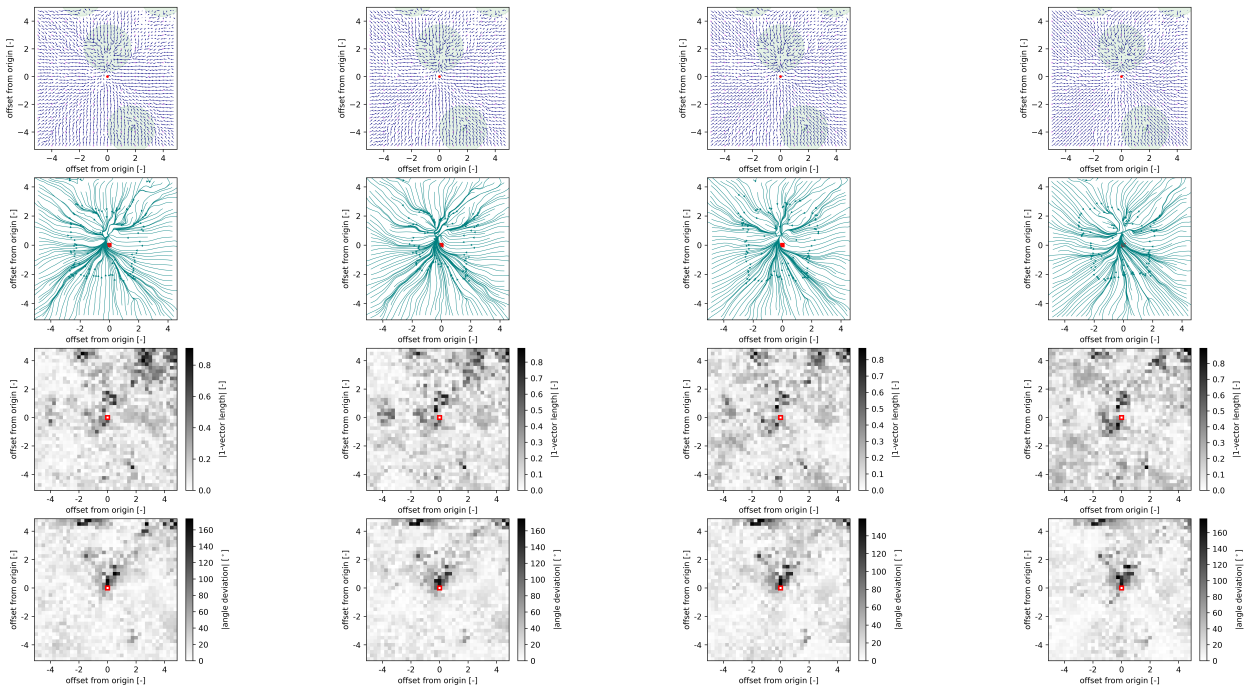
(m) Gaze angle: 310° (n) Gaze angle: 320° (o) Gaze angle: 330° (p) Gaze angle: 340° (q) Gaze angle: 350°

Fig. 13: Performance of network trained on an Archimedean **spiral** acquisition trajectory. Gaze angle is relative to vector pointing North/upwards.

APPENDIX D
SIMULATED GENERALIZATION PERFORMANCE FOR 360 GAZE ANGLES

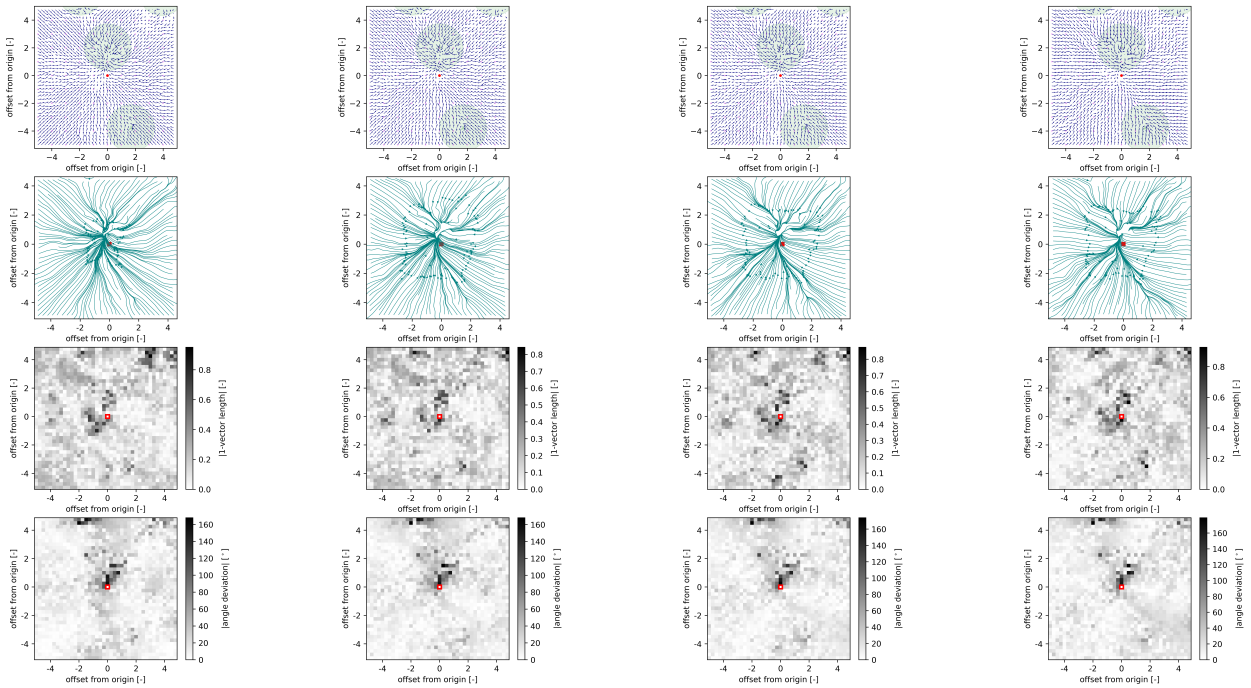


(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40°

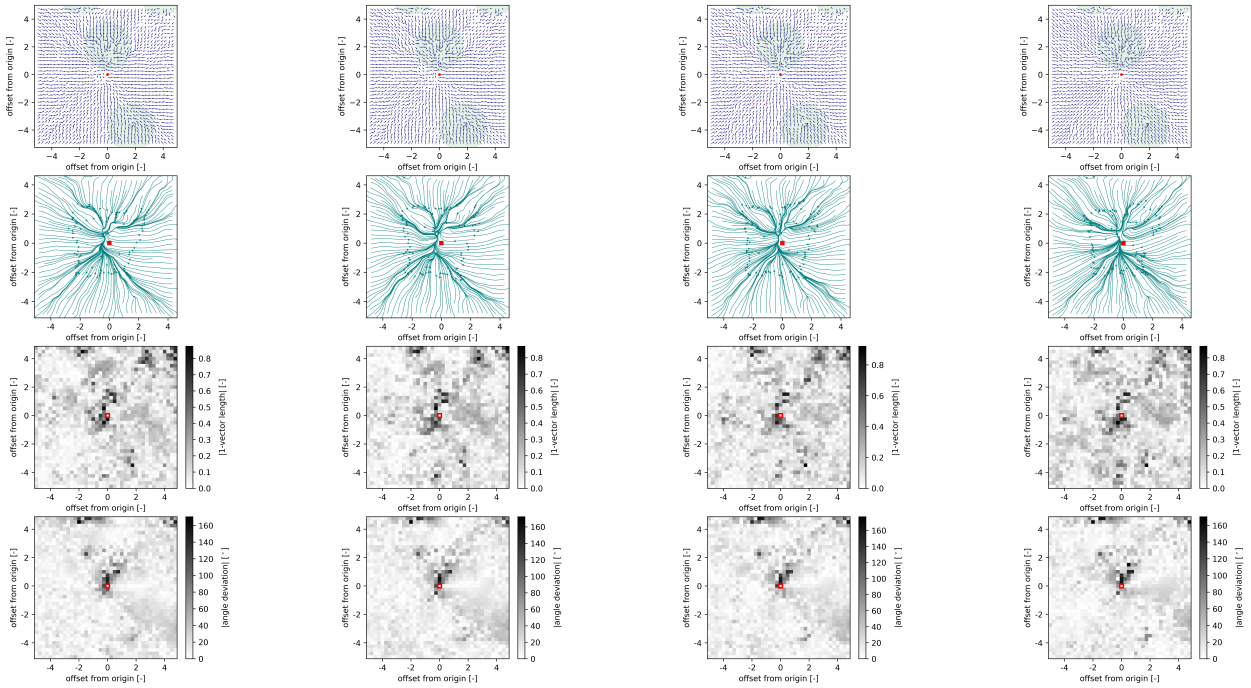


(e) Gaze angle: 50° (f) Gaze angle: 60° (g) Gaze angle: 70° (h) Gaze angle: 80°

Fig. 14: **Generalization** performance of network trained on a square 10×10 regular **grid** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

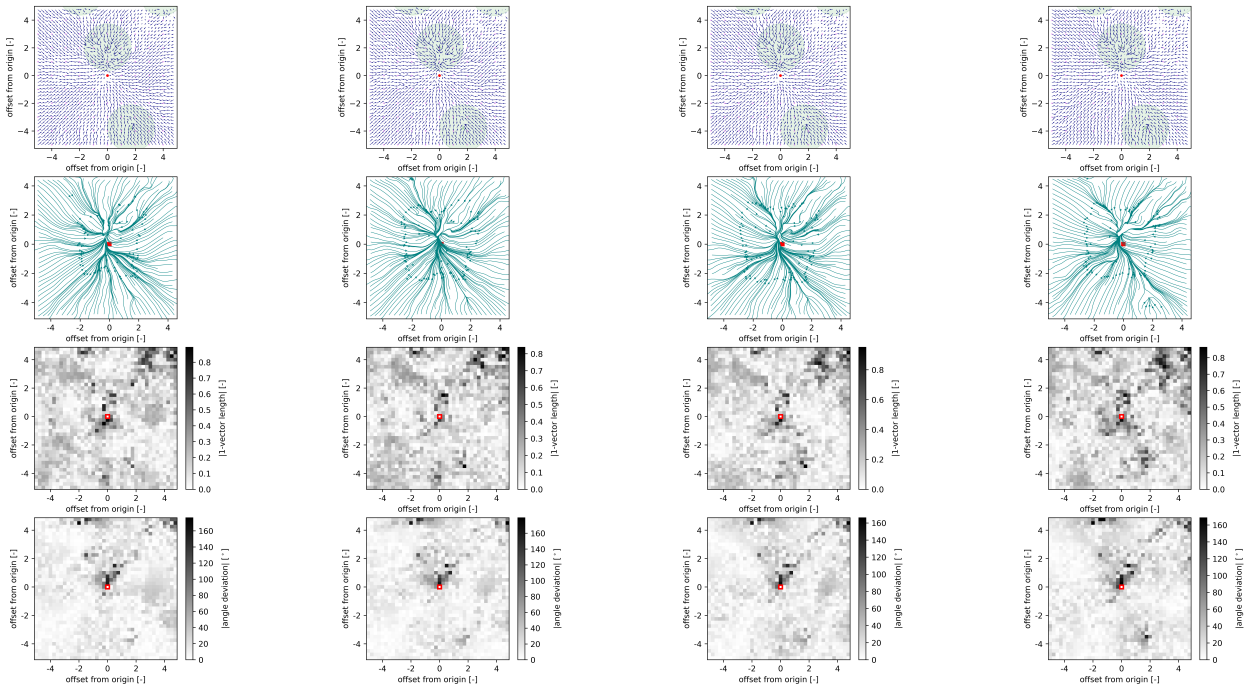


(a) Gaze angle: 90° (b) Gaze angle: 100° (c) Gaze angle: 110° (d) Gaze angle: 120°

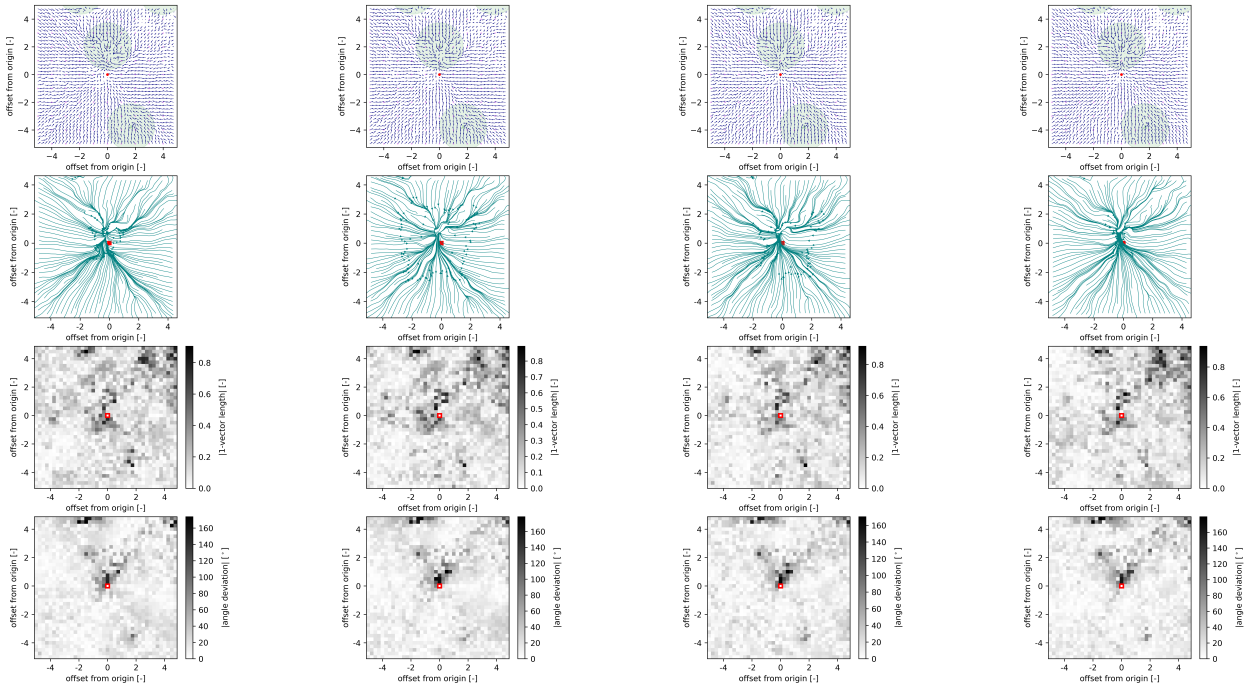


(e) Gaze angle: 130° (f) Gaze angle: 140° (g) Gaze angle: 150° (h) Gaze angle: 160°

Fig. 15: **Generalization** performance of network trained on a square 10×10 regular **grid** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

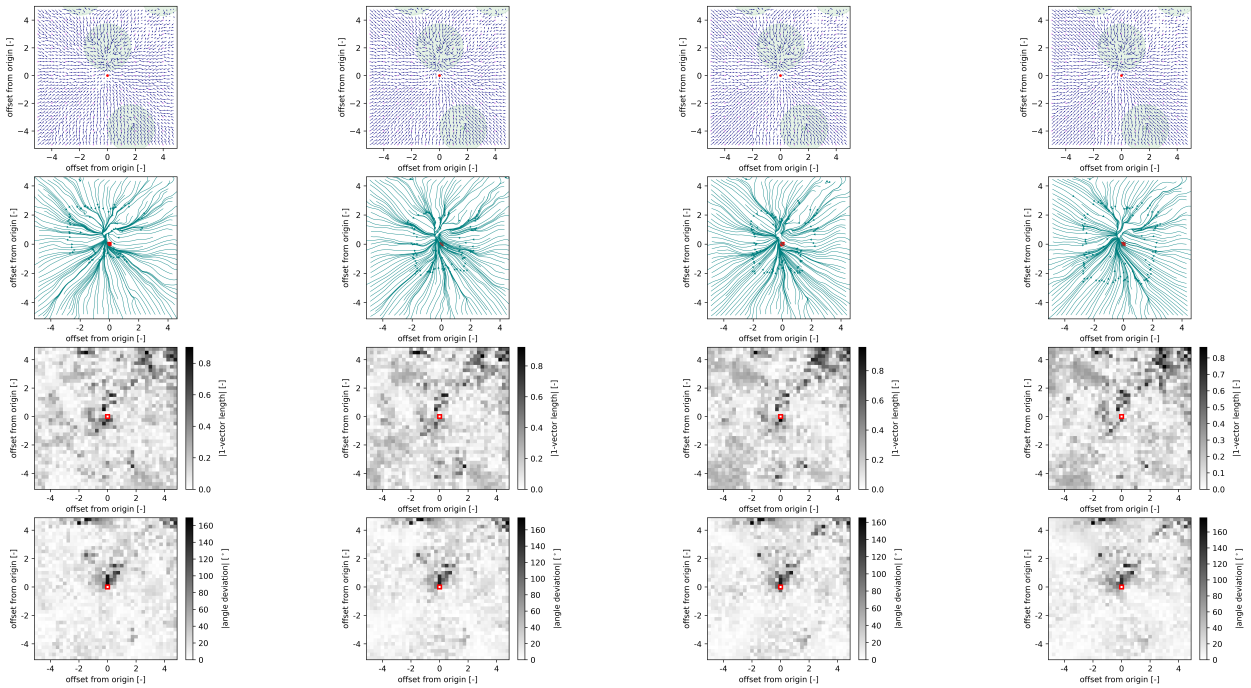


(a) Gaze angle: 170° (b) Gaze angle: 180° (c) Gaze angle: 190° (d) Gaze angle: 200°

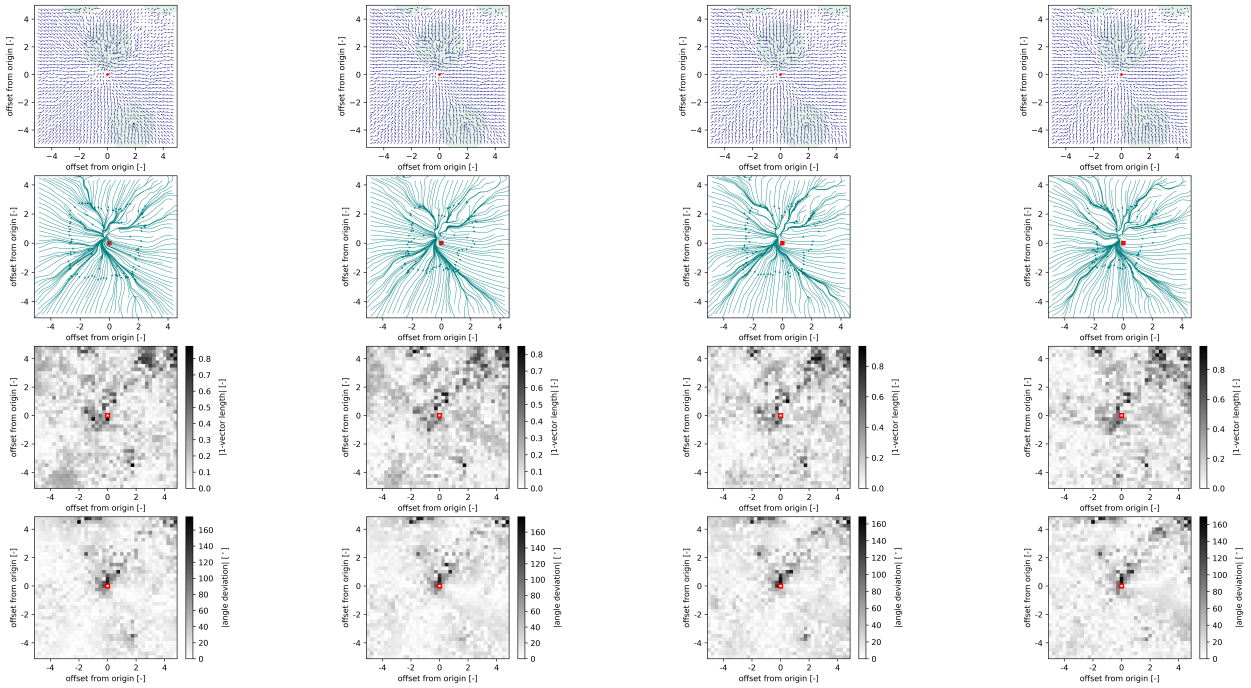


(e) Gaze angle: 210° (f) Gaze angle: 220° (g) Gaze angle: 230° (h) Gaze angle: 240°

Fig. 16: **Generalization** performance of network trained on a square 10×10 regular **grid** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

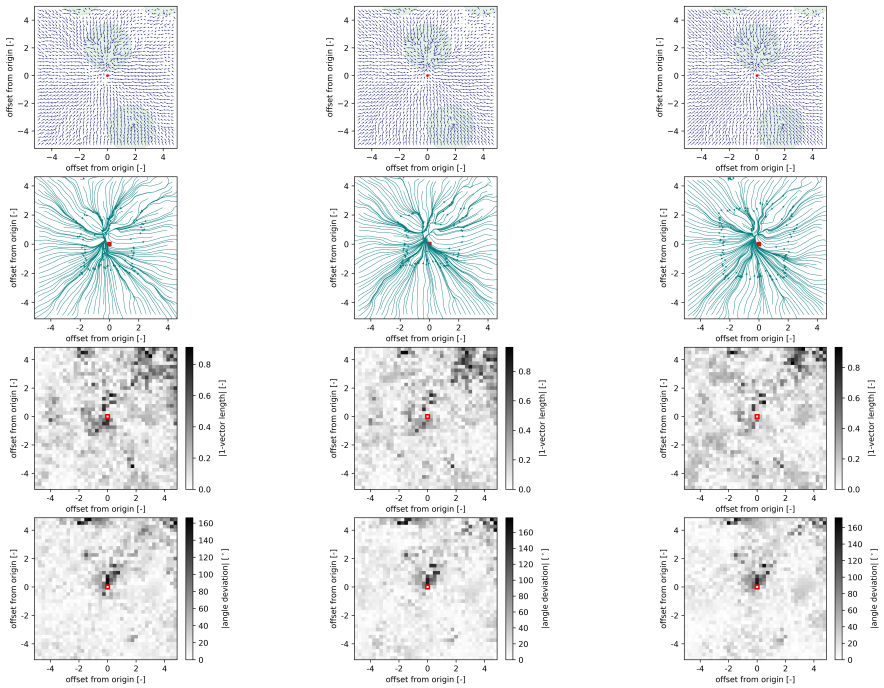


(a) Gaze angle: 250° (b) Gaze angle: 260° (c) Gaze angle: 270° (d) Gaze angle: 280°



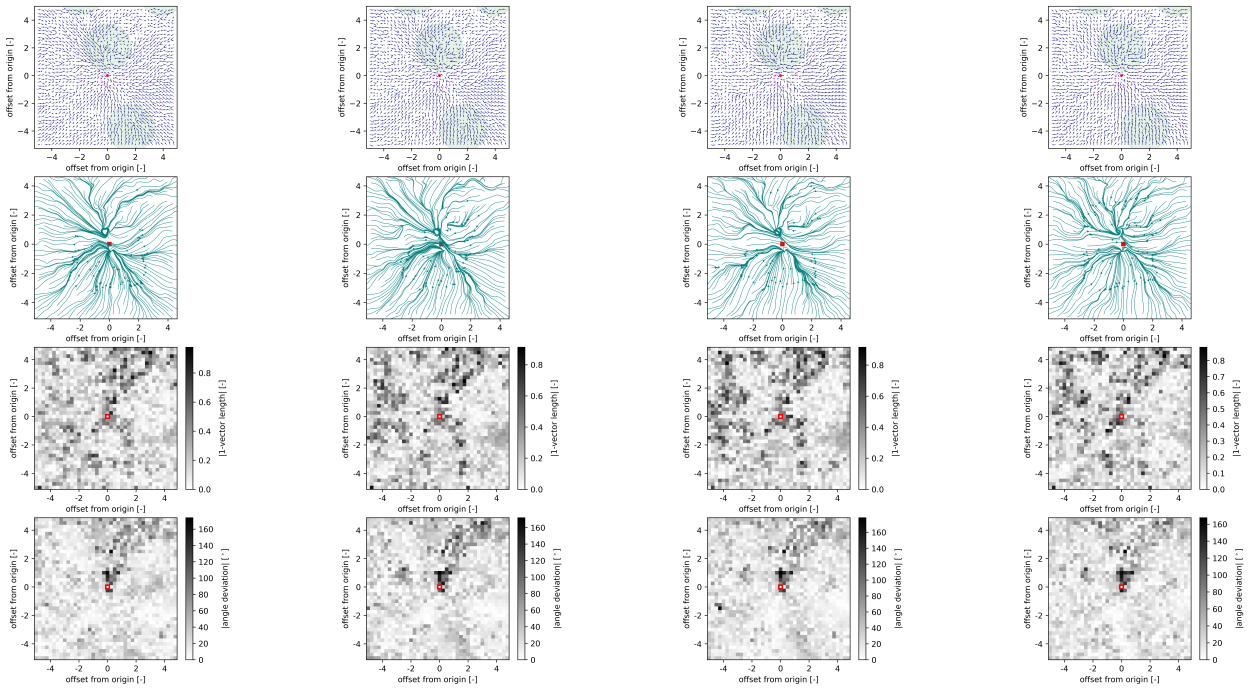
(e) Gaze angle: 290° (f) Gaze angle: 300° (g) Gaze angle: 310° (h) Gaze angle: 320°

Fig. 17: **Generalization** performance of network trained on a square 10×10 regular **grid** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

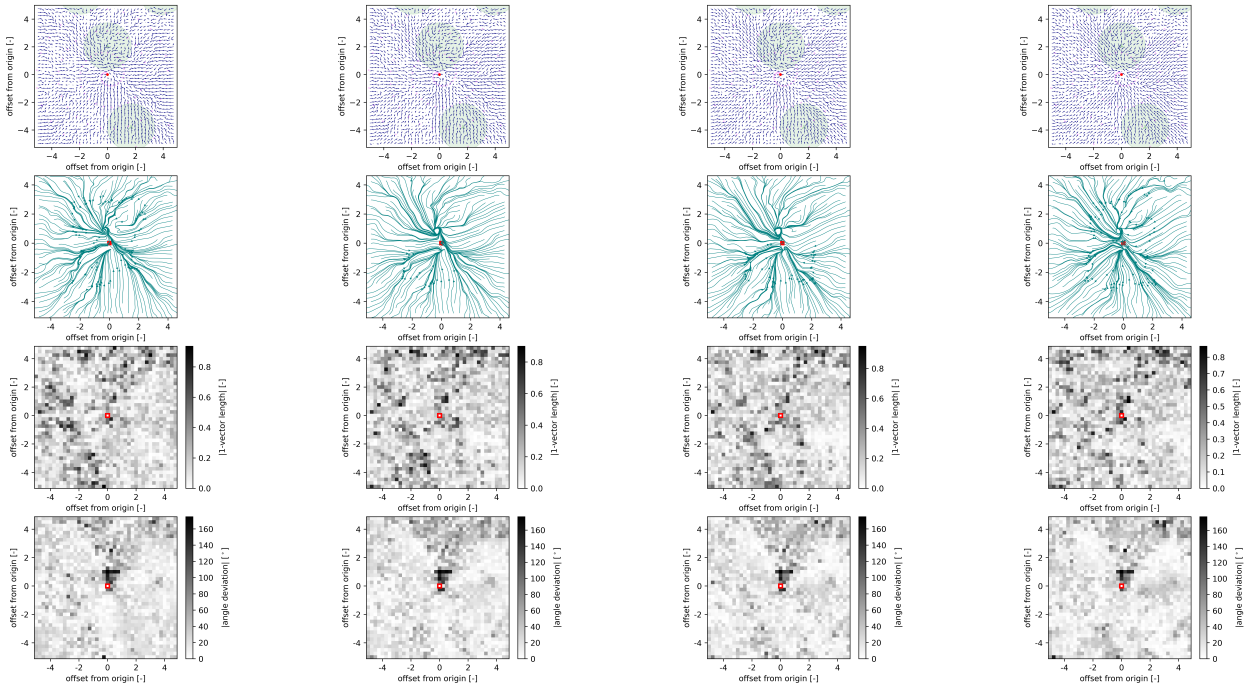


(a) Gaze angle: 330° (b) Gaze angle: 340° (c) Gaze angle: 350°

Fig. 18: **Generalization** performance of network trained on a square 10×10 regular **grid** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

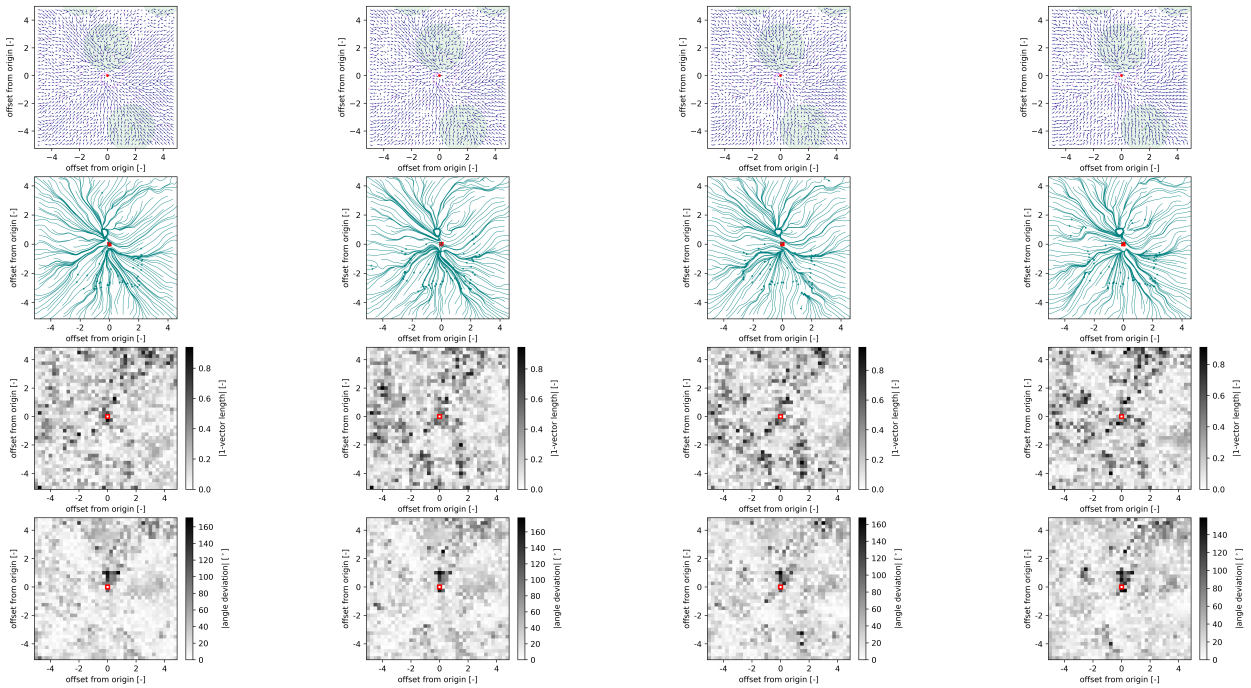


(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40°

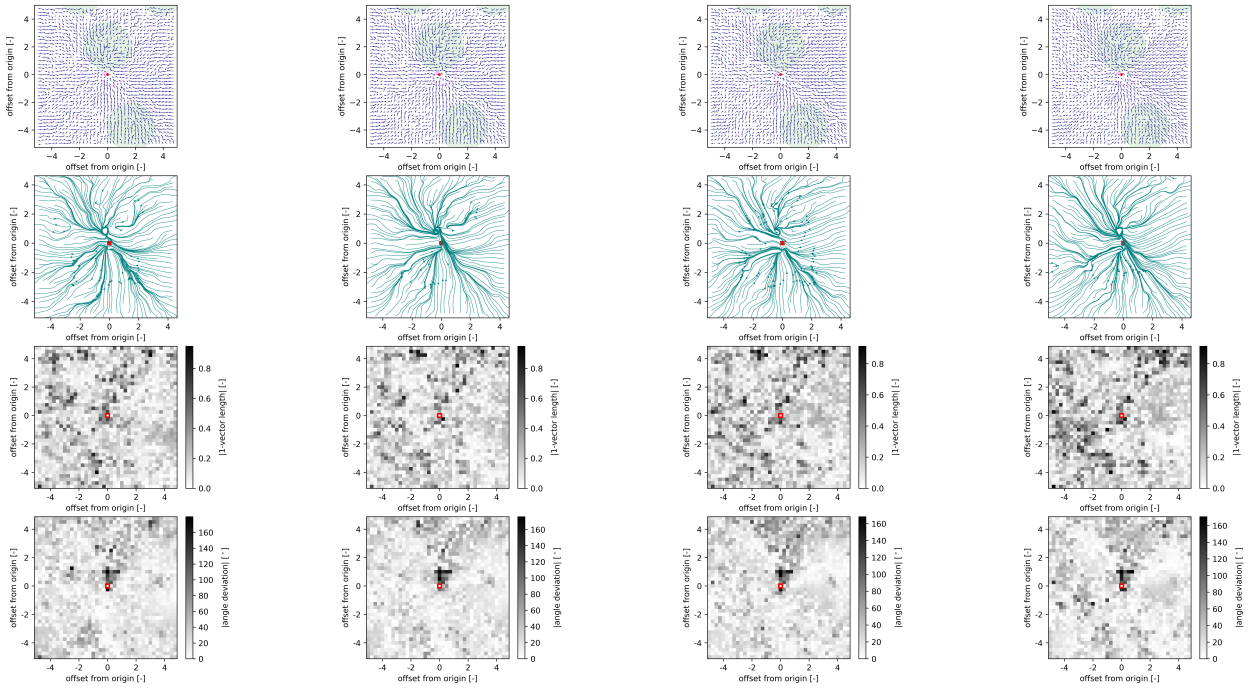


(e) Gaze angle: 50° (f) Gaze angle: 60° (g) Gaze angle: 70° (h) Gaze angle: 80°

Fig. 19: **Generalization** performance of network trained on an Archimedean **spiral** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

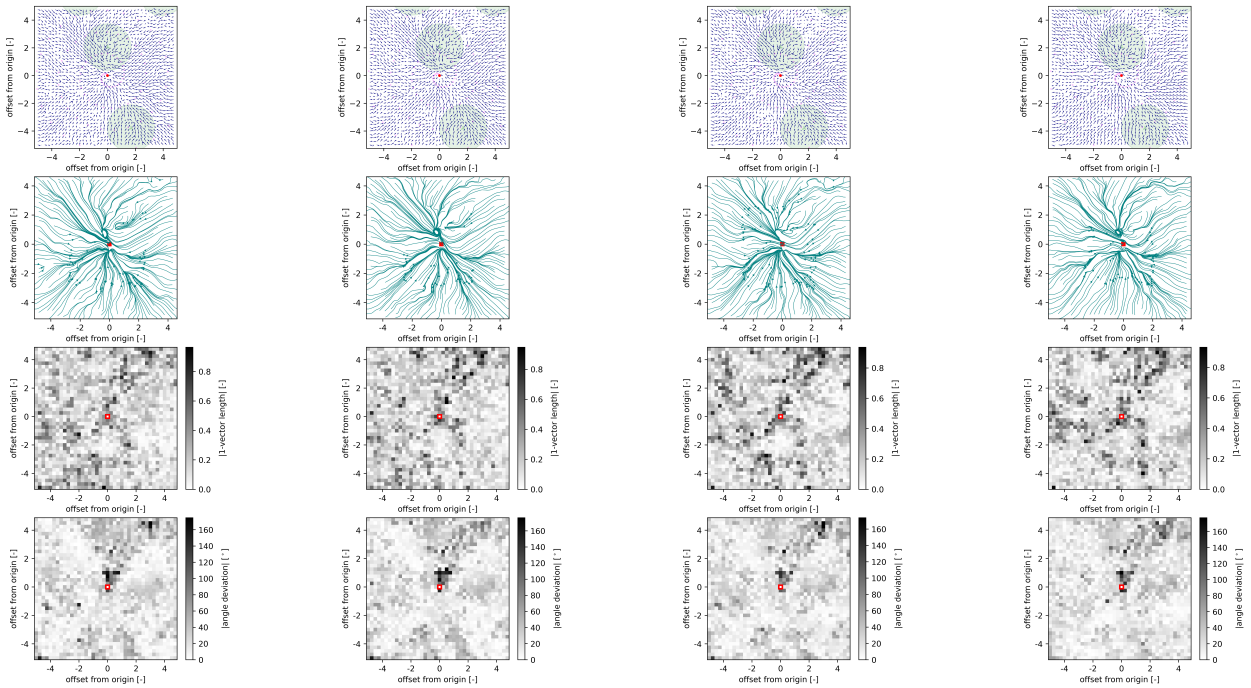


(a) Gaze angle: 90° (b) Gaze angle: 100° (c) Gaze angle: 110° (d) Gaze angle: 120°

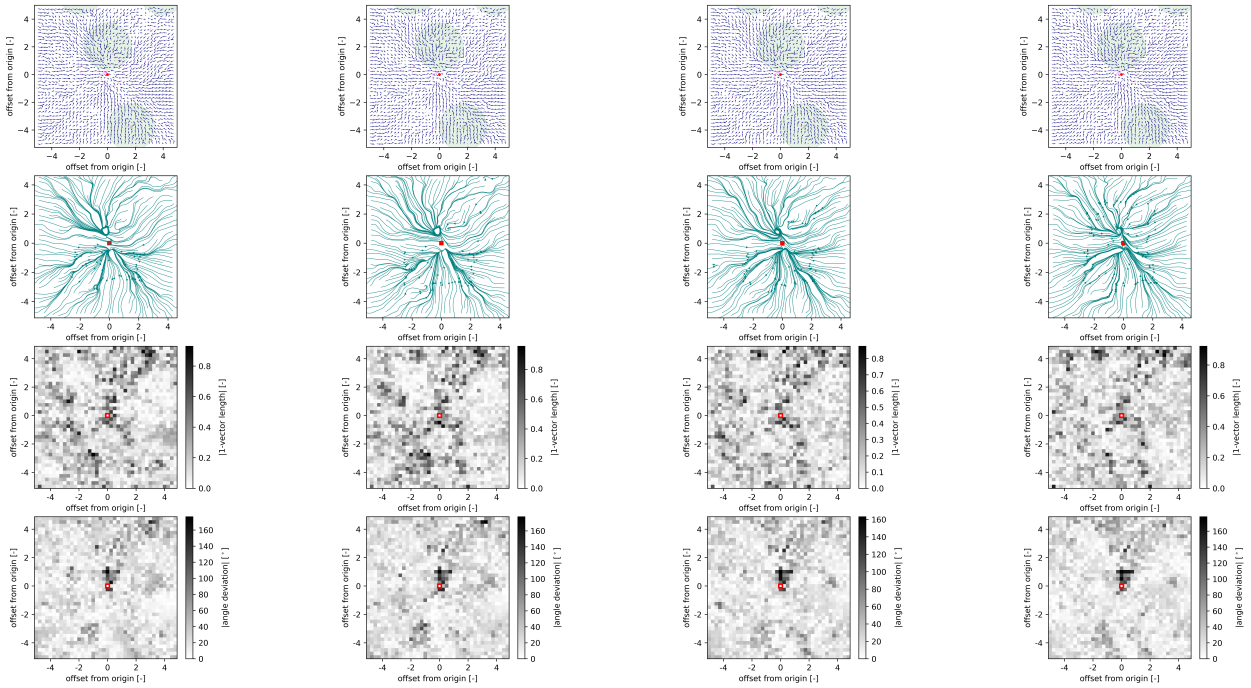


(e) Gaze angle: 130° (f) Gaze angle: 140° (g) Gaze angle: 150° (h) Gaze angle: 160°

Fig. 20: **Generalization** performance of network trained on an Archimedean **spiral** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

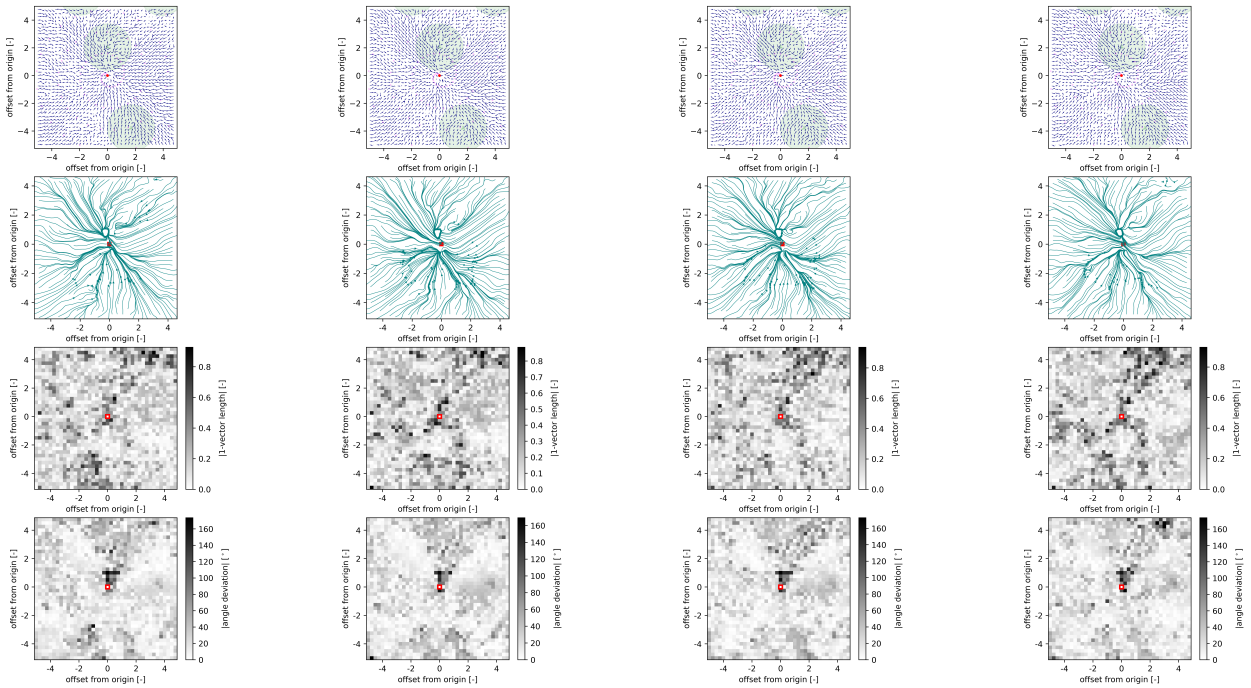


(a) Gaze angle: 170° (b) Gaze angle: 180° (c) Gaze angle: 190° (d) Gaze angle: 200°

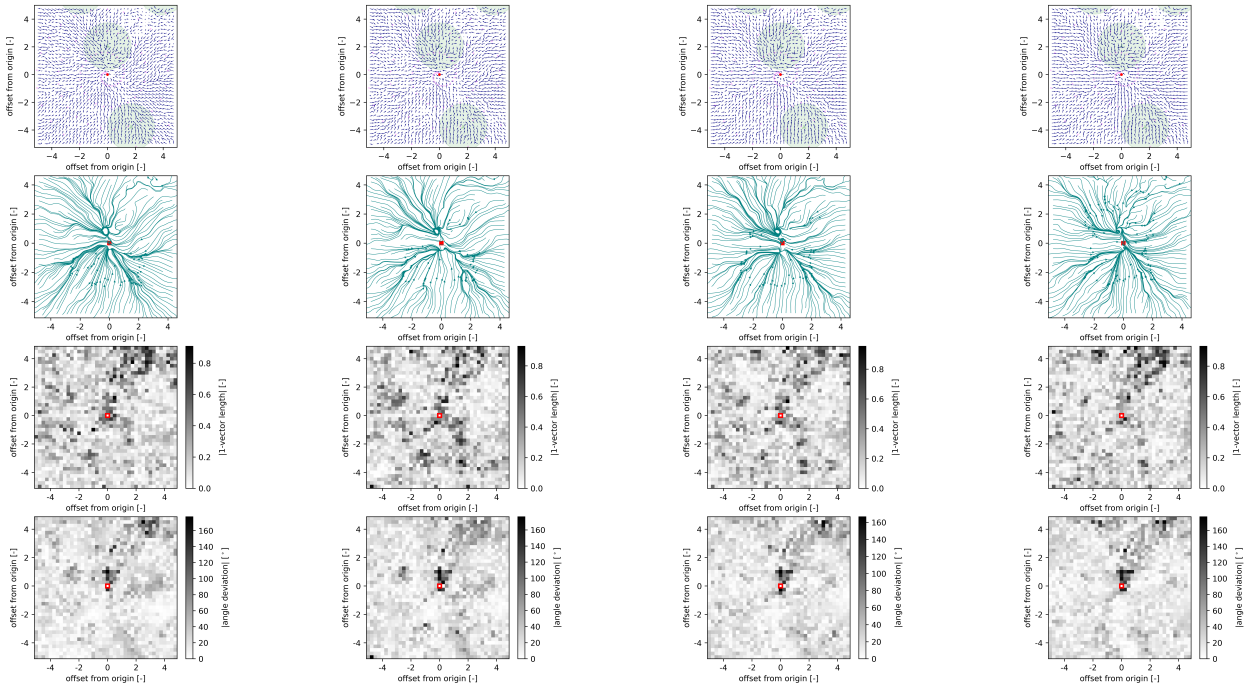


(e) Gaze angle: 210° (f) Gaze angle: 220° (g) Gaze angle: 230° (h) Gaze angle: 240°

Fig. 21: **Generalization** performance of network trained on an Archimedean **spiral** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

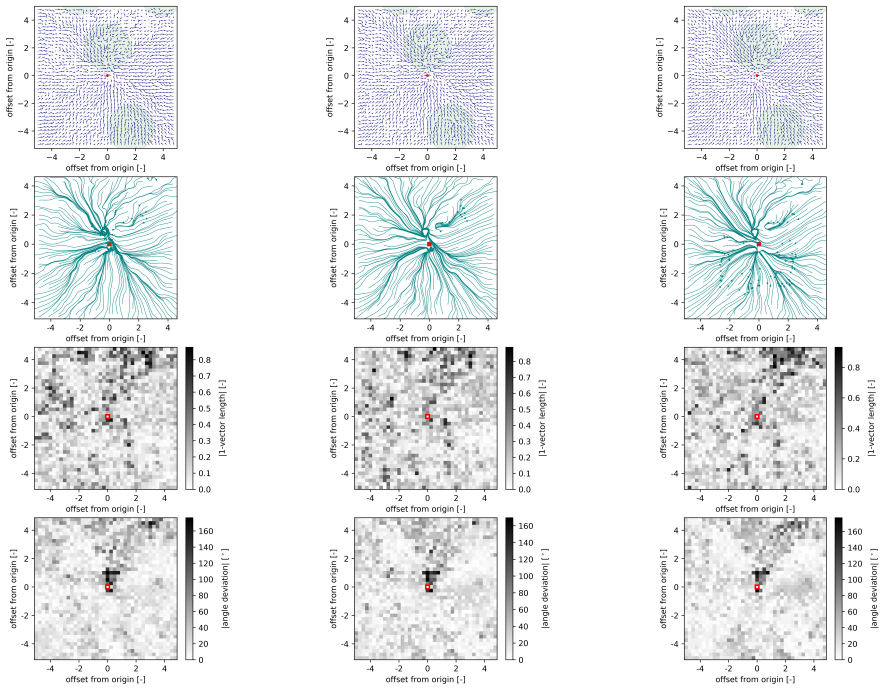


(a) Gaze angle: 250° (b) Gaze angle: 260° (c) Gaze angle: 270° (d) Gaze angle: 280°



(e) Gaze angle: 290° (f) Gaze angle: 300° (g) Gaze angle: 310° (h) Gaze angle: 320°

Fig. 22: **Generalization** performance of network trained on an Archimedean **spiral** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 330° (b) Gaze angle: 340° (c) Gaze angle: 350°

Fig. 23: **Generalization** performance of network trained on an Archimedean **spiral** acquisition trajectory, evaluated on a square 40×40 regular grid. Gaze angle is relative to vector pointing North/upwards.

APPENDIX E
EQUIPMENT AND LOCATION FOR REAL GRID DATASET



Fig. 24: Equipment for capturing the real grid dataset.



Fig. 25: Equipment for capturing the real grid dataset at location 1.



Fig. 26: Equipment for capturing the real grid dataset at location 1.



Fig. 27: Location 1.



Fig. 28: Location 1.



Fig. 29: Field adjacent to location 1.



Fig. 30: Sample locations at location 1.



Fig. 31: Equipment for capturing the real grid dataset at location 2.



Fig. 32: Location 2.



Fig. 33: Location 2.



Fig. 34: Sample locations at location 2.



Fig. 35: Sample locations at location 2.

APPENDIX F
REAL REGULAR GRID DATASET FOR 360 GAZE ANGLES AT LOCATION 1



(a) Offset from nest: $(-5, -4)$, Gaze angle: 0°



(b) Offset from nest: $(-5, -3)$, Gaze angle: 0°



(c) Offset from nest: $(-5, -2)$, Gaze angle: 0°



(d) Offset from nest: $(-5, -1)$, Gaze angle: 0°



(e) Offset from nest: $(-5, 0)$, Gaze angle: 0°



(f) Offset from nest: $(-5, 1)$, Gaze angle: 0°



(g) Offset from nest: $(-5, 2)$, Gaze angle: 0°



(h) Offset from nest: $(-5, 3)$, Gaze angle: 0°



(i) Offset from nest: $(-5, 4)$, Gaze angle: 0°

Fig. 36: Selection of sample locations on the square 10×10 regular **grid** used for training (**location 1**).



(a) Offset from nest: $(-4, -5)$, Gaze angle: 0°



(b) Offset from nest: $(-3, -5)$, Gaze angle: 0°



(c) Offset from nest: $(-2, -5)$, Gaze angle: 0°



(d) Offset from nest: $(-1, -5)$, Gaze angle: 0°



(e) Offset from nest: $(0, -5)$, Gaze angle: 0°



(f) Offset from nest: $(1, -5)$, Gaze angle: 0°



(g) Offset from nest: $(2, -5)$, Gaze angle: 0°



(h) Offset from nest: $(3, -5)$, Gaze angle: 0°



(i) Offset from nest: $(4, -5)$, Gaze angle: 0°

Fig. 37: Selection of sample locations on the square 10×10 regular **grid** used for training (**location 1**).



(a) Offset from nest: $(0, 0)$, Gaze angle: 10°



(b) Offset from nest: $(0, 0)$, Gaze angle: 20°



(c) Offset from nest: $(0, 0)$, Gaze angle: 30°



(d) Offset from nest: $(0, 0)$, Gaze angle: 40°

Fig. 38: Selection of sample locations on the square 10×10 regular **grid** used for training (**location 1**).



(a) Offset from nest: (0, 0), Gaze angle: 50°



(b) Offset from nest: (0, 0), Gaze angle: 60°



(c) Offset from nest: (0, 0), Gaze angle: 70°



(d) Offset from nest: (0, 0), Gaze angle: 80°



(e) Offset from nest: (0, 0), Gaze angle: 90°



(f) Offset from nest: (0, 0), Gaze angle: 100°



(g) Offset from nest: (0, 0), Gaze angle: 110°



(h) Offset from nest: (0, 0), Gaze angle: 120°



(i) Offset from nest: (0, 0), Gaze angle: 130°



(j) Offset from nest: (0, 0), Gaze angle: 140°



(k) Offset from nest: (0, 0), Gaze angle: 150°



(l) Offset from nest: (0, 0), Gaze angle: 160°



(m) Offset from nest: (0, 0), Gaze angle: 170°



(n) Offset from nest: (0, 0), Gaze angle: 180°



(o) Offset from nest: (0, 0), Gaze angle: 190°



(p) Offset from nest: (0, 0), Gaze angle: 200°



(q) Offset from nest: (0, 0), Gaze angle: 210°



(r) Offset from nest: (0, 0), Gaze angle: 220°



(s) Offset from nest: (0, 0), Gaze angle: 230°



(t) Offset from nest: (0, 0), Gaze angle: 240°

Fig. 39: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 1**).



(a) Offset from nest: (0, 0), Gaze angle: 250°



(b) Offset from nest: (0, 0), Gaze angle: 260°



(c) Offset from nest: (0, 0), Gaze angle: 270°



(d) Offset from nest: (0, 0), Gaze angle: 280°



(e) Offset from nest: (0, 0), Gaze angle: 290°



(f) Offset from nest: (0, 0), Gaze angle: 300°



(g) Offset from nest: (0, 0), Gaze angle: 310°



(h) Offset from nest: (0, 0), Gaze angle: 320°



(i) Offset from nest: (0, 0), Gaze angle: 330°



(j) Offset from nest: (0, 0), Gaze angle: 340°

Fig. 40: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 1**).



(a) Offset from nest: (0, 0), Gaze angle: 350°

Fig. 41: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 1**).

APPENDIX G
REAL REGULAR GRID DATASET FOR 360 GAZE ANGLES AT LOCATION 2



(a) Offset from nest: $(-5, -4)$, Gaze angle: 0°



(b) Offset from nest: $(-5, -3)$, Gaze angle: 0°



(c) Offset from nest: $(-5, -2)$, Gaze angle: 0°



(d) Offset from nest: $(-5, -1)$, Gaze angle: 0°



(e) Offset from nest: $(-5, 0)$, Gaze angle: 0°



(f) Offset from nest: $(-5, 1)$, Gaze angle: 0°



(g) Offset from nest: $(-5, 2)$, Gaze angle: 0°



(h) Offset from nest: $(-5, 3)$, Gaze angle: 0°



(i) Offset from nest: $(-5, 4)$, Gaze angle: 0°

Fig. 42: Selection of sample locations on the square 10×10 regular **grid** used for training (**location 2**).



(a) Offset from nest: $(-4, -5)$, Gaze angle: 0°



(b) Offset from nest: $(-3, -5)$, Gaze angle: 0°



(c) Offset from nest: $(-2, -5)$, Gaze angle: 0°



(d) Offset from nest: $(-1, -5)$, Gaze angle: 0°



(e) Offset from nest: $(0, -5)$, Gaze angle: 0°



(f) Offset from nest: $(1, -5)$, Gaze angle: 0°



(g) Offset from nest: $(2, -5)$, Gaze angle: 0°



(h) Offset from nest: $(3, -5)$, Gaze angle: 0°



(i) Offset from nest: $(4, -5)$, Gaze angle: 0°

Fig. 43: Selection of sample locations on the square 10×10 regular **grid** used for training (**location 2**).



(a) Offset from nest: $(0, 0)$, Gaze angle: 10°



(b) Offset from nest: $(0, 0)$, Gaze angle: 20°



(c) Offset from nest: $(0, 0)$, Gaze angle: 30°



(d) Offset from nest: $(0, 0)$, Gaze angle: 40°

Fig. 44: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 2**).



(a) Offset from nest: (0, 0), Gaze angle: 50°



(b) Offset from nest: (0, 0), Gaze angle: 60°



(c) Offset from nest: (0, 0), Gaze angle: 70°



(d) Offset from nest: (0, 0), Gaze angle: 80°



(e) Offset from nest: (0, 0), Gaze angle: 90°



(f) Offset from nest: (0, 0), Gaze angle: 100°



(g) Offset from nest: (0, 0), Gaze angle: 110°



(h) Offset from nest: (0, 0), Gaze angle: 120°



(i) Offset from nest: (0, 0), Gaze angle: 130°



(j) Offset from nest: (0, 0), Gaze angle: 140°



(k) Offset from nest: (0, 0), Gaze angle: 150°



(l) Offset from nest: (0, 0), Gaze angle: 160°



(m) Offset from nest: (0, 0), Gaze angle: 170°



(n) Offset from nest: (0, 0), Gaze angle: 180°



(o) Offset from nest: (0, 0), Gaze angle: 190°



(p) Offset from nest: (0, 0), Gaze angle: 200°



(q) Offset from nest: (0, 0), Gaze angle: 210°



(r) Offset from nest: (0, 0), Gaze angle: 220°



(s) Offset from nest: (0, 0), Gaze angle: 230°



(t) Offset from nest: (0, 0), Gaze angle: 240°

Fig. 45: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 2**).



(a) Offset from nest: (0, 0), Gaze angle: 250°



(b) Offset from nest: (0, 0), Gaze angle: 260°



(c) Offset from nest: (0, 0), Gaze angle: 270°



(d) Offset from nest: (0, 0), Gaze angle: 280°



(e) Offset from nest: (0, 0), Gaze angle: 290°



(f) Offset from nest: (0, 0), Gaze angle: 300°



(g) Offset from nest: (0, 0), Gaze angle: 310°



(h) Offset from nest: (0, 0), Gaze angle: 320°



(i) Offset from nest: (0, 0), Gaze angle: 330°



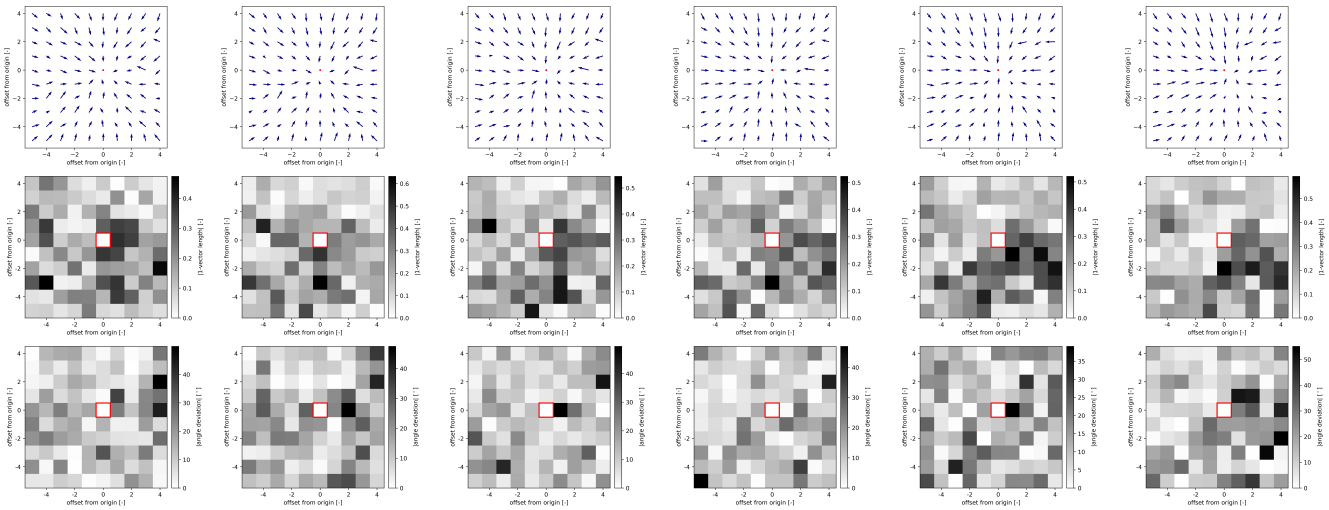
(j) Offset from nest: (0, 0), Gaze angle: 340°



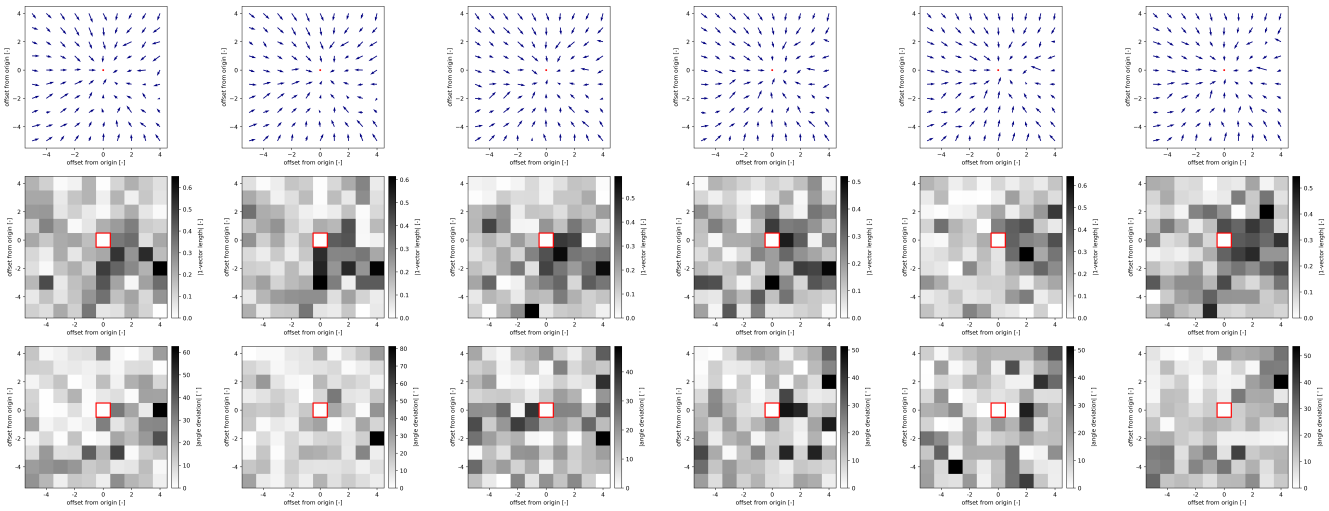
(k) Offset from nest: (0, 0), Gaze angle: 350°

Fig. 46: Selection of sample gaze angles on the square 10×10 regular **grid** used for training (**location 2**).

APPENDIX H
REAL TRAINING PERFORMANCE FOR 360 GAZE ANGLES

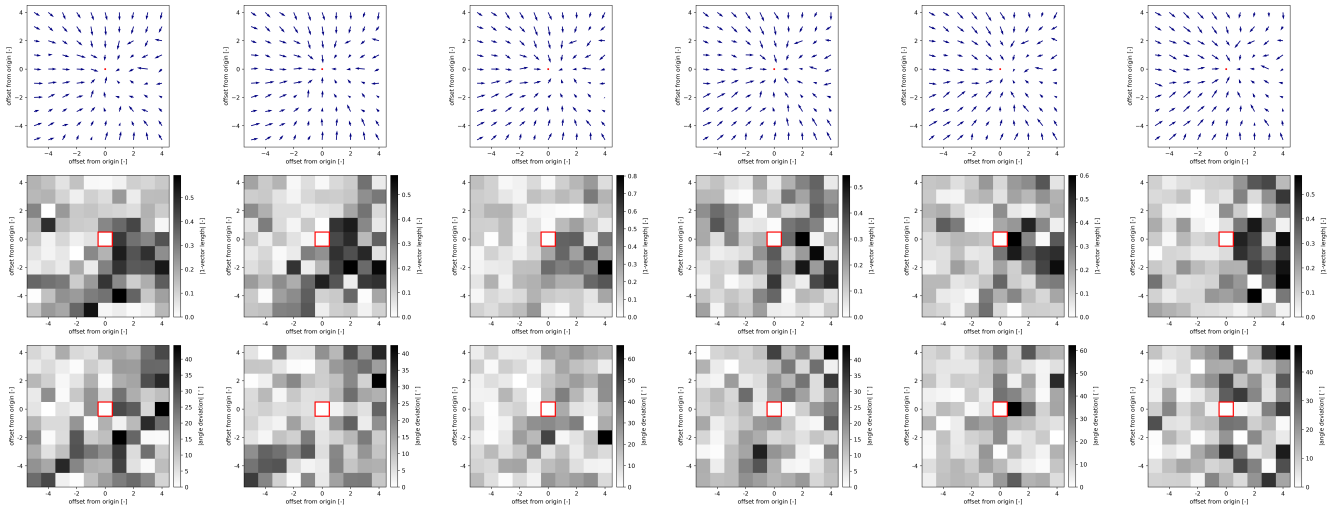


(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40° (e) Gaze angle: 50° (f) Gaze angle: 60°

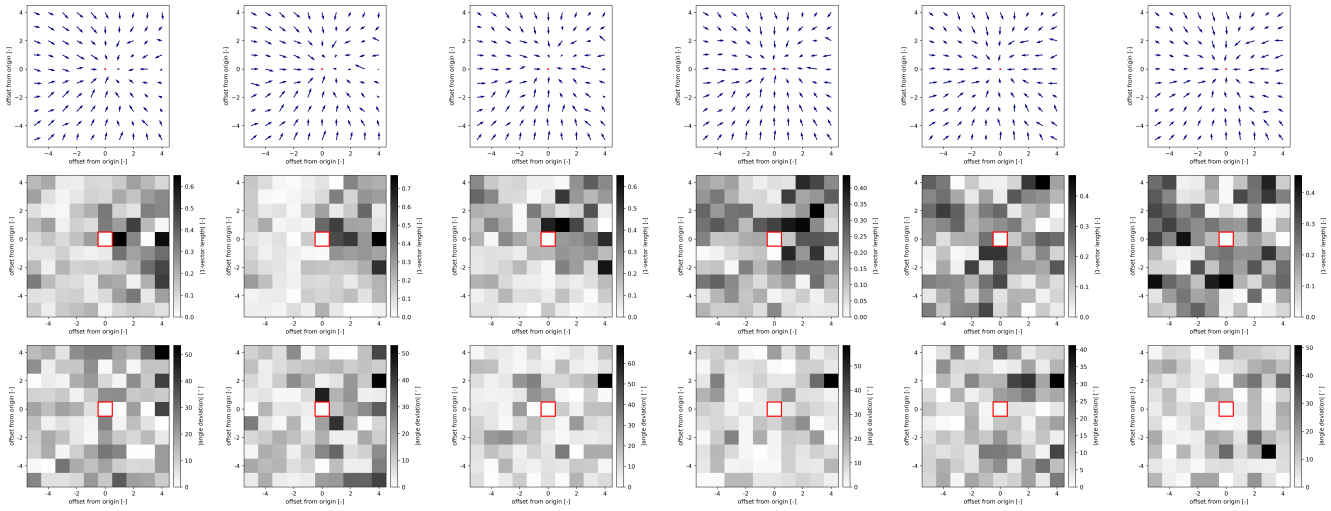


(g) Gaze angle: 70° (h) Gaze angle: 80° (i) Gaze angle: 90° (j) Gaze angle: 100° (k) Gaze angle: 110° (l) Gaze angle: 120°

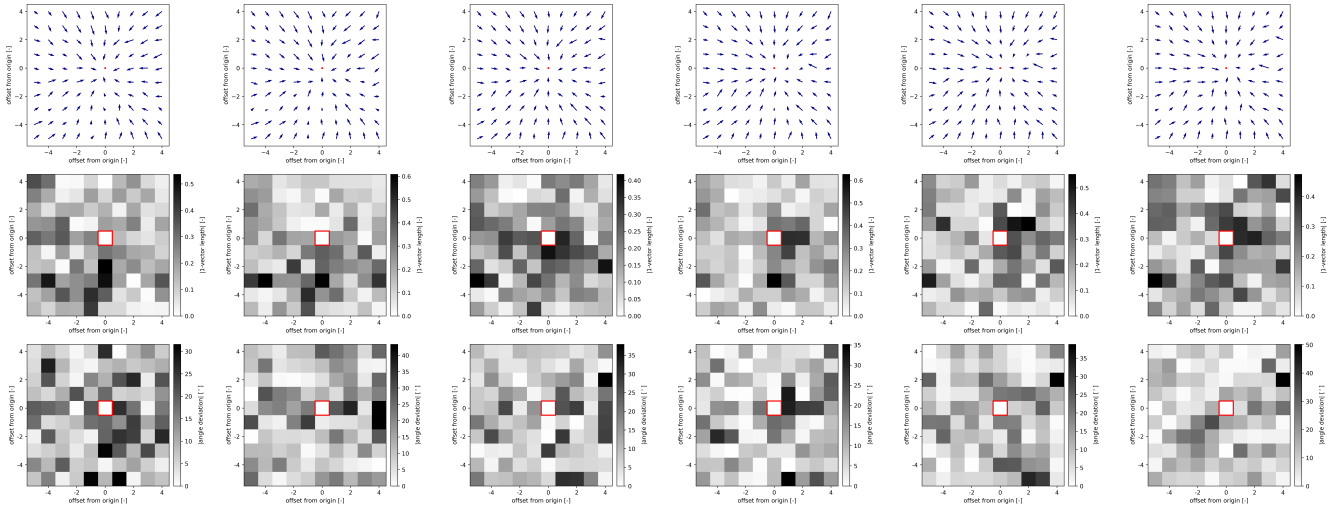
Fig. 47: Performance of network trained on a square 10×10 regular **grid** acquisition trajectory at **location 1**. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 130° (b) Gaze angle: 140° (c) Gaze angle: 150° (d) Gaze angle: 160° (e) Gaze angle: 170° (f) Gaze angle: 180°

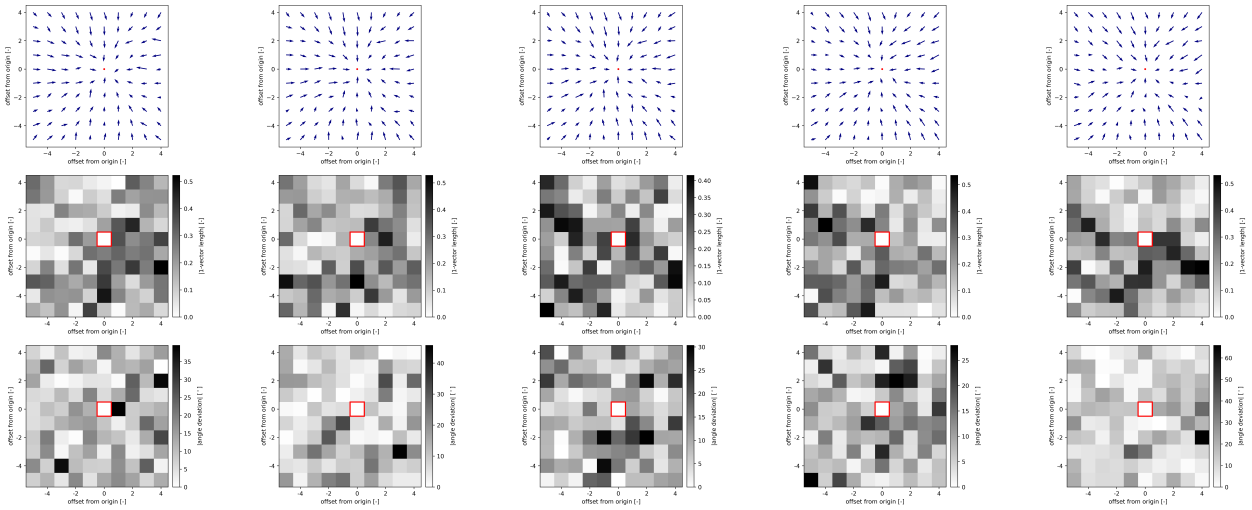


(g) Gaze angle: 190° (h) Gaze angle: 200° (i) Gaze angle: 210° (j) Gaze angle: 220° (k) Gaze angle: 230° (l) Gaze angle: 240°



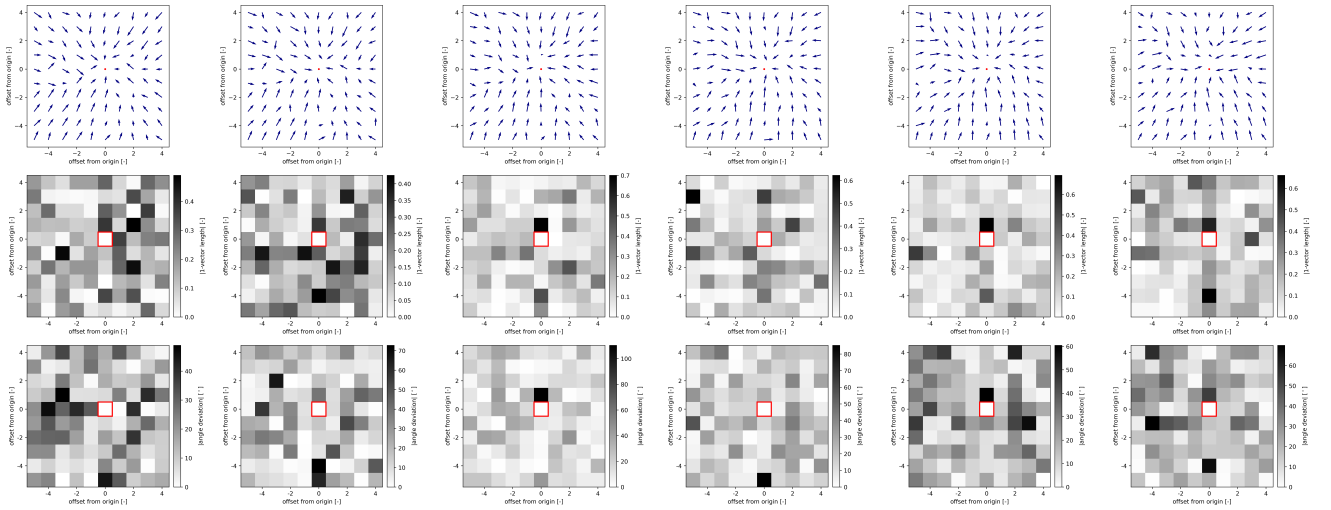
(m) Gaze angle: 250° (n) Gaze angle: 260° (o) Gaze angle: 270° (p) Gaze angle: 280° (q) Gaze angle: 290° (r) Gaze angle: 300°

Fig. 48: Performance of network trained on a square 10×10 regular **grid** acquisition trajectory at **location 1**. Gaze angle is relative to vector pointing North/upwards.

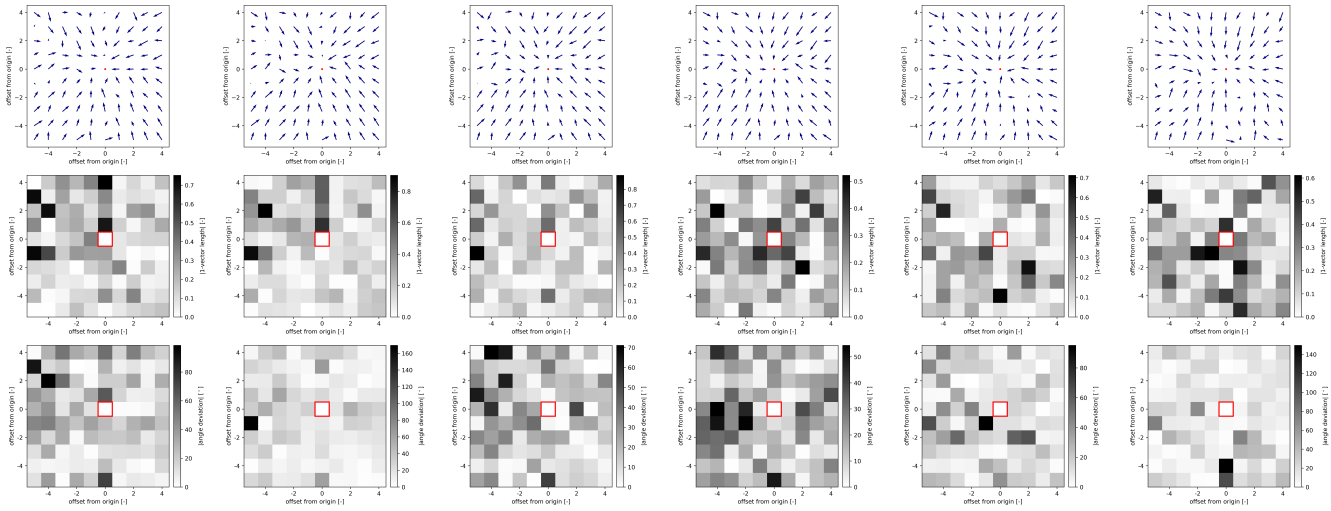


(a) Gaze angle: 310° (b) Gaze angle: 320° (c) Gaze angle: 330° (d) Gaze angle: 340° (e) Gaze angle: 350°

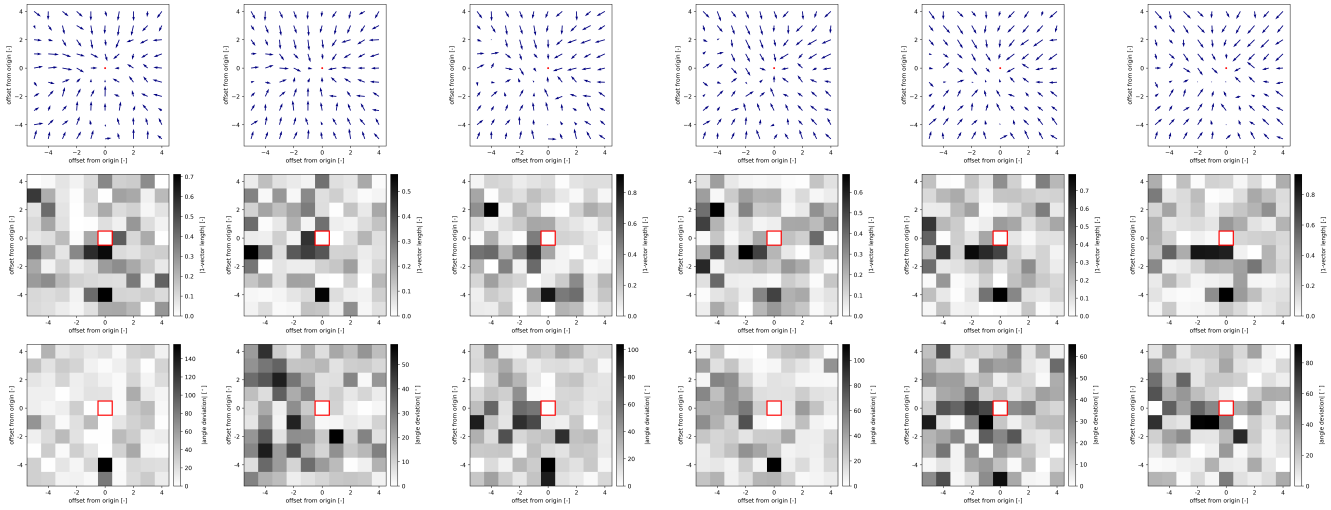
Fig. 49: Performance of network trained on a square 10×10 regular **grid** acquisition trajectory at **location 1**. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 10° (b) Gaze angle: 20° (c) Gaze angle: 30° (d) Gaze angle: 40° (e) Gaze angle: 50° (f) Gaze angle: 60°

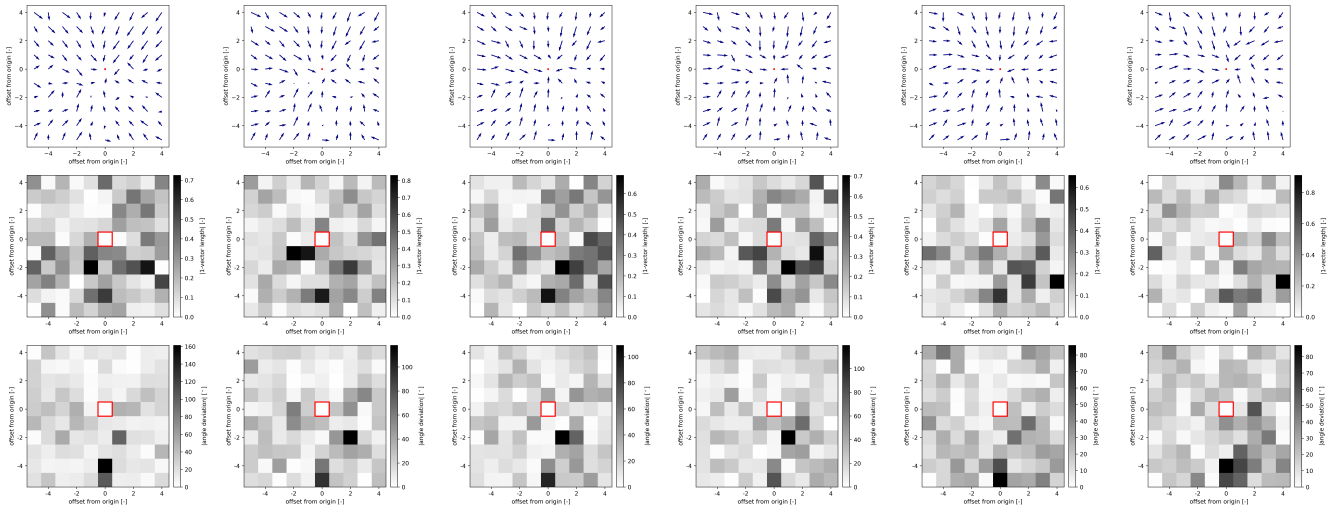


(g) Gaze angle: 70° (h) Gaze angle: 80° (i) Gaze angle: 90° (j) Gaze angle: 100° (k) Gaze angle: 110° (l) Gaze angle: 120°

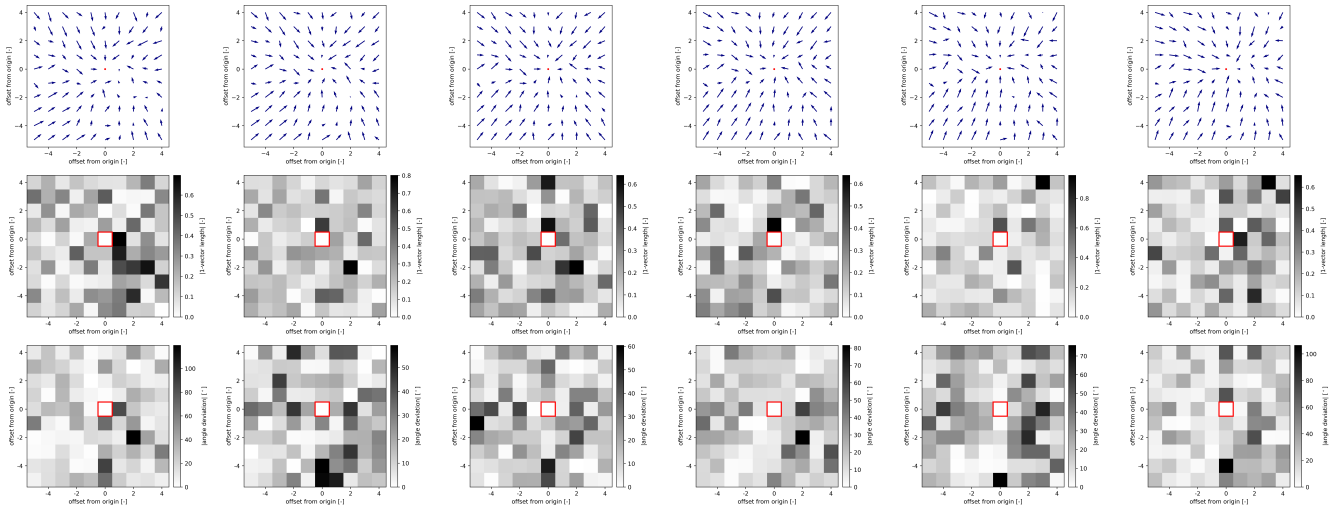


(m) Gaze angle: 130° (n) Gaze angle: 140° (o) Gaze angle: 150° (p) Gaze angle: 160° (q) Gaze angle: 170° (r) Gaze angle: 180°

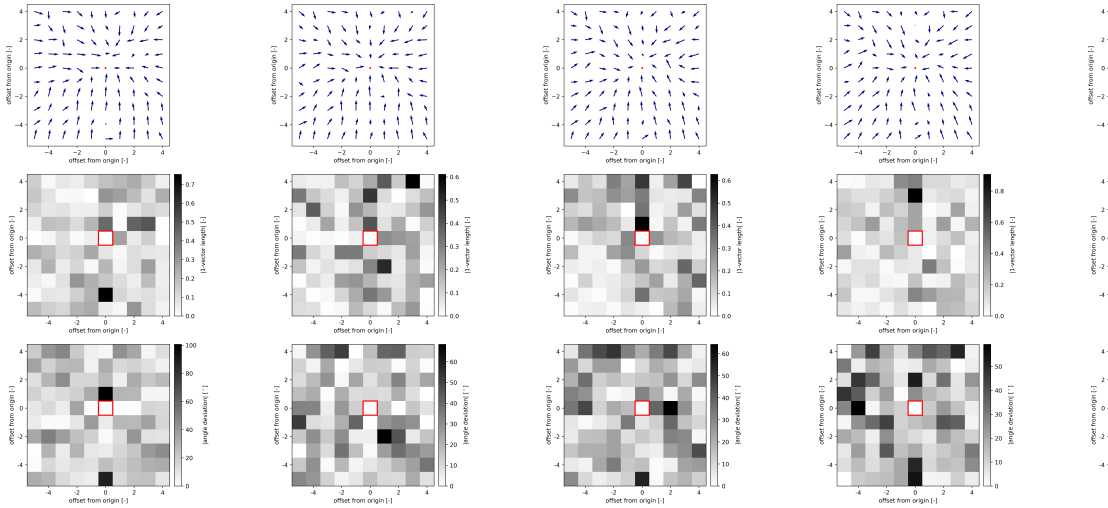
Fig. 50: Performance of network trained on a square 10×10 regular **grid** acquisition trajectory at **location 2**. Gaze angle is relative to vector pointing North/upwards.



(a) Gaze angle: 190° (b) Gaze angle: 200° (c) Gaze angle: 210° (d) Gaze angle: 220° (e) Gaze angle: 230° (f) Gaze angle: 240°



(g) Gaze angle: 250° (h) Gaze angle: 260° (i) Gaze angle: 270° (j) Gaze angle: 280° (k) Gaze angle: 290° (l) Gaze angle: 300°



(m) Gaze angle: 310° (n) Gaze angle: 320° (o) Gaze angle: 330° (p) Gaze angle: 340° (q) Gaze angle: 350°

Fig. 51: Performance of network trained on a square 10×10 regular **grid** acquisition trajectory at **location 2**. Gaze angle is relative to vector pointing North/upwards.

Part II

Relevant Literature

2

Insect Intelligence: Findings from Biology

This chapter aims to give an outline of the different approaches various biologists have used to study insect intelligence over the years. Initially, it has been chosen to briefly explain the motivation behind modeling insects in Section 2.1. Although a little philosophical, it is deemed a good starting point to categorize the many works that need to be evaluated. The identification of specific tools, such as path integration and a sky compass, with a vital role in guiding the animal during navigation is described in Section 2.2. Lastly, Section 2.3 gives a detailed overview of how insect conduct their learning of the environment.

2.1. Understanding the Modeling Effort

To understand the multi-faceted field of insect navigation it is often helpful to categorize research according to a clear and concise frame of reference. F. Le Moël and A. Wystrach have applied Marr's philosophical abstraction levels, commonly used in cognitive psychology, to the problem of understanding insect behavior [12]. They argue that such behaviors cannot be understood by solely studying a single module, such levels need to be seen as an intricate whole. Instead of the classic levels: (1) *computational*, (2) *algorithmic*, and (3) *implementational*, they opt for a different naming convention with similar meanings. The computation level remains unchanged in convention. It refers to mechanisms such as visual odometry or an internal compass based on the Sun's polarization. A prerequisite for the implementation of computational modules is the neuronal level. Neurons are their building blocks as it were. Certain parts of the insect's brain are related to separate functions. Computational and neuronal give rise to the behavioral level. Most prior research is focused on the behavioral aspects of insects, since biologists find it easier to study an animal's behavior than its learning process [13]. Naturalists have been studying the habits of animals for centuries, although first formalized by scholars such as Émile Zola [14] and Charles Darwin [15]. Even for much of modern human's existence, animals needed to be studied in order to comprehend their behaviors and exploit them to survive.

The three aforementioned levels are linked by models. A model is needed in order to understand the transition from one level to the next. It should inherently be clear and simplified. Complete brain simulations may be considered as the ultimate model of the transition from the neuronal to the behavioral. What can actually be learned from such frameworks? What purpose does a model have when it holds the same degree of complexity as the actual subject it is supposed to study? An effective model only accounts for a subset of the animal's cognition or anatomy. The focus of this thesis lies in the transition from computational modules to behavioral manifestations, which involves the insect's embodiment in concatenation with its environment. The evolved behavior of a certain species is intricately connected with its natural surroundings as well as its previous evolutionary stages. Most robotic implementations, such as the snapshot model and scene familiarity, have been adopted after studying insect's behavior while artificially varying its operating environment. These mechanisms are largely unrelated to the neuronal implementation. In fact, this work shows that the neuronal implementation can be replaced by

more simplified methods such as a convolutional kernels and nodes in an elementary function chain.

2.2. Embodiment and Sensorimotor Integration

How do insects utilize their available means to conduct foraging trips and return to their colonies without diverging from their intended path? Evolution has provided them with a versatile toolkit. Bees, wasps, and ants possess multi-faceted eyes (ommatidea) [4], where one ommatidium is associated with a 1-4°horizontal field of view (FoV) along the horizon [16, 3], but also small photo-receptors (ocelli) located more towards the dorsal region compared to the ommatidea. Both bees and wasps have 2 sets of wings and 3 sets of legs, whereas ants are wingless. Due to their similar anatomy, it is not wrong to assume that they use comparable navigational mechanisms. Biologists believe that these anthropods guide themselves based on path integration (PI), visual or leg-based, and visual interpretation of the moving retinal image [17, 18].

PI involves the integration of the whole outbound foraging trajectory to subsequently return home in a straight line. This mechanism is available to the animal even in environments that are totally barren, in other words lacking any landmarks, or in full darkness (in the case of ants) [17]. It is currently not known whether the dependence on PI differs based on species or on accumulated experience [19]. Ofcourse, counting steps (estimating distance) is insufficient to integrate the full movement, a current compass direction is therefore essential. It is generally accepted that anthropods with ocelli derive their current compass direction based on polarized light from the Sun [4].

Several models of PI have been proposed the last 80 years [1]. Two relevant models are presented here: (1) Mittelstaedt & Mittelstaedt's and (2) Gallistel's model. Both of them utilize the planar motion assumption. One has to take into account that it is convenient to lay out mechanisms such as PI in mathematical terms, but they cannot be confused with the biological mechanism itself. Animals and humans perform approximate computations in their neural structures. First and foremost, several arbitrary variables for an example trajectory are given in Figure 2.1. These include the following: (1) timestep n [-], (2) Cartesian coordinates (X_n, Y_n) [m] of the specimen's location, (3) home vector bearing Φ_n [rad] as seen from home, (4) home vector bearing $\omega_n = \Phi_n - \theta_n \pm \pi$ [rad] as seen from the animal's head orientation, (5) home vector distance D_n [m], (6) step length P_n [m], (7) step direction θ_n [rad], (8) direction change from current to next step $\alpha_n = \theta_{n+1} - \theta_n$ [rad], and (9) angle by which the next step deviates from Φ_n centered on the insect's location $\delta_n = \theta_{n+1} - \Phi_n = \pm\pi - (\omega_n - \alpha_n)$ [rad].

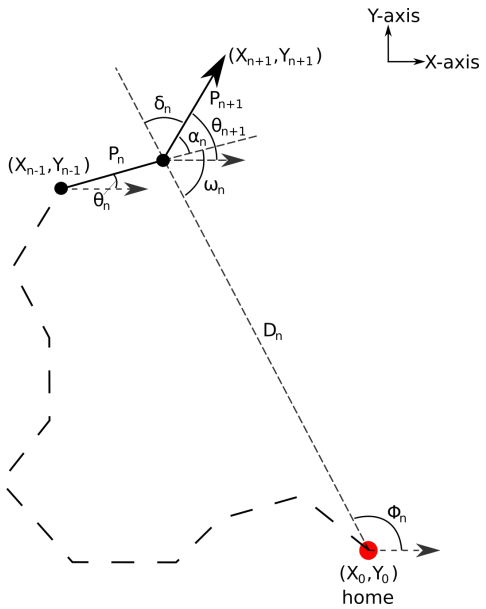


Figure 2.1: Depiction of arbitrary path integration model variables, taken from [1]. Coarse dotted lines depict the trajectory of the specimen up till n-1 timesteps. Full lines show trajectories in between timesteps.

The first model, from Mittelstaedt & Mittelstaedt (Equation (2.1)), is based on Cartesian coordinates only.

It was not possible to be translated into egocentric recurrent formulae, so the geometrical relations of the outbound path are given:

$$\begin{aligned} \Phi_n &= \arctan\left(\frac{Y_n - Y_0}{X_n - X_0}\right) \\ D_n &= \sqrt{(X_n - X_0)^2 + (Y_n - Y_0)^2} \end{aligned} \tag{2.1}$$

The second model that was extracted from the aforementioned paper, Gallistel (Equation (2.2)), consisted of differential equations describing a continuous trajectory of the animal:

$$\begin{aligned} \frac{\mathrm{d}\omega}{\mathrm{d}t} &= -\frac{\mathrm{d}\theta}{\mathrm{d}t} + \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\sin\omega}{D} \\ \frac{\mathrm{d}D}{\mathrm{d}t} &= -\frac{\mathrm{d}s}{\mathrm{d}t}\cos\omega \end{aligned} \tag{2.2}$$

Where t [s] is time, $\frac{\mathrm{d}s}{\mathrm{d}t}$ [m/s] is the specimen's linear speed, and $\frac{\mathrm{d}\theta}{\mathrm{d}t}$ [rad/s] is the angular speed. This set of equations can be rewritten as an egocentric recurrent formula:

$$\begin{aligned} \omega_{n+1} &= (\omega_n - \alpha_n) + \frac{\sin(\omega_n - \alpha_n)P_{n+1}}{D_n} \\ D_{n+1} &= D_n - P_{n+1}\cos(\omega_n - \alpha_n) \end{aligned} \tag{2.3}$$

Insects exploit the directions of Sun polarization (e-vectors), rays scattered by air molecules, in the sky as a directional reference. These e-vectors can be imagined as a pattern of concentric circles around the Sun and can be detected by insects using their ocelli, specialized ommatidea above the compound eye, also called the polarization(POL)-area. POL-neurons, associated with the interpretation of the POL-area input, are tuned to different e-vector directions. A model of a polarized Sunlight compass has been given by [4] in terms of polarization-opponent (POL-OP) units, which approximate the function of POL-neurons. Just like the three ocelli of insects, three POL-sensors are mounted on top of a robotic platform and tuned to e-vector directions at $60°$intervals. The output s (assumed unitless) of a POL-sensor is given as:

$$s(\phi) = KI(1 + d\cos(2\phi - 2\phi_{max})) \tag{2.4}$$

where ϕ [rad] is the current orientation of the sensor with respect to the solar meridian, ϕ_{max} [rad] is the value of ϕ that maximizes s, K [m^2/W] is a constant gain, I_{max} [W/m^2] the maximum intensity, I_{min} [W/m^2] the minimum intensity, $I = I_{max} - I_{min}$ [W/m^2] the total intensity, and d the degree of polarization. Consequently, the output of the POL-OP units is:

$$\begin{aligned} p_1(\phi) &= \log\left(\frac{1 + d\cos(2\phi)}{1 - d\cos(2\phi)}\right) \\ p_2(\phi) &= \log\left(\frac{1 + d\cos(2\phi - 2\pi/3)}{1 - d\cos(2\phi - 2\pi/3)}\right) \\ p_3(\phi) &= \log\left(\frac{1 + d\cos(2\phi - 4\pi/3)}{1 - d\cos(2\phi - 4\pi/3)}\right) \end{aligned} \tag{2.5}$$

where p_1, p_2, p_3 are the outputs of the POL-OP units for $0°$, $60°$, and $120°$respectively. Note that the elevation of the Sun is variable throughout the year and is also dependent on geographic latitude. One hypothesis on how insects cope with these variations is given in the form of an innate ephemeris function [20]. However, it could also be theorized that insects learn the changes in Sun elevation, as a part of their immediate surroundings, during learning flights. Another alternative is given in [21], where it is stated that insects may have an innate preferred angle of polarization.

PI for bees and wasps is conducted differently than in the ant's case. Very little of their journey is tracked using leg odometry, instead visual odometry is used. Visual odometry is based on optic flow, the relative speed at which the retinal image translates and rotates. During forward flight, bees balance the optic flow of both compound eyes in order to fly straight [18]. A course heading is held by minimizing the generation of rotational optic flow on the retina. They gauge the flight distance correctly even in windy situations. This is done without regard for the geometrical structure of the environment. A prerequisite for accurate optic flow estimation is an environment with sufficient texture in its geometry to track the image movement across the retina.

A simple visual odometry model, based on 2D images, is given in [22]. The recurrent camera poses at timesteps $n-1$ and n are computed using a rigid body transformation matrix $T_{n,n-1} \in \mathbb{R}^{4\times4}$:

$$T_{n,n-1} = \begin{bmatrix} R_{n,n-1} & t_{n,n-1} \\ 0 & 1 \end{bmatrix} \tag{2.6}$$

where $R_{n,n-1} \in SO(3)$ is the standardized rotation matrix and $t_{n,n-1} \in \mathbb{R}^{3\times1}$ is the translation vector. The current camera pose $C_n \in \mathbb{R}^{4\times4}$, indicating the series of concatenated transformations with respect to the initial pose C_0, can thus be computed as follows:

$$C_n = C_{n-1}T_{n,n-1} \tag{2.7}$$

For which the first camera pose C_0 needs to be initialized before the recurrent pose computations can be conducted. The visual odometry model is based on the matching of two subsequent images I_n and I_{n-1}. The algorithmic explanation for the modeling effort, if 2D-to-2D feature correspondences are used, is:

1. Capture new image I_n
2. Extract and match feature correspondences of I_n and I_{n-1}
3. Compute geometric relations for image pair
4. Decompose geometric relations into rotation and translation
5. Construct $T_{n,n-1}$
6. Compute relative scale and adjust rotations and translations
7. Compute recurrent camera pose C_n
8. Repeat

The geometric relations between feature correspondences, meaning the same feature shifted between frames, are encoded in the so-called essential matrix E. $R_{n,n-1}$ and $t_{n,n-1}$ can be extracted from E using the solution of the Longuet-Higgins eight-point algorithm. The resulting nine constraints are to be solved using singular value decomposition (SVD). Note that the geometric relations between images can only be determined up until a relative scale, whereas the absolute scale needs to be determined with another method.

2.3. How Bees and Wasps Conduct Their Learning Flights

Literature often mentions that insects perform some form of 'learning' or 'orientation' flights at early stages in their life. These terms are used ambiguously and sometimes it is not clear which phase they are suppose to portray. In this thesis, the whole flight trajectory relating to the learning of the hive's surroundings will be coined the learning flight. In lieu of the nest, a food source could be used as a substitute, while still observing the same learning process. From observations, it can be deduced that the *learning* flight of bees and ground-nesting solitary wasps consists of 2 distinct phases (see Figure 2.2): an *orientation* flight phase, directly followed by an *exploration* flight phase.

The orientation flight is otherwise known as the turn-back-and-look behavior (TBL), check Figure 2.2 (A.1). This part is characterized by the animal's emergence from the nest. The triggering cue, to initiate learning, is a transition from a dark environment, such as the hive, to an (partly) unknown illuminated environment [19]. In this situation, the insect immediately performs a 180°turn to orient her gaze towards the nest entrance and shortly hovers in place [13]. This maneuver is followed by repeatedly backing

away and converging to the nest entrance. Simultaneously, the subject starts turning in a saccadic fashion centered around the entrance, increasing the relative distance between curves (Figure 2.2 (C)). Lehrer and Bianco [2] argue that this phase is related to learning the distance through the optic flow generated from approaching the nest multiple times in succession, possibly making use of motion parallax in the process. TBL is much like remembering a fine grid of landmark positions or features around the nest entrance.

The exploration phase is also known as “circling”, depicted in Figure 2.2 (A.2), which includes faster flight speeds compared to the TBL phase. The circling phenomenon has been mentioned in multiple studies [2, 23, 24, 25], but a trajectory trace remains to be documented since these circles extend beyond human visual range. As a consequence, the schematic in Figure 2.2 (A.2) is an artistic impression of the circling phenomenon, inferred from the author’s own impression of video material and personal anecdotes. This phase consists of circular turns with a linear distance-height relationship. Consequent flight trajectories are cone-shaped, converging onto the nest. Whenever the animal recognizes that she has returned to this imaginary cone, she homes towards the entrance of the colony. This conveniently maps the homing problem from a 3D space into two dimensions once the circling cone has been encountered. The nest entrance itself will not be visible to the insect anymore, therefore she remembers coarse landmark positions.

Currently, it is unknown whether these two learning flight phases are the resultant of discrete or continuous cognitive decisions. In some cases, insect navigation is a consequence of the same continuous cognitive process, e.g. oscillation modulation in olfactory homing [6].

Over time, experienced foragers will exhibit shorter learning flights until they fly straight to the food source and back without performing any TBL or circling behavior, shown in Figure 2.2 part (B). These learning phases can usually only be triggered again whenever the hive is not in the same location upon return or anytime the nest’s surroundings have been changed significantly [26, 2]. In such situations, the insect will conduct a search in the location where she believes the nest location should have been. The time she spends on subsequent orientation flights is dependent on how long she took to perform the nest search in the first place.

Up until now, the same learning mechanism was assumed for bees as for ground-nesting solitary wasps, but there are of course differences between the practices of these two species. The most significant is observed in the amount of times either performs their learning flight. For honeybees and bumblebees alike, the learning flight is carried out over multiple excursions coupled with further discovery of foraging sites. Each excursion is usually focused on a particular direction and they venture much further from their colonies. Wasps stay closer to their homes and conduct only a single learning flight before foraging [27]. It should also be noted that bees fly much higher above the ground while foraging and communicate direction and distance to food sources to their peers through round and waggle dances.

While learning flights are the most valuable for this work, particularly the circling behavior, it is deemed necessary to mention what occurs after the learning flight. Due to advances in harmonic radar, the flight of bees could also be observed beyond human visual range, albeit with limited temporal resolution (patterns related to learning flight are not visible). Studies such as [28, 13] describe learning and exploitation flight trajectories of bees over their lifetime. A flight would be classified as a *discovery* flight whenever the subject would fly over newly discovered regions (discovery flight incorporates learning flight and the search for foraging sites), whereas an *exploitation* flight would be a single loop straight to the foraging grounds and back. Bees released early in the summer season would have ample food nearby the colony and would not venture as far as the ones released at the end of August. Therefore, shorter discovery flights were observed for the former. Exploitation flights were solely conducted at foraging areas that were discovered beforehand. Peculiarly, whenever the exploitation did not result in a yield as favorable as experienced before, the bee would conclude that that area was exhausted and either initiate a new search for a better food supply or switch to an already discovered site. The remembered foraging areas can remain in the bees’ memory for at least 9 days, where most foragers’ lives only last for about 12 days [28].

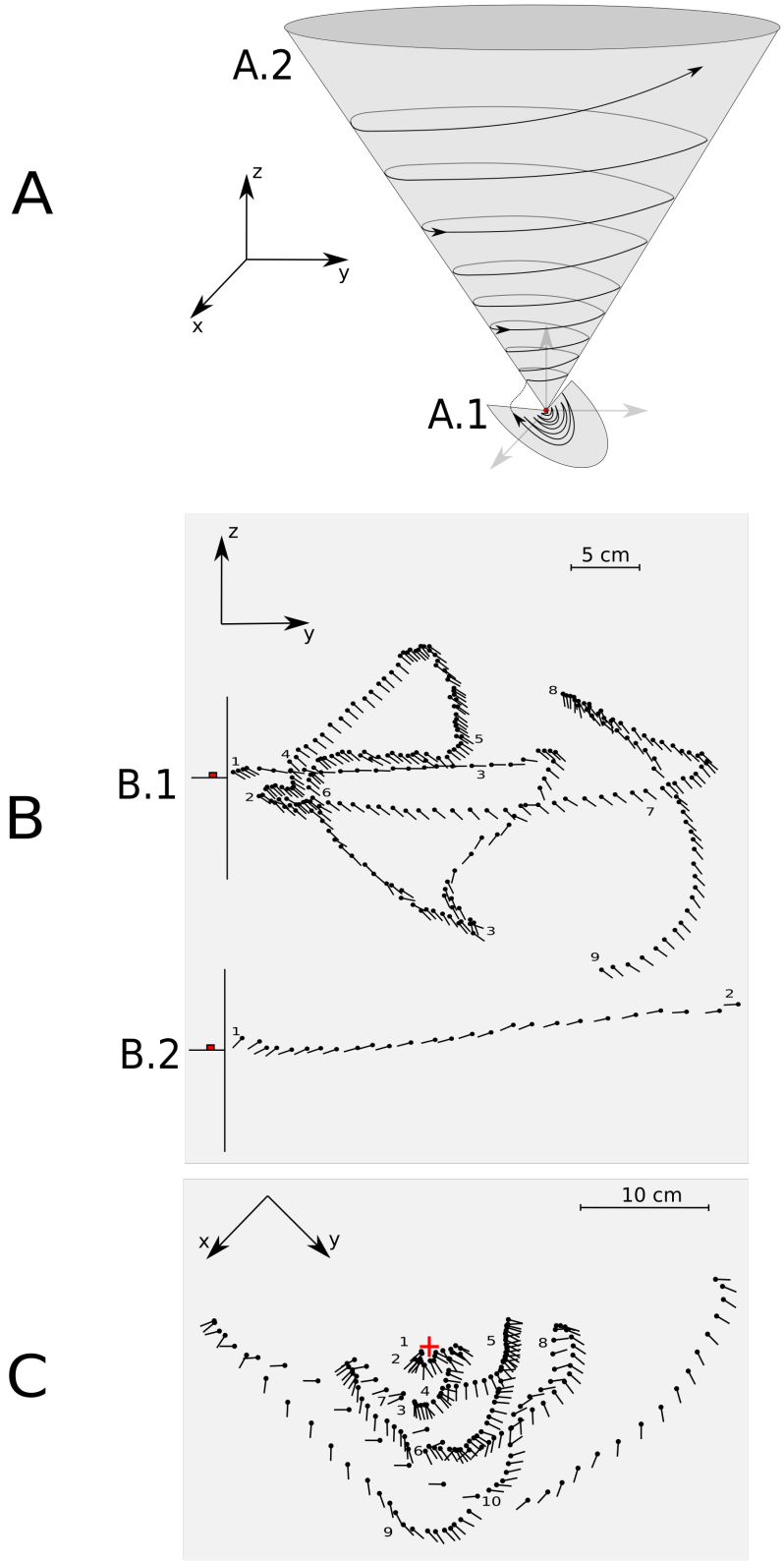


Figure 2.2: Schematic representation of learning flight and TBL time traces. (A) Learning flight is split into 2 components: (A.1) orientation flight or TBL and (A.2) exploration flight or circling. Schematics are an approximation of the observed behavior. Arrows indicate flight direction. (B) TBL time trace, side view, of foraging bees extracted from video recording. (B.1) First TBL of individual bee. (B.2) Departure when TBL is no longer required. (C) TBL time trace, top view, of ground-nesting solitary wasp (*Cerceris*) extracted from video recording. Circles and dashes represent head and body respectively. Numbers are chronologically ordered according to flight trajectory. The nest is indicated by a red box/cross. (B) and (C) are taken from [2].

It is evident that during the learning flight, the flying insect remembers home vectors (acquired during learning trips). The direction and distance to home is the simplest set of information that would result in a successful return. Like discussed, insects keep track of their home vector using PI and remembered visual cues. Whenever the integrated home vector is utilized to return to the hive location, the vector magnitude (distance) is followed until null. In this case, the subject may find herself not exactly at the nest entrance. Her location could have drifted somewhat due to the dead reckoning nature of path integration mechanisms. Until the visual or olfactory recognition of the colony, she must (hypothetically) use her visual memories of the surroundings to generate a supplementary home vector. It is the author's opinion that such a supplementary home vector, constructed from neurally encoded memories, would be the most parsimonious solution that can be conceptualized and that guarantees a safe return home.

The findings of this chapter will be combined with insect-inspired robotic approaches found in literature resulting in a new navigation algorithm that mimics the way insects conduct their learning flights. An attempt has been made to approach the effectiveness and efficiency of circling behavior in honey bees and ground-nesting solitary wasps in particular. For some, it may be irrelevant to study biology in order to come up with new ideas for navigational algorithms. Simply looking at the stringent memory and computational limits of the navigating insect's brain should be enough to change the sceptic's mind. Bees and wasps navigate long-range over often ambiguous or barren terrain without any external aids. Understanding the underlying mechanisms of these creatures allows the creation of parsimonious algorithms, with the least resources possible, exhibiting complex behavior.

3

Insect-Inspired Robotics

By studying the mechanisms and behavior of animals, researchers in the field of bio-inspired robotics aim to create robots that can mimic the natural world in its form and function. To bridge the world of biology and robotics, several design options are to be evaluated (see Section 3.1). Like in Chapter 2, the multitude of research papers needs to be categorized, shown in Section 3.2, to track what has been achieved so far. Navigational strategies can be subdivided into two categories, namely local navigation and way-finding algorithms.

3.1. From Biological Observations to Navigational Algorithms

In Chapter 2 several behaviors of bees and ground-nesting solitary wasps have been discussed. Now the question arises how to implement this knowledge in a robotic design solution such that the ideal result is a program with similar power, performance, and memory consumption as the real insects. The primary element for insect intelligence is parsimony, that is, simple yet robust mechanisms that achieve successful behavior in real world environments [8]. For a solution to be parsimonious, three aspects need to be taken into account: (1) embodiment, the body and sensor suite; (2) sensory-motor coordination, the concurrent coordination of muscles and sensors; and (3) swarming, social interactions. Studying these aspects for a specific species creates a potential to design systems that were not possible in the past, e.g. removal of active actuation of the wing angle of attack achieved lift greater than weight at insect scale. Although, studying the insect's behavior and classifying it according to its idiothetic and allothetic means has the advantage of direct implementation in robotics, like the case of passive wing angle. It becomes more challenging whenever more complex behaviors are to be implemented. To translate the observations from nature into something tangible, three design principles are proposed that give the designer a general framework to work out their ideas:

- Manual design
    1. Divide-and-conquer strategy: split problem into submodules that are developed independently without exploiting sensory-motor interactions
    2. Biologically inspired strategy: inspiration from animal that excels at a particular conduct knowing its sensory inputs and motory outputs
- Automatic design
    3. Machine learning: automatic design by implementation of a general model

Every choice made by the designer leads to implicit constraints. Each of these design principles has their advantages and disadvantages. As apparent from Chapter 2, this work focuses on a design that is biologically inspired. Manually designing the connection between in- and outputs of a biological analog that excels at navigation, namely the honey bee (*Cataglyphis*). Observations relating to the ground-nesting solitary wasp (*Cerceris*) are also taken into account, since her behavior is very similar to that of the former in the vicinity of the nest during learning flights. The difficulty of this design method is that knowledge gaps need to be filled in by the designer's choices. Simulation has helped to map neurons

relating to certain computational modules of the animal, such as the polarized (POL) light compass. Such studies help in hinting how the animal goes about its daily routine, but in practice it is often quite a challenge to fill in the knowledge gaps.

Approaches that have been successful in this regard include the snapshot model and scene familiarity, discussed in Section 3.2.1 and Section 3.2.2 respectively. They are built on the same paradigm of hypotheses. Hypotheses of what the underlying computational model for navigation should look like. They function much like a data-driven approach to understanding a 'black box' system. Whenever the animal's conduct is not known explicitly, the designer gives comments on its actions. Whereafter, the ideas are distilled into a framework that exhibits similar performance as observed in nature. The model's compliance is tested in simulation as well as robotic implementation. Then, it is adjusted slightly anytime discrepancies are detected. If it 'fits' the insect's behavior sufficiently, meaning: displaying the same output states and being accepted by enough researchers in the community, the model gains viability and one can be more confident that the proposed ideas are representative for the study subject.

3.2. Different Types of Navigational Algorithms

A classification method for existing navigational algorithms is desirable. This way, it can be identified what the state of the current research on bio-inspired navigation is and which aspects can be improved. Past, and to a certain extent present, navigational efforts hail from nautical applications [20]. Terms such as localization, mapping (on charts), bearings, and dead reckoning were coined in an age where seafaring was the most prominent form of long-range transport. The lexicon has been copied in the field of robotics, which cannot be said about the systems being applied to the problem of navigation. None of the previously engineered way-finding solutions remotely comes close to the flexibility and performance that is observed in insects, such as bees and ants.

While studying human navigation, the debate is usually focused on the question "Where am I?" in order to answer "Where do I need to go?". It is often desirable to mentally visualize concepts as humans, but it must be considered that a bias starts to develop whenever concepts are analyzed in this manner. Humans analyze problems starting from their own reference point. It is hard to imagine other forms of cognition without taking into account our own. Insects provide proof that the question of "Where to go?" can be answered without ever knowing one's explicit ego-location. Their limited neuronal capacity is simply not capable of storing any long-range trajectories. An insect must exploit her limited resources intelligently before she can learn to navigate the real world successfully.

So, how does an insect go about finding her way without a map-like representation of her environment? The elements along her route, especially around the home and goal locations, should be encoded in some form of long-term memory. How the memories are being utilized for the purpose of finding a way back home is currently a matter of speculation. Different algorithms have been developed with these memory prerequisites in mind, they can be categorized in the following overview:

- Local navigation
    1. Search: no active orientation towards the goal
    2. Direction-following and Path integration: aligning course with locally available direction
    3. Aiming: body axis is oriented such that the goal is in front of it, the goal is associated with salient cue
    4. Guidance: exploiting spatial configuration of objects around the goal, the goal is not associated with salient cue
- Way-finding
    1. Recognition-triggered response: connecting two locations by means of a local navigation method, the spatial representation is goal dependent
    2. Topological navigation: routes are generated independently of each other and each goal needs its separate route (possibly integrated in other routes), the spatial representation is goal independent
    3. Survey navigation: embedding of all known locations and their spatial relations into a common frame of reference

At the top level of the overview, we have local navigation techniques and way-finding. In essence, local navigation always deals with a single goal location with the help of locally available information. Way-finding, on the other hand, either uses coupled local navigation methods, simple bug algorithms [29], or some form of pre-existing map of the environment to connect two locations. The rest of the literature study will mostly deal with local navigation methods. Particularly, guidance will be considered in subsequent sections, since the goal of an insect is to return to a previously visited goal location using the surrounding landmarks. This does not mean that the mentioned local navigation methods could not be used to conduct way-finding. One of the more popular biologically inspired approaches to way-finding is coupling these relatively simple local navigation techniques to achieve recognition-triggered responses as demonstrated quite successfully in [30]. Insects evolved in such a way that they reverberate with their environment [6], making a complete topological representation unnecessarily complicated for their foraging tasks. Recognition-triggered navigation seems the only viable way-finding option for insects due to their limited memory capacity and inability to store complete maps of routes and surroundings.

3.2.1. Local Navigation Based on Snapshot Model

As mentioned in Section 3.2, guidance is the main topic of interest regarding visually inferred bearings as well as all the navigational methods that inspired the idea of learning the home bearings in the first place. The snapshot model [3] and the familiarity concept [31] are the most influential insect navigational models to date. Their underlying principles will be explained in the current and following section (Section 3.2.2) respectively. The snapshot concept (shown in Figure 3.1 (A)) emanated from a series of feeding experiments where bees were trained on a single landmark or an array of landmarks in different configurations. After training, the bees were evaluated on a different configuration of landmarks. From the results, Collett and Cartwright concluded that bees are primarily concerned with the landmark's apparent size on the retina as well as its relative bearing. In the paper, it is suggested that bees make a mental snapshot of the surrounding objects and terrain at the goal location. The snapshot is then continuously compared to the observed scene on the retina until a proper 'fit' is achieved. Five iterations of their model were tested with varying degrees of complexity, ranging from only comparing the nearest edges to matching the landmark areas. Snapshot couples were assumed to be acquired in the same orientation, stabilized using an external compass cue of some sort. All models determined flight direction from apparent size and observed bearing.

The homing vector ensuing from the formal description of the snapshot model is comprised of radial and tangential components, which are related to the apparent size and bearing difference of matched landmarks respectively. Assuming that the specimen's reference frame is corresponding with the global reference frame, by means of compass stabilization, the positions of n landmarks in the two-dimensional plane are given by $\overline{\mathbf{x_i}}$ with $i = 0, ..., n$ [32]. Whereas the position of their edges, as seen from the insect's current position, are defined as $\overline{\mathbf{e_i}}$ with $i = 0, ..., n$. Then, from an animal's position $\overline{\mathbf{x}}$ in the plane, the sweep angle $\theta_i(\overline{\mathbf{x}})$ (footprint of landmark on horizontal FoV) follows from the dot product of the landmark's centroid and edge vectors:

$$\theta_i(\overline{\mathbf{x}}) = 2\cos^{-1}\left(\frac{(\overline{\mathbf{x_i}} - \overline{\mathbf{x}}) \cdot (\overline{\mathbf{e_i}} - \overline{\mathbf{x}})}{\|\overline{\mathbf{x_i}} - \overline{\mathbf{x}}\| \cdot \|\overline{\mathbf{e_i}} - \overline{\mathbf{x}}\|}\right) \quad \text{with} \quad 0 \leq \theta_i \leq \pi \tag{3.1}$$

The sweep angle $\theta_i(\overline{\mathbf{x}})$ is related to the radial component of the home vector, whereas the landmark vector $\overline{\mathbf{L_i}}(\overline{\mathbf{x}})$ with unit length, pointing to the centroid of landmark i, is associated with the tangential home vector component:

$$\overline{\mathbf{L_i}}(\overline{\mathbf{x}}) = \frac{\overline{\mathbf{x_i}} - \overline{\mathbf{x}}}{\|\overline{\mathbf{x_i}} - \overline{\mathbf{x}}\|} \tag{3.2}$$

Given a goal position $\overline{\mathbf{x_0}}$, the magnitude of the home vector $\overline{\mathbf{H}}(\overline{\mathbf{x}})$ is the average of the scaled differences of the currently experienced sweep angles and the remembered snapshot sweep angles $\theta_i(\overline{\mathbf{x_0}})$:

$$\left\|\overline{\mathbf{H}}(\overline{\mathbf{x}})\right\| = \frac{s}{n}\sum_{i=1}^{n}(\theta_i(\overline{\mathbf{x}}) - \theta_i(\overline{\mathbf{x_0}})) \tag{3.3}$$

where s is an arbitrary scaling factor. The proportional vector (PV) model (seen in Figure 3.1 (B)) is a snapshot equivalent which does include the home vector's magnitude $\left\|\overline{\mathbf{H}}(\overline{\mathbf{x}})\right\|$. It should be noted that

in Equation (3.3) the computation is theoretical, resulting in a perfect match between the landmarks. In practice the light and dark regions would be matched according to their closest neighbor with the same sign. Also, the magnitude of the home vector is neglected in the original snapshot model [4]. Instead, it is represented by its unit vector $\hat{\overline{\mathbf{H}}}(\overline{\mathbf{x}})$, which can be obtained by taking the difference of the landmark vector averages (see Equation (3.4)).

$$\hat{\overline{\mathbf{H}}}(\overline{\mathbf{x}}) = \frac{1}{n}\sum_{i=1}^{n}\left(\overline{\mathbf{L_i}}(\overline{\mathbf{x}}) - \overline{\mathbf{L_i}}(\overline{\mathbf{x_0}})\right) \tag{3.4}$$

These modeling efforts result in viable behavior in simulation, but have some disadvantages as well. At the outset, the implementation does not result in fully realistic behavior since the acquired view is never in the same orientation, mental transformations would be peculiar considering the inherent maneuverability of an insect. A bee's horizontal FoV is also only about $270°$[33, 34] compared to the full omni-directional view often used in relevant simulations and robotic experiments. Also, there is no condition present that indicates whether the bee has actually reached the location of her goal. The last snapshot model performs poorly in situations where it encounters some noise, e.g. a configuration where only two instead of three landmarks are portrayed, or when distant landmarks remain unfiltered.

The most prominent inconsistency would be that of memory scarcity. Assuming that: (1) the whole bee's neural structure, $1 \cdot 10^6$ neurons, is related to solely remembering the snapshots of the environment; (2) The bee's brain contains $1 \cdot 10^9$ synapses, taken from an unpublished paper [35]; (3) one synapse stores 4.7 bits [36]; (4) visual sensing is represented by a $270°$x $90°$image with $1°$acuity [33, 34, 3]; (5) one conventional digital RGB pixel takes an equivalent storage space of 24 bits, 8 bits per color channel; (6) the bee only moves in a two-dimensional grid; and (7) the catchment area is equivalent to 1 m^2 [37]. Where the catchment area is a region, encompassing the snapshot position, that confines the operating capability of visual homing. If an attempt is made to make a robotic bee with these lenient constraints, the total storage capacity would be $4.7 \cdot 10^9$ bits or 560.3 MB. While the storage requirement for a single image would result in $538,200$ bits or 71.2 kB. This would result in an available storage volume of 8058 full images in the cranal library, which would cover a total rectangular area of 7879 m^2 or a maximum distance (diagonal of rectangular area) of 125.5 m from the starting location. Ofcourse, biological memory capacity and ommatidea clusters' FoV cannot be mapped in digital entities. Nevertheless, taking into account the non-representative nature of this thought experiment, it does demonstrate the intuitive conclusion that the pure snapshot model seems to partly misrepresent the bee's memory capacity.

The snapshot model remains a popular representation of insect navigation, even with its aforementioned limitations, which is strongly rooted in the research community [26, 11, 7, 16]. Most of the research that is inspired on the snapshot model (template-based homing) attempts to overcome the restrictions of the original model. van Dijk [30] shows that memory scarcity is only a problem if the snapshots are stored in their original forms. In his thesis, the snapshots are stored in the form of a select amount of Fourier components [38]. His algorithm was tested in simulation and robotic implementation on 63 m flight trajectories. It only consumed 17.5 bytes/m, orders of magnitude smaller than the memory requirements in the presumed thought experiment. Variations on the snapshot model strongly implicate that raw snapshots are not stored explicitly in nature. Instead, the snapshot is assumed encoded in another representation, which is much scarcer memory-wise without losing its intended purpose. The average landmark vector (ALV) [4], presented in Figure 3.1 (C), takes only the rotational contribution of the landmark differences $\overline{\mathbf{L_i}}(\overline{\mathbf{x}})$ into account. The home vector is then the resultant of the difference of the landmark vector at the goal location and at the insect's current location. It has the advantage of only having to calculate and store vectors. Although it solves the main complication of the snapshot model, some disadvantages are apparent as well. Whenever a landmark array is encountered with a similar structure and orientation as observed at the goal location, the specimen gets stuck in local solutions instead of searching for the global destination. Also, getting close to landmarks or the presence of occluded landmarks may result in an ambiguous home vector and unwanted behavior.

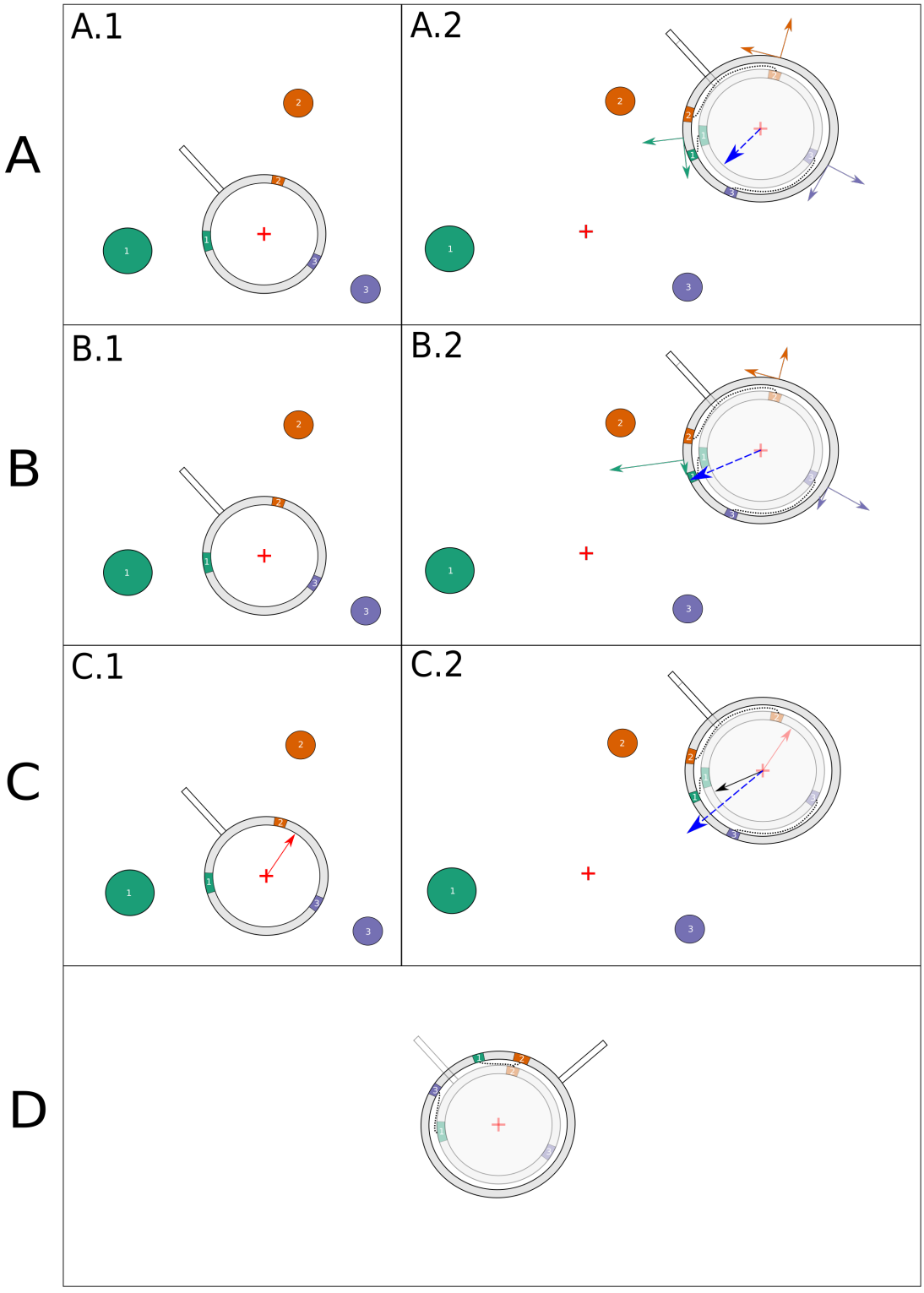


Figure 3.1: The snapshot model and its derivations. (A) The original snapshot model, home vector contributions are unit length. (B) The proportional vector model, home vector contributions are proportional. (C) The average landmark vector model, only rotational home vector contribution. For (A), (B), and (C): (*.1) horizontal view at goal position and (*.2) horizontal view at subject's current position. Nest position is indicated by the red cross. Teal, orange, and purple circles are landmarks (numbered). Teal, orange, and purple arrows are home vector contributions. Blue dashed arrows are home vectors. Black and red arrows are the average landmark vectors. (D) Horizontal views taken at different reference frames. Teal, orange, and purple circle areas are projections of the landmarks (numbered), not always to scale due to concentric view. Snapshot circles and rectangular boxes represent the subject's head and body respectively. Adapted from [3] and [4].

For completeness, Figure 3.1 (D) shows what happens whenever the shared reference frame assumption does not hold in snapshot-derived models. The reader can observe that the goal snapshot is taken in a different orientation than the current view. This results in an undesirable match between landmark and background regions, as indicated by the dotted lines.

3.2.2. Local Navigation through Scene Familiarity

It can be concluded that in order to compress the memory required for a snapshot, it needs to be stored in a different form. Such an encoding could be in the form of an ALV, but it could also be more abstract. The familiarity measure [31] does not have a physical meaning, it can better be interpreted as a feeling or some kind of notion (see example situation in first paragraph of Chapter 1). Instead of explicitly storing the snapshot view, it is run through an artifical neural network (ANN) in several training episodes. The network's output node encodes the raw image pixels to an abstract familiarity value, e.g. 1 for most and 0 for least familiar. When all training is completed, previously seen or unseen images can be provided to the network and an associated familiarity measure is provided. If a current view is not that familiar to the specimen, it can rotate until a more familiar one is encountered (search). This results in distinct stop-scan-go movements, which are often observed in ants. Baddeley et al. [31] argue that their scene familiarity approach solves the drawback of the snapshot model that it is only viable in the immediate vicinity of the goal snapshot. Their main hypothesis states that if the most familiar views are followed at all times, successful homing is achieved.

The insect maintains a secure connection to the nest through her PI abilities even when she departs from it. It is suggested that since bees and ants tend to align their body axes with their translations during foraging, the central point of their current view can be considered as their overall direction of movement. Artificial learning flights are carried out to gather training data for the ANN, taking into consideration the foraging routes. This data is fed to the ANN in the form of omni-directional images, such that the result is a holistic mental route representation encoded in its weights. Multiple runs make it possible to learn broader, more robust route corridors.

When distant landmarks are blocked from view, it is more likely for the subject to experience difficulty in returning home (less of a problem for flying insects). Stop-scan-go behavior has been observed in ants, but is seen a lot less frequent in nature than needed for scene familiarity route following. Most importantly, the familiarity associated with the current view does not tell the specimen whether she should rotate or not. Florent Le Möel and Antoine Wystrach propose an alternative to performing mental or physical rotations for familiarity-based navigation: attraction/repulsion scene familiarity [10] inspired by goal and anti-goal oscillations observed in ants (*Myrmecia*). Instead of only relying on the classic familiarity measure (attraction) associated with the goal direction, a repulsive measure linked with the anti-goal direction is included as well. After learning, the resultant MBON (mushroom body output neurons) signals, triggered by a newly experienced view, are integrated to form an angular error to the most familiar view direction. This angular error would be defined with respect to the insect's longitudinal body axis if a planar environment is assumed.

The sampling technique, for training purposes, includes a commonly observed phenomenon in foraging instances, that of oscillatory paths. The authors argue that these oscillations are a resultant of familiarity-based amplitude modulations. The attraction/repulsion approach would inherently be robust against illumination changes, since the ratio of attractive/repulsive views is taken into account as opposed to any absolute measure. Also, no goal condition should be defined because both measures should balance each other out at the goal location.

During route tracking, the subject may get confused between familiarity measures of different goals as no distinct difference is being made between views at different locations. She has no available means to switch to subsequent waypoints without considerable overlap of the catchment areas. Local familiarity loops or measure-balancing may start to occur in between way-point locations. Also, the drawback of a vanishing familiarity count is still present. If the specimen should leave a familiar corridor, there is no fail-safe that would return her within bounds. Otherwise, when only considering local visual homing around the nest entrance, this model seems to be the most successful out of all the ones discussed in this work. In retrospect, one can argue: Why go through the trouble of inferring two abstract measures from a

single image at all? Consequently, they will need to be integrated into a relative angular difference, which on its turn indicates the home vector. Instead, the same result can be obtained in a much simpler way by learning the home vector bearing directly from the input image (see Section 3.3).

3.2.3. Way-finding Strategies

Several local navigation algorithms have been established in this section. Some of them, like the ALV model, inherently include some form of obstacle avoidance capabilities, while most of the considered algorithms do not take into account objects at all. How does an insect utilizing such approaches go about not bumping into landmarks along its route? A simple, bio-inspired strategy is to implement one of many available bug algorithms (BAs) [29]. All local navigation is ultimately concerned with computing a home vector. This home vector gives a straight line indication of some direction that needs to be adhered to in order to safely arrive at the hive. Straightforward BAs provide a convenient strategy to locally alter a trajectory, based solely on a home vector direction and some object detection mechanism.

The most simple algorithm that can be imagined is a random-walker, which is really only useful for comparing other algorithms to. As a step up from the random-walker, the wall-follower can be identified. This approach does what it advertises, it follows walls until a certain goal condition is reached. A prerequisite for the wall-follower bug is that the subject should be in 'contact' with the wall, either physically or through a sensor reading. A small overview giving an example of other BAs (note: each algorithm is an extension of a lower level one, category-wise):

- Angle-bugs: using azimuth angle towards the goal whenever possible
    1. COM: follows outline of obstacle from point of contact (hit-point) until home vector is obstacle-free
    2. COM1: remembers previous hit-point's distance to goal and initiates the home vector following earlier, based on the closest distance to goal
- M-line-bugs: using straight line between start and goal locations
    1. Bug2: follows obstacle's boundary until M-line is encountered
    2. Alg1: reverse direction of obstacle boundary following when previous (remembered) hit-point is encountered
- Range-bugs: using range sensors in one way or another
    1. FuzzyBug: angle/range hybrid which follows closest obstacle's boundary
    2. TangentBug: angle/range hybrid takes shortest tangent of current position and obstacle border

Whenever the local navigation algorithm does not inherently avoid obstacles, BAs need to be utilized in order to perform recognition-triggered way-finding. Other, more advanced, algorithms take into account more information of the environment in the form of topological or metric maps. These maps are usually constructed in concatenation with localizing the specimen's current location, enter the realm of simultaneous localization and mapping (SLAM). The robotics community has a profound interest in SLAM-based approaches. These mechanisms form the ultimate source of information from the environment and are usually feature rich, maintaining a large amount of information. SLAM map construction is a computationally intensive process that is challenging to implement on relatively simple embedded processors.

The details of SLAM-based algorithms are not discussed here, but a lot can be learned from studying insect's learning flights with SLAM [39]. In the paper, the authors concluded that bumble bees presumably do not explicitly structure their learning flights to learn about positions of landmarks. It is also noted that the saccadic turning during TBL maneuvering would serve to estimate distance through motion parallax. It remains doubtful whether this would actually be the case, but some biologist seem to keep it as a viable hypothesis for insect depth estimation. This hypothesis was put to the test in a series of SLAM-based experiments mimicking the flight maneuvers that were observed in bumble bees. It could be perceived that during the experiments, the uncertainty of the positional estimate of the artificial bee would increase, while the positional estimate of the landmarks became more certain. Uncertainty of the

landmark position could be considered as a viable measure of how well the bee would be able to retain information of the landmarks in her surroundings. A bias in learning flights towards the area of the nest was observed, but the hypothesis that bees would focus more on the particular positions of landmarks was rejected. These results bolster the notion that bees rely solely on local navigation tactics and lack the computational capability to explicitly map their surroundings.

3.3. Identifying the Research Gap

This chapter discussed a range of visual homing mechanisms and algorithms that were successfully implemented in real world experiments. The hypothesis of template-based insect navigation was central. Snapshot related approaches provide very viable points on why insect would use a template-based approach to conduct visual homing. All this research shows the general notion that roboticists have a drive to give back to the biology community for their excellent observational work over the years. In a way, implementing hypotheses in robots enables another layer of understanding in the behavior of insects. Modern robots are computationally much more capable than their organic counterparts, but the shortage of smart algorithmic implementations keeps them from being as impressive as bees, wasps, or ants. One can only hope that the community continues to make massive strides forward in understanding the navigational wonders of these tiny animals. This thesis aims to contribute its tiny part in this global effort.

Template-based homing such as snapshot-based algorithms have always been very attractive to researchers [16], even though the neurons related to vision and navigational functions can arguably be seen as a parameter-based structure. From this reasoning and the fact that a convolutional neural network (CNN) is inspired by the way animals handle visual data through their visual cortex, it can be concluded that a parametric model would lean closer to neural computations of insect brains. That would make template-based models ideal to understand what the underlying concepts are for visual homing, but less representative for robust, realistic implementations.

CNNs have been deployed for a variety of navigational purposes in the past, including: odometry, localization, path-following, and homing tasks [40, 41, 42, 43, 44], unexpectedly never with the intention of estimating the home vector directly. A CNN-based visual homing approach would only need a database containing images labeled with the relative bearing, between subject's gaze direction and the nest location, to train the net's weights. In this case, the gaze direction is represented by the middle of the camera's FoV. Acquired images can be discarded once they are used for training (or acquired on-the-fly) and labeling can be based on visual odometry. There is no need for landmark segmentation, taking into account the goal view, prior knowledge of the animal's surroundings, nor knowing her explicit location. Learning flights could be conducted in the vicinity of the nest to keep the odometry drift low, while drift corrections could be performed by periodically returning to the nest entrance.

Hafner [11] was the first to show that learning the home vector is an efficient, parsimonious way of guiding an insect to a target location. As an extension, an attempt to generate similar outcomes with more modern neural architectures, simulation tools, and a slightly different input is proposed. This work presents the first visual homing algorithm that learns the home vector orientation directly from a single raw omni-directional image (based on all considered literature). To suppress the labor and cost associated with data acquisition through numerous real world flights, a set of simulation environments was developed. The specimen can acquire its training data in an online or offline fashion. The underlying idea is to retrain its neural representation for every newly encountered environment, analogous to bees that emerge from the hive for the first time.

4

Literature Synthesis

Biological observations

Odometry	[17] → evolve neural model,	[1] → path integration models
Visual Odometry	[18] → optic flow	
Ontogeny	[28] → bumble bee, [21] → neural circuits, [19] → neurobiological mechanisms,	[13] → honey bee, [6] → emergence of decisions, [45] → desert ant
Learning	[26] → learning flights/walks	
Other	[12] → multi-level understanding	

Robotic implementations

Bug Algorithms	[29] → comparison	
Visual Odometry	[46] → review, [47] → tutorial,	[22] → tutorial, [40] → end-to-end RCNN
Place Recognition	[48] → SNN ensembles, [49] → guided locality preserving	[7] → neural cognitive map,
Visual Homing	[11] → learning home vector, [4] → average landmark vector, [50] → lyapunov equation, [51] → angle-based, [3] → snapshot, [16] → adaptive homing	[31] → scene familiarity, [38] → snapshot Fourier transform, [30] → snapshot low-memory, [10] → attraction/repulsion familiarity, [52] → visual compass,
Olfactory Homing	[53] → gas source localization	
Localization and Mapping	[54] → hippocampal SLAM model, [55] → holistic vs features, [41] → CNN localization	[56] → localization, [57] → visual compass,
Visual Navigation	[42] → RGBD images (target), [58] → image to action (path),	[43] → RGB images (target), [44] → collision avoidance (path)
Other	[20] → review insect-inspired navigation, [59] → $360°$image CNN	[5] → omnidirectional image generation,

Table 4.1: Overview of literature relating to how biologists and roboticists interpret insect intelligence.

Starting from the overview given in Table 4.1, each work is briefly examined to provide the reader with a summary of the literary works covered. Like the two previous chapters, chapter 2 and chapter 3, papers are split up according to two categories: biological observations (section 4.1) and robotic implementations (section 4.2). If the article includes any robotic experimentation whatsoever, it is classified in the latter category. Note that these lenient constraints result in a lot of overlap, due to the fact that most robotic works also include many references to biological studies or contain experiments with insects. After stipulating the interpretations of biologists and roboticists on how insects navigate intelligently, the CNN bearing inference experiments are to be presented in Part III.

4.1. Biological Observations

Odometry
All organic life is the result of an evolutionary, survival-of-the-fittest, process. Genetic algorithms are used to evolve a novel neural model for path integration in [17], where the amount of neurons and their connections are varied in order to create a population encoding model that integrates the direction and distance of the performed motion. Path integration is the integration of translations and rotations in order to keep an up-to-date home vector in memory. Odometry in itself is usually only referring to translatory movement without any compass information. Several mathematical models of path integration are compared in [1].

Visual Odometry
Experiments of bees flying in tunnels [18] give clues on how their visually driven odometer works and how it is utilized for flight control. Balancing the speed in both retinal images to traverse the tunnel, slowing down when the tunnel narrows, and steadily decreasing forward speed whenever altitude is decreased for smooth touchdowns.

Ontogeny
To understand insect behavior, their everyday life needs to be studied in detail. The flight of honey bees and bumble bees are examined by [28] and [13] using harmonic radar and antennae glued to the dorsal region of the animal. As such, the foraging flights could be tracked in the natural habitat over a full lifetime. Embodiment of desert ants, which exhibit navigation that is less dependent on pheromones than their forestal counterparts, and sensor usage are layed out in [45]. In [21] the neural aspect of insects were inspected further, where neural models showed how the animal's current and desired headings were represented in the brain. The discrepancy between these two was then translated and mapped to motor commands. This paper showed diagrams of the latest findings regarding the desert locust, Monarch butterfly, bees, and cockroaches. A look into the mushroom body neural structure of desert ants, as well as a schematic of the whole brain, is given [19]. The mushroom body is related to navigational memories and computation. Insect decision making [6] is not deemed a conscious process. A reverberation with the environment is assumed that intrinsically couples their movements to their surroundings. Different observations should not be seen as resultant from discrete decisions, but rather as a holistic process, e.g. oscillating behavior of larvae that seamlessly flows into homing due to changes in odour gradients.

Learning
An insect does not have innate instructions on how to navigate its environment [26], instead it has the capability to learn its surroundings and behave accordingly. Learning flights and walks are the main tool the animal uses in order to return to shelter and food. Bearings and distances to important locations are encoded in the neural structure to be compared to the currently experienced views. These encodings are learned through an intricate sequence of (flight) maneuvers.

Other
Frameworks are handy to discretize literature findings, such that they can be grouped on higher abstraction levels. In Table 4.1 literature is organized according to the dominant functions it portrays. Alternatively, psychology [12] can help to contemplate what has been learned and what still needs to be understood in the realm of insect intelligence.

4.2. Robotic Implementations

Bug Algorithms

A wide variety of bug algorithms [29] provides simple path solvers without an explicit topological or metric map of the environment. Angle-bugs, range-bugs, and M-line bugs show that obstacles can be circumnavigated without much explicit knowledge about their appearance. In essence, these algorithms are more practical in computation and memory restricted agents.

Visual Odometry

In [46] several state-of-the-art visual odometry techniques are layed out, categorized according to the sensor readings they rely on. The computations as well as historical notes regarding visual odometry have been excellently described in a tutorial series [22, 47]. Odometry is an essential guiding mechanism for insects in their navigation. Usually, learning flights/walks start out fully relying on odometry to home towards the nest. Over the lifetime of the agent, the odometric means are gradually replaced by visual memories to guide their way to goal positions. Instead of relying on explicit computations based on feature extraction and outlier detection, generic algorithmic means, such as the implementation of a recurrent convolutional neural network (RCNN) [40], could also be implemented. The RCNN takes an image as input and computes the relative camera positions as output, providing an end-to-end learning framework. All computations are implicitly encoded in the network's parameters. Odometry remains a dead reckoning device for keeping track of integrated movement, making it prone to drift. External aids, e.g. GPS, are usually employed to correct this drift.

Place Recognition

Place recognition aims to match the currently experienced views with views that are stored in memory (not necessarily snapshot views). This way, the current location of the agent can be inferred based on stored visuals alone. Just like neurons in the brain, spiking neural network (SNN) models contain nodes that encode their current activation as internal state. The communication layer of these nodes is facilitated by spikes whenever a stimulus exceeds a pre-determined threshold. Ensembles of SNNs were implemented in [48] to perform place recognition tasks. Inspired by place cells, areas in the brain associated with physical locations, a cognitive map could be made of all the experienced places [7]. Nodes in this map were interconnected according to the location they show activity for. In this way, metric information could be inferred from the cognitive map without explicitly remembering the geometric structure of the environment. A similar idea with a very different implementation was presented in [49], where homing vectors were extracted from the optic flow's focus of expansion. Outliers in the optic flow computation were eliminated by retaining the local geometric patterns of the location where the calculation was performed.

Visual Homing

A big interest of insect intelligence is the homing behavior these animals exhibit. No other organisms can home to their colonies or forage sites with comparable parsimonious solutions, often relying solely on visual means. The first to implement this fact [3], created a conceptual framework based on the capabilities of bees. Due to the difficulty in mapping their neural structure, it remains unsure how homing is performed by these foragers. One can only provide an hypothesis, verify, and compare it to the natural behavior to infer whether the hypothesis is viable or null. The snapshot model states that bees capture landmark images during their learning flights. These snapshots are related to key locations such as the hive or a food source. Whenever the animal is close to the desired location, its current retinal image is compared to the remembered one to ascertain that it arrived or consequent actions need to be taken. Early snapshot theory was based on the detection of landmarks as a priori.

Several snapshot spin-offs followed that initial article. Discrepancies in landmark unit vectors (apparent size and bearing) were replaced by scaled vectors in the proportional vector (PV) model and only landmark bearing changes were taken into account with average landmark vector (ALV) models [4]. Fourier components provide a sparse memory representation of raw snapshot images [38] while not having to align the snapshots to corresponding reference frames, since the Fourier transform is indifferent to rotations. Even more memory efficient implementations resulted from also taking into account odometry and the catchment area of Fourier transformed snapshots [30].

The most related work for this thesis is given by Hafner [11]. She implemented a snapshot-inspired approach, where a multi-layer perceptron (MLP) learned to output a population encoding in the form of 8 possible homing directions based on the home view and current view. A perceptron with Delta Rule was also shown to learn vector coordinates of a normalized homing vector, but this model was more sensitive to noise. The simulation environment is small and limited to a white background with black cylinders as landmarks, although the idea is essentially the same for this thesis. Namely, learning the home vector bearing from visual input. The main difference between the two approaches would be that for this thesis a single current image is used as input, whereas Hafner's work uses a target-driven visual approach taking the home view superimposed with the current view as input. Also, a more state-of-the-art convolutional neural network (CNN) is implemented for home vector inference and the simulation environment is much richer in visual cues (and noise).

An alternative homing hypothesis was given in the form of familiarity [31], where experienced views were encoded in an abstract measure using an ANN. Current views were then fed to the ANN to compute their related familiarity measure. The agent needed to turn on the spot to identify the direction that seems the most 'familiar'. To break the constant stop-turn-proceed cycle, an attraction/repulsion familiarity model has been proposed [10]. Regular familiarity relates to the attraction factor, whereas the inverse direction is learned as a repulsive metric. These two values were compared for current views in order to turn more efficiently while on the move.

Some additional homing approaches are: (1) control law based on landmark bearings and Lyapunov equation [50]; (2) landmark bearing extraction using 1D trifocal tensor (triangulization between three views) [51]; and (3) a visual compass approach based on learned pixel weights, where the orientation is relative to the home location instead of to a global reference frame [52].

Olfactory Homing
Insects such as bees and desert ants rely mostly on visual means to home towards the nest. Close to the nest, olfactory means could be utilized whenever possible. Other ant species primary use pheromones to find the shortest path to a regularly visited food source. The olfactory mechanism was exploited in [53] to localize a gas source in an unknown environment using a swarm of nano quadcopters.

Localization and Mapping
In robotics, the standard approach to find a way back to the starting location is facilitated by SLAM. In such approaches, metric maps are constructed through odometric and visual means, sometimes supplemented by laser range finders and GPS sensors. The map is used to simultaneously localize the agent in the map and consequently in the physical environment. A subset of SLAM, simply localization, is often used when less accuracy is demanded or when the operating environment is constantly changing. A comparison between holistic, e.g. image-warping, and feature-based localization means was given in [55]. Alternatively, a visual compass could be used to find the agent's global orientation [57] or line features could be tracked over several omni-directional images [56]. In [41] a CNN was trained with images, labeled through odometry and GPS data, in order to infer the metric coordinates and orientation respectively for experienced views. Whereas a full SLAM implementation was presented in [54], which was based on a model of the hippocampus of a rat.

Visual Navigation
Visual navigation methods are not necessarily related to finding a viable path to a goal location. They can be visually target-driven or simply follow a prominent visual guide such as a forest path [58]. In [42] and [43] a reinforcement learning (RL) approach was learned with the goal view superimposed on the current view. Red, Green, Blue, Depth (RGBD) and Red, Green, Blue (RGB) images were used in the respective works. The output of the RL algorithm were actions rather than an explicit homing vector. To keep the action domain manageable and suppress noise, actions are usually limited to a small set, such as: forward, left, and right. RL approaches in concatenation with ANNs could also be used for obstacle avoidance in unknown environments [44]. A single image was used to estimate the probability of collision, whereafter an appropriate steering angle could be computed that steers clear of the obstacle before collision.

Part III

Preliminary Experiments

5

Methods and Tools

This chapter sets up the preliminary simulation, which serves as a guideline to future experimentation. In Section 5.1, the complete CNN architecture is dissected. Image sample acquisition and preprocessing is facilitated by the Flightmare simulation framework (see Section 5.2). Simulation is essential for a cost effective neural implementation, since the CNN performance is mostly dependent on the quality of data that has been fed to it.

5.1. Neural Architecture

For the decision on what neural network (NN) to use, it is important that the architecture is relatively simple, is biologically inspired in some way, and does not result in an excess of parameters to store. Since it remains very hard to study the structure of insect's brains, it is currently not possible to determine the exact extent of their storage capacity. Therefore, it is preferable that the storage requirements are as small as possible. NNs are a form of general problem solvers, they give implicit cues to the underlying functional mechanism of the studied phenomena [60]. Even if these networks are sometimes seen as a 'black box', forthcoming research is being performed to make them more tangible. They are data-driven approaches. As a consequence, it is important to take care of the quality of the inputs, especially for supervised learning. Outliers and data inconsistencies could result in flawed outputs of the network. NN based applications are numerous, including but not limited to: classification, localization, detection, segmentation, and registration [61]. They are employed in several fields, such as medical imagery, construction, language processing, and robotics.

A popular procedure for learning from visual data is given in the form of the Convolutional Neural Network. Filter weights, associated with certain features of the image, are learned. These filters slide over patches of image pixels and accumulate them in a single cell. Several other layers, such as filters or fully connected (FC) layers could be added following the initial kernel layer. For this thesis, a very simple implementation is used. At first, the full $360°$color image is used as input to the network. It was quickly deemed unnecessary to retain the full color due to the excess in network parameters related to multiple color channels. A single input channel is sufficient to portray most of the relevant information. The input image resolution is currently chosen to be 1600×201 pixels, but could be resized in the future according to the needs of the algorithm. Smaller image sizes result in even less weights to be trained and it would also approximate the natural retinal image acuity of bees better. Whereas the inclusion of more pixels would result in more detail in the captured image frame.

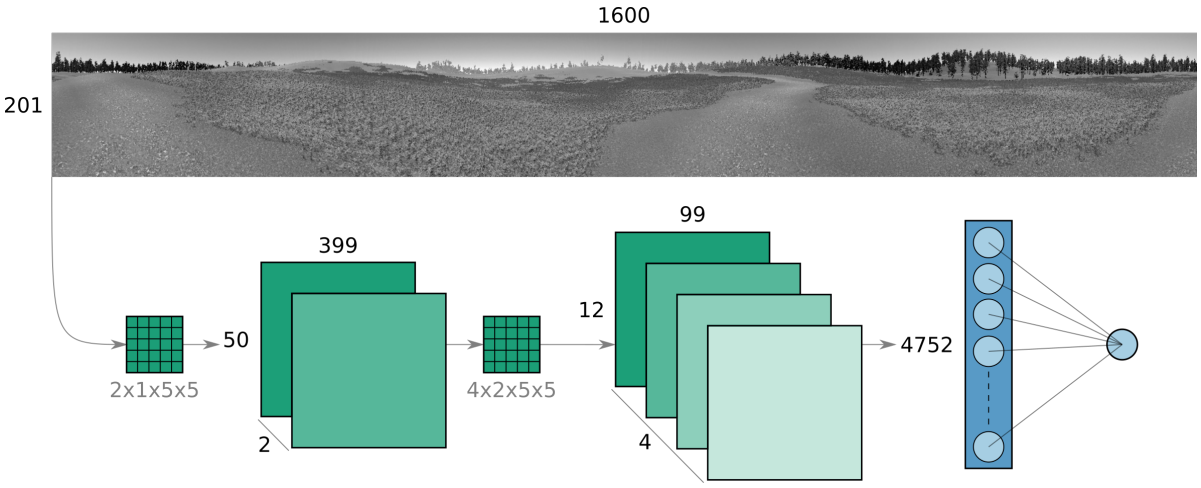


Figure 5.1: CNN Architecture with example input and feature map sizes. Dimensions are given per layer. Kernel weights are portrayed at half opacity. Teal layers are convolutional and blue layers are fully connected. The last node has an output value in [-1,1], encoding the relative home bearing.

The chosen CNN architecture includes two convolutional layers and one fully connected layer. It is implemented using the popular PyTorch library[1] for Python. The previously mentioned 1600×201 image was used as input, but this case functions primarily as a first case study of what a practical input could look like (see Figure 5.1). First, a 5×5 convolutional kernel is used to expand the input image into feature maps. Inserting the $1 \times 201 \times 1600$ (channels x height x width) into the network, it is expanded into a $2 \times 50 \times 399$ feature map by the $2 \times 1 \times 5 \times 5$ (output channels x input channels x kernel height x kernel width) filter with a stride of 4, where the distinction of the kernel is written conventionally in order to easily link it to the amount of weights to be learned later on. Again, the $2 \times 50 \times 399$ feature map is run through a second convolutional layer, holding a $4 \times 2 \times 5 \times 5$ kernel with stride 4, to output a $4 \times 12 \times 99$ feature map. Subsequently, this map is flattened into a $1 \times 4752 \times 1$ one-dimensional tensor. Lastly, the nodes of the tensor are fed into a FC layer concatenating into a single output node. The two convolutional layers and the FC layer all use the $\tanh$ activation function in their respective intermediate outputs. Intermediate output sizes are determined using the following formula[2]:

$$\begin{aligned} W_{out} &= \text{Floor}\left(\frac{W_{in} - (W_{kernel} - 1) - 1}{stride} + 1\right) \quad where \quad \text{Floor}\left(\frac{a}{b}\right) = \frac{a}{b} - \frac{a \bmod b}{b} \\ H_{out} &= \text{Floor}\left(\frac{H_{in} - (H_{kernel} - 1) - 1}{stride} + 1\right) \end{aligned} \tag{5.1}$$

Where W_{in} and W_{kernel} are the input and kernel widths respectively; the heights H_{in} and H_{kernel} are defined equivalently to the widths; and the variables a, b are arbitrary and used to define the Floor function.

Following, the total parameter count of the network is computed, again using the 1600×201 image as an example. The first convolutional layer contains 50 weights to be trained, evident from the $2 \times 1 \times 5 \times 5$ notation. The number of biases per layer is dependent on the amount of output channels. Thus, the total amount of parameters to learn in the network is equal to 5009 (5002 weights and 7 bias terms). The following list is given to summarize (parameters are given relative to their output layer):

1. convolutional layer 1: 50 weights + 2 bias terms
2. convolutional layer 2: 200 weights + 4 bias terms
3. fully connected layer 1: 4752 weights + 1 bias term

The output of the network is presented using a single node at the end of the fully connected layer. Since it is the output of a $\tanh$ activation function, it is bound in the interval $[-1, 1]$. The only step left is

[1]The PyTorch library main website: https://pytorch.org/
[2]The PyTorch Conv2d function: https://pytorch.org/torch.nn.Conv2d.html

to multiply the value of the last node with π and an angle is obtained that represents the home vector bearing angle with respect to the gaze direction of the agent. In practice, the network is never trained on -1 labels, otherwise the interpretation of the resulting angle would be ambiguous. Counter-clockwise angles with respect to the gaze direction are chosen as positive, whereas clockwise angles are taken as negative. If the agent's gaze direction is equivalent to the home vector, an output of 0 is produced.

5.2. Simulation and Environment

Data collection is of primary concern for the training of CNNs. Therefore, it is convenient to train and test neural architectures in simulation before implementing them in real world experiments. A simulation is defined as a framework to make different virtual environments possible [62]. A simulated environment refers to a distinct scene in the framework. Flightmare is used as an initial template to extract visual data [63]. It is a flexible program in the sense that the trade-off between photo-realistic environments and computational efficiency is made by the end-user. Flightmare consists of a client[3] which communicates to the renderer[4] on the server side. The renderer is based on the Unity Editor[5], which is a game engine that has recently been applied to robotics research as well as animation and construction fields. Unity provides an easy to work with editor to sculpt photo-realistic virtual environments. Its main benefits are the countless object meshes and terrain tools that are available on its asset store. With these tools, a multitude of environments can be generated using C# scripts. The Flightmare client interfaces with Unity to instantiate simulated agents, e.g. rotorcraft, and manipulate their interactions with the environment.

Several modifications are appended to the initial template, which will be made available on the author's Github. The original Flightmare code only provides the user with an interface for the control of a single agent. To efficiently capture as much data samples as possible, preferably 100 or more agents are required. Adjustments were made to the Flightmare client and server as to facilitate that many instances. Unity crashes occurred whenever a lot of perspective views were rendered during the same frame. Also, the Unity Terrain tool packages resulted in some problems. These were related to the limitations of the Unity terrain toolbox and the available random access memory (RAM) of the development system. Out Of Memory tools such as EarlyOOM[6] were used to keep the renderer's memory usage in check. In order to split the collection of imagery over multiple frames, only mesh objects could be utilized in the simulation. Furthermore, communication between the server and the client was built in ROS[7](C/C++) workspaces using the ZMQ messaging protocol[8]. All experiments were performed on an Intel®Core™ i7-8550U CPU @ 1.80GHz. Occasionally, a low-end GPU was utilized for demanding graphical computations.

Generated environments (and resultant sensor data) need to portray the natural surroundings of insects as accurately as possible. Forest environments were created with a variety of trees, tussocks, ground, flowers, and bushes. Unity only allows sampling of the environment using perspective cameras. To construct an omni-directional image from perspective images, a cubemap is constructed using 6 cameras per agent. This cubemap is on its turn transformed into a catadioptric projection and subsequently rectified to be fed into the CNN. Perspective cameras in the simulation all have a FoV of 90°and a resolution of 1024×1024 pixels. The perspective images are taken from the front, back, left, right, dorsal, and ventral regions of the agent. Then, stitched together in the form of an unfolded cube. Catadioptric projections are performed on the cubemap according to [5], assuming a specific mirror shape, its latus rectum lr [m], and distance to the camera image plane d [m]. The overall process is summarized in Figure 5.2. It can be noted that the catadioptric image, in the middle (Figure 5.2 (B)), seems to be horizontally mirrored with respect to the cubemap and rectified image as a direct consequence of the projection.

A point in the three-dimensional environment is projected onto the image plane using a normalized plane, which is defined as an imaginary plane in between the mirror and the image plane of the per-

[3]Flightmare Client Github: https://github.com/flightmare
[4]Flightmare Unity Renderer Github: https://github.com/flightmare_unity
[5]The Unity Editor: https://unity.com/
[6]EarlyOOM tool for Ubuntu: https://manpages.ubuntu.com/earlyoom.html
[7]The robot operating system (ROS) main page: https://www.ros.org/
[8]The ZMQ protocol main website: https://zeromq.org/

spective camera image. This projection is characterized by the non-linear function $h(x)$, which can be approximated by a discrete pixel index map (PIM) $H \in \mathbb{R}^{3\times(W_{plane}\cdot H_{plane})}$, representing a sawtooth pattern, assuming a parabolic mirror and an image plane of $W_{plane} \times H_{plane}$ in pixels:

$$H = \begin{bmatrix} 0 & 1 & \cdots & PIM[-1] & \cdots & 0 & 1 & \cdots & PIM[-1] \\ 0 & 0 & \cdots & 0 & \cdots & PIM[-1] & PIM[-1] & \cdots & PIM[-1] \\ 1 & 1 & \cdots & 1 & \cdots & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{5.2}$$

Where $PIM[-1] = (W_{plane}\cdot H_{plane}-1)$ is the final pixel index. The projection of the normalized plane to the image plane is defined by the global calibration matrix $H_c \in \mathbb{R}^{3\times3}$ (global refers to total catadioptric system):

$$H_c = K_c M_c \tag{5.3}$$

Where K_c is the perspective calibration matrix and M_c the mirror behavioral matrix:

$$K_c = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}; \quad M_c = \begin{bmatrix} \nu-\xi & 0 & 0 \\ 0 & \xi-\nu & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

Where (u_0, v_0) refers to the optical center of the perspective camera. The geometry of the parabolic mirror is captured by the parameters $\nu = 1$ and $\xi = 1-20p$ with $4p = lr$. Furthermore, (f_x, f_y) represent the focal lengths of the camera with a receptive field of $W_{persp} \times H_{persp}$ in pixels and an arbitrary FoV τ:

$$f_x = \frac{W_{image}/2}{\tan(\tau/2)}; \quad f_y = \frac{H_{image}/2}{\tan(\tau/2)} \tag{5.5}$$

The PIM and global calibration matrix can be combined to give a projection from the image plane to the environment:

$$H^{env}_{plane} = (H_c^{-1})^T H \tag{5.6}$$

Through the back-projection model, an inverse of H^{env}_{plane} tuned to the mirror's geometry can be used to compute the projection from the environment to the image plane. Taking $\hat{x}$, $\hat{y}$, and $\hat{z}$ as the respective rows of H^{env}_{plane}, the coordinates $(x, y, z)^T$ of a point in the environment can be obtained:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{plane} = \left(\left[\frac{\hat{z}\nu+\sqrt{\hat{z}^2+(1-\nu^2)(\hat{x}^2+\hat{y}^2)}}{\hat{x}^2+\hat{y}^2+\hat{z}^2} \right] \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} \end{bmatrix} \right)^T - \begin{bmatrix} 0 \\ 0 \\ \nu \end{bmatrix} \tag{5.7}$$

Finally, the ray from the environment to the image plane must be rotated from the image plane reference frame to the environment reference frame. Then, the ray must be rotated according to the orientation of image plane in the environment. These rotations are lumped together in the global rotation matrix R_c:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{env} = R_c \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{plane} \tag{5.8}$$

The rectification of the catadioptric image is straightforward. Taking the region in between two concentric circles as a mask and mapping it from polar (r, θ) to Cartesian (u, v) coordinates:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} r\cos\theta \\ r\sin\theta \end{bmatrix} \tag{5.9}$$

It must be considered that during the Flightmare simulation, all images are captured with zero tilt in the pitch and roll directions. Tilt results in a sinusoidal pattern along the horizon after rectification [59]. The CNN may learn an undesirable bias towards tilt angles experienced during training. Thus, a tilt correction must be implemented in the form of an image shift before rectification is conducted. There will definitely be tilt present in the final robotic implementation of the algorithm, but for the simulations it can be assumed that all data samples are acquired without any tilt.

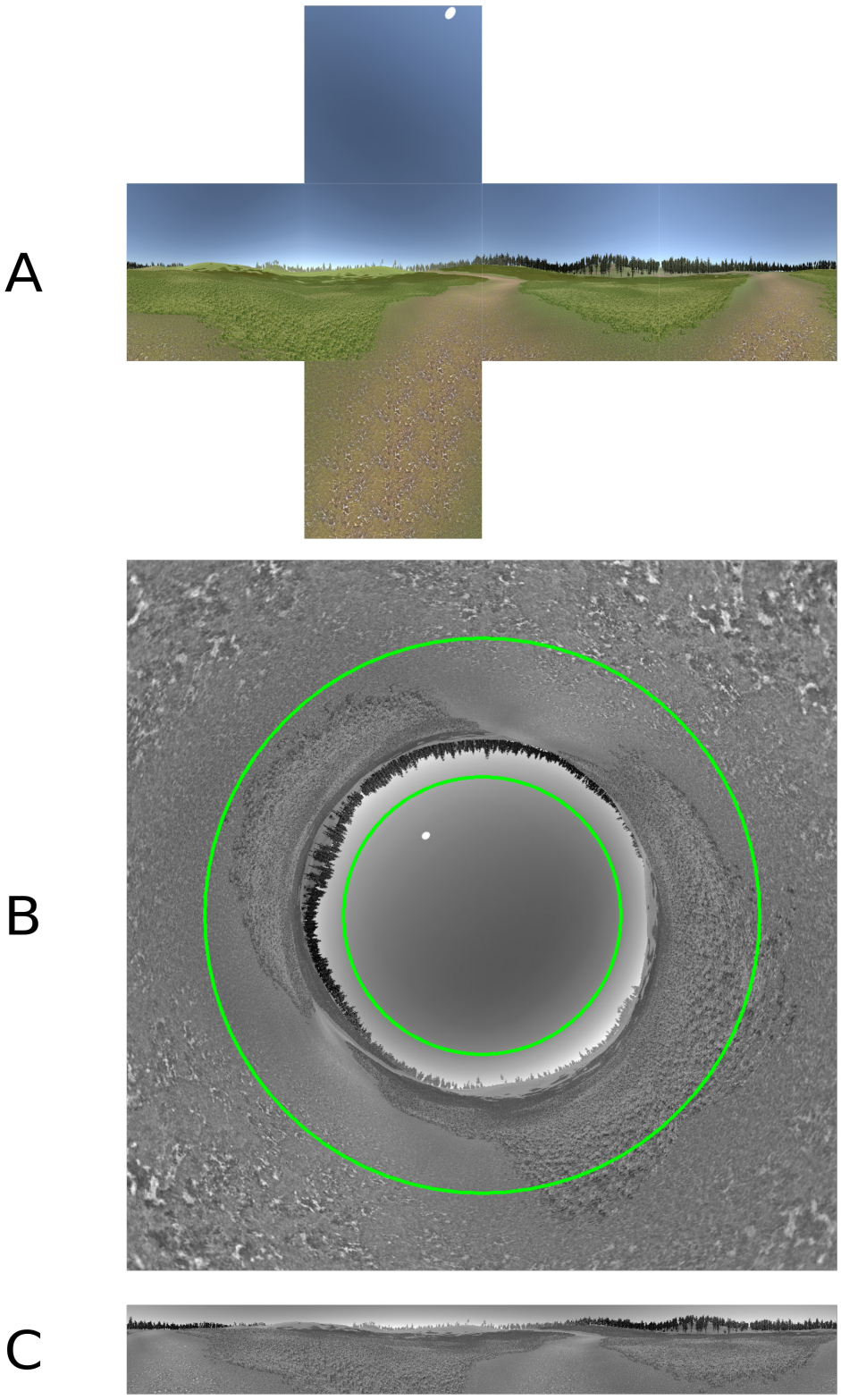


Figure 5.2: Summary of omni-directional image construction. (A) Unfolded cubemap of 6 perspective images with 90° FoV, image order from left to right: left, front, right, back. Dorsal image is on top and ventral image on the bottom. (B) Catadioptric projection of cubemap according to [5], green circles depict the region of interest with radii of 200 and 400 pixels. (C) Unfolded catadioptric image that is used as input for the CNN.

6

Home Bearing Learning and Inference Strategy

Implementing the CNN in simulation is but the first step in achieving insect-inspired bearing navigation. The preliminary simulation report mainly focuses on a simulated agent, which is supervised. The labels are calculated using simple geometry in a grid-like pattern where an omni-directional image is captured at every grid location. The CNN is then trained on all these images, excluding the nest location (because no relative angle is available for labeling). This way, the nest location can be easily defined as any arbitrary location on the grid. These experiments are presented in Chapter 7.

As with the snapshot model [3] and its relatives, no explicit goal condition has been defined. In other words, it is not exactly clear how accurate the current snapshot must match the snapshot at the goal for the agent to know that the goal location is reached. The bearing CNN presented in Section 5.1 also does not have an explicit goal condition. In an ideal implementation without any faults, the agent would constantly oscillate over the nest location. Additional information is needed for the agent to know that it has completed its homing task. The goal condition could be implemented using different approaches based on whether extra information should be incorporated or not.

- No extra knowledge required
    1. Oscillating agent, distance measure according to modulated amplitude
    2. Equilibrium point, flipping of home vector orientation
- Extra knowledge required
    1. Learning distance explicitly, adding output node to CNN
    2. Learning normalized relative angle, home vector bearing divided by distance
    3. Recognizing nest cue with ventral camera

The simplest proposed method includes no additions to the network output nor to the agent's environment. The goal condition would be defined as the equilibrium point over which the subject oscillates. Imagine all home vectors pointing towards the nest. When the agent overshoots the nest, the next home vector update would be flipped and she would turn around accordingly. Repeating this procedure would result in oscillating over one particular spot, the nest location.

Most of the aforementioned ideas are used to incorporate the inference of a distance measure in the agent's logic. This would have the disadvantage that any discrepancies in the approximation of the distance would result in a faulty home state. Idea number 3, recognizing the nest with a ventral, downward pointing, camera has the most promise, because it does not rely on any approximate measures. A region of interest (ROI) could be defined in the frame of the camera and the agent could constantly check whether the goal condition is met. The nest would be presented as a bright colored box or with a specific code that is distinct from the rest of the environment. The algorithm could be evaluated only on

its ability to home effectively using the generated home vector, instead of having to take into account a possibly noisy measurement estimate that could confuse the agent. This approach is thus deemed as most appropriate to implement.

On the other hand, if it would be beneficial to have a distance measurement, e.g. cross-check with battery life in order to maximize the time at the operating site, one of the other options could be implemented. The oscillating agent idea originated from the literature study, particularly from [6]. In this paper, A. Wystrach explains that animals have distinct oscillation patterns and argues that the amplitude of these oscillations are modulated based on the current familiarity of the experienced scene, while keeping the frequency constant. Assuming an ideal bearing calculation and odometry (also the case for next options), one could imagine an agent at an arbitrary distance from the nest. Gazing directly to the nest, where the relative angle would be equal to zero, the agent could translate a certain distance to the left and remember the observed difference in relative angle from its first location. Then, she could move back to the center and repeat the maneuver to the right. The same difference in relative angle would be observed for the same distance travelled. Keeping this difference in relative angle fixed along with the frequency of oscillation would result in bigger oscillations farther out from the nest, while exhibiting smaller oscillations closer to the nest. Therefore, the amplitude of the oscillations would give an indirect distance measure to the nest location. Whenever the oscillation amplitude would be zero, the nest would be reached.

The other approaches to estimate distance would result in alterations of the current CNN architecture and data, including an additional output node. The acquired data samples would have to be labeled with distance information as well. The distance label could be simply a binary value indicating whether the agent has arrived at the hive or not. The omni-directional image at the colony location would be taken into account during training and the relative home bearing angle would be ignored whenever this image would be evaluated. If a non-sense bearing angle estimation at the nest location would be undesirable, the ring of grid locations around the nest could be used as goal conditions instead. Alternatively, a more exact distance measure could be learned in concatenation with the home bearing angle. This distance value could be absolute or encoded in a Gaussian distribution with the peak being the nest, although this approach seems even more susceptible to noise than the other ones.

The CNN could also not be altered, instead the label could be modified to include an implicit distance cue. Instead of the pure relative home bearing angle that is being inferred, the normalized relative angle with distance could be used. This would not be a direct measurement since it still needs a predefined threshold. The closer the agent would be to the nest, the faster the inferred relative bearing angle will change. At a certain crossover-point, she could conclude that the nest is reached.

For the preliminary simulations, only four gaze directions are considered (rectified images): North, West, South, and East. It is envisioned that during future experiments, as much as 360 different gaze directions could be taken into account (more are expected to be extensive). This data could be extracted from a single omni-directional acquisition at a certain location. In theory, these gaze directions could be sampled and trained randomly to generate a good fit. Once the agent's performance is adequate in offline simulations, the goal of this thesis is in reach: A fully online implementation of the neural architecture able to learn in a self-supervised manner. The agent should be able to emerge from her nest, conduct a learning flight in gradually increasing circles around the colony (spiral), learn the relative bearings towards it, and venture out into the unknown. Upon her return to the vicinity of the nest, she should switch from PI to visual homing using the learned bearings in the learning flight. An example of such a learn/forage trajectory has been presented in Figure 6.1 as a schematic. Labeling of the training images would be done using imperfect, noisy odometry readings. Therefore, while testing this online approach in simulation, the visual system as well as the odometry system should have a realistic amount of Gaussian noise added to their readings. The agent should be tested in a multitude of environments with different objects and textures in order to evaluate whether different situations or lighting conditions affect the performance.

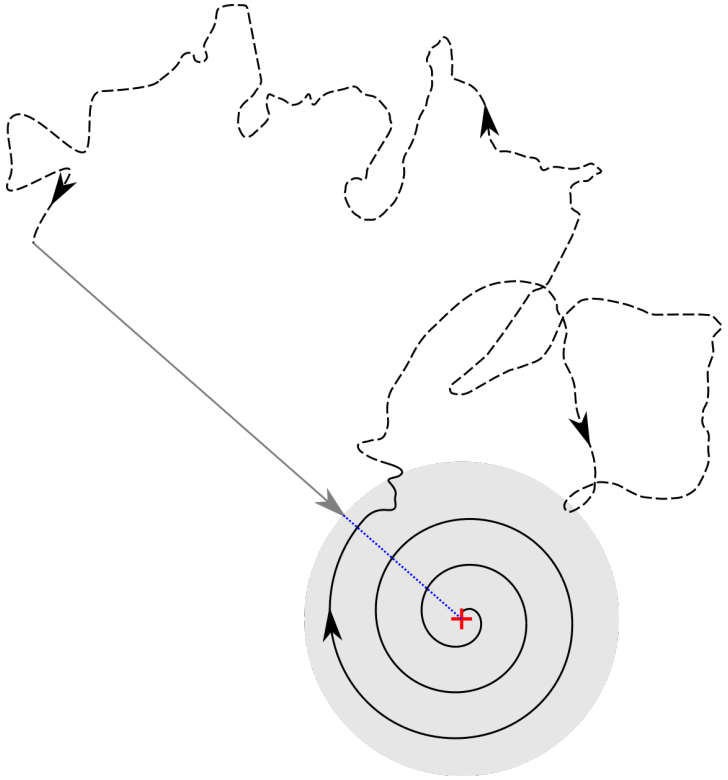


Figure 6.1: Schematic of an example trajectory for the online training and visual homing of an intelligent agent. Nest position is indicated by the red cross. The black solid line shows the flight trajectory of the learning flight (planar). The dashed black line displays an arbitrary foraging flight. The opaque line is the trajectory resultant from the PI home vector. The dotted blue line is the flight path associated with visual homing. Arrows indicate the flight direction.

Verification of software, deployed for the following experiments, was performed in the form of unit tests. Small code blocks were tested as solitary, independent elements before being embedded in the greater whole. In case of computations, e.g. geometric relations for labels, test cases were subjected to manual implementation. Compiler errors along the program pipeline were dealt with on-the-fly. Whenever compiler errors withheld crucial information for solving a particular issue, problem hypotheses were eliminated one after another until a conclusion could be drawn from the remainder.

7

Experimental Results

Two main experiments were conducted in simulation: (1) altering nest location across the grid and (2) a sensitivity analysis. The sensitivity analysis aimed to give an indication of the maximum distance from the colony at which successful homing can still be achieved, the catchment area. Experiment 1 (see section 7.1) is self-explanatory, just shift the nest location to a different grid location and repeat while keeping the grid distance constant. Experiment 2 (see section 7.2), relating to the catchment area, was dependent on multiple factors though, while keeping the nest location fixed. It consisted of varying the grid distance to figure out what the maximum effective distance of the homing would be. Based on grid distance alone, no viable conclusions could be drawn for the sensitivity analysis. Therefore, experiment 2 also included evaluation on image samples that were not seen during training. The agent would train on every other grid position, afterwards testing would be performed on all grid positions. All experiments assume the planar motion constraint and take into account four gaze directions centered on their respective quadrants.

7.1. Experiment 1: Impact of Changing the Nest Position

The performance of the CNN might be dependent on what location is selected to represent the nest location. A grid of 10×10 units was constructed in Flightmare's original "Industrial" environment. Generic units are used here because Unity does not represent any distance measures explicitly. The scale of the environment is chosen to represent metric values as much as possible. The same training and evaluation data are used. This would correlate with an agent that evaluates her trained home bearing at the location for which she learned these bearings in the first place. Iteration of the nest location was conducted row-wise from left to right. An overview of the full experimental procedure is listed below:

1. Calculate relative home vector bearings and write to file.
2. Remove nest image from training image directory.
3. Train the bearing CNN.
4. Log training results and network parameters.
5. Load network parameters and evaluate.
6. Log performance figures.
7. Move nest image back to image directory.
8. Change nest location.
9. Repeat.

As stated in chapter 6, labels are computed from geometric relations across a uniform grid. The relative angle between the estimated home bearing and gaze direction ω is deduced from two vectors situated at the agent's current location, where the gaze direction is represented by a pre-determined unit vector (e.g. North is [0,1]). Assume two vectors, (x_{home}, y_{home}) and (x_{gaze}, y_{gaze}) indicating the direction of the nest location and the gaze direction respectively. Then, the relative angle between the two vectors can be calculated by taking the 2-argument $\arctan$ ($\arctan 2$) of the determinant of the vectors' augmented matrix divided by their dot product:

$$\omega = \arctan2\left(\frac{x_{gaze} \cdot y_{home} - y_{gaze} \cdot x_{home}}{x_{gaze} \cdot x_{home} + y_{gaze} \cdot y_{home}}\right) \tag{7.1}$$

During initial simulation runs, the highest observed errors in relative home bearing approximation were always associated with home vectors that were oriented $180°$from the gaze directions. Imagine an agent gazing North with the nest towards the South. The inferred relative angle was almost never $180°$and approximation errors in these grid locations were so high that no other useful conclusion could be drawn from initial results. Thus, an unusual approach was taken to mitigate these high errors. All labels were scaled by 1.1^{-1} for training and multiplied by 1.1 before evaluation. This was done because it was thought that $\tanh$ outputs would saturate around values of 1, that the network would have difficulty inferring values at the edge of the activation function's range while keeping accuracy over the rest of the output range. Another viable explanation was given in [59], where the authors argue that errors may occur due to the fact that rectified images are discontinuous. The cutting line of an omni-directional image, associated with the edges of the rectified equivalent, introduces a discontinuity in what would else be a continuous circular view. As a result, ANNs may infer the left of the image in a totally different way as the right side of the rectified image. Their mitigation option proposed a spherical image representation with adjusted kernel shapes made out of triangular cells. It was found that scaling the labels worked quite well as a simpler alternative.

The training directory contains unique file names such that the labels can be linked to the associated images by a .csv file. The annotated images are passed to an instance of the standard PyTorch dataloader. The following choices were made regarding the set-up of the CNN: (1) training batches with single images; (2) 100 training epochs per nest location; (3) mean squared error (MSE) loss function; and (4) Adam optimizer with initial learning rate of 10^{-4}. The image pixel values are normalized between [0,1] according to PyTorch convention. A discrete learning rate decay was applied to avoid getting stuck in local minima of the loss function. Future change of this learning rate schedule is a possibility, but it was found that the discrete decay worked well for all the initial experiments. The schedule drops the order of magnitude of the learning rate whenever the accumulated MSE loss over the whole epoch is below a specific value. Three learning rate drops were implemented at an accumulated MSE loss of 1, 0.01, and 0.0005 respectively. Network parameters were saved to a .pth file if the accumulated loss of the current epoch was smaller than the one experienced in the previous best training cycle. Logging of the network parameters as well as the lowest encountered accumulated loss during training was performed before the trained network was evaluated. After evaluation, all the results could be presented in a series of plots.

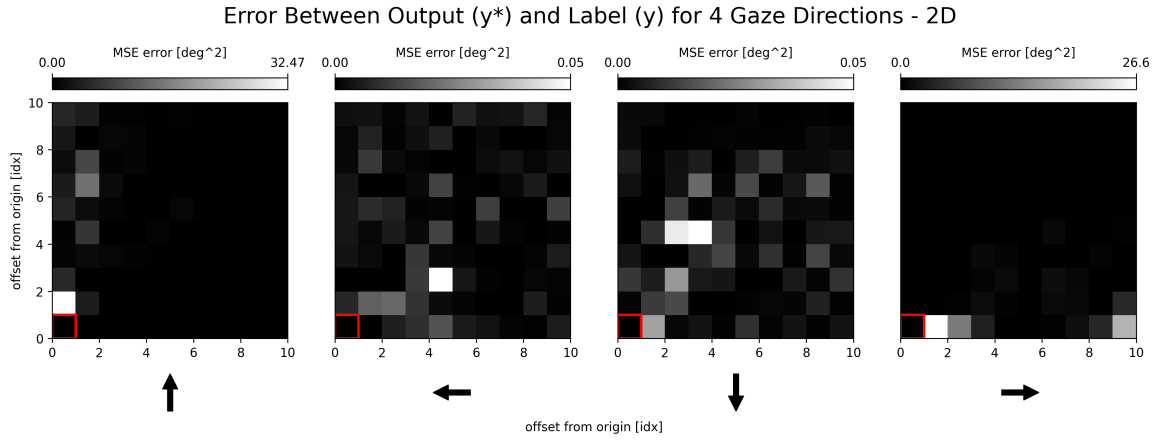


Figure 7.1: MSE error between network output and true angle for nest location (0,0) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location.

From Figure 7.1 one can notice that the error is largest whenever the agent is facing away from the nest. This is associated with the Northern and Eastern gaze directions, where the errors are orders of

magnitude higher than their opposite gaze directions. This could be explained by the fact that the agent is looking away from the nest during her training in the first and last plot. When looking in Southern or Western directions, she has a lot of training samples that look straight at the nest location. The error related to these orientations is distributed over the grid. Whereas looking in Northern or Eastern directions, she does not have any training samples with a direct view of the colony. High errors seem to occur more often in locations adjacent to the hive for these orientations. This could be the resultant of the agent adjusting to her environment, since the error decreases considerably for more distant locations. Another reason could be that it is harder for the agent to estimate the relative angle when she is very close to the nest. Slight misinterpretations may have a larger impact close to the nest.

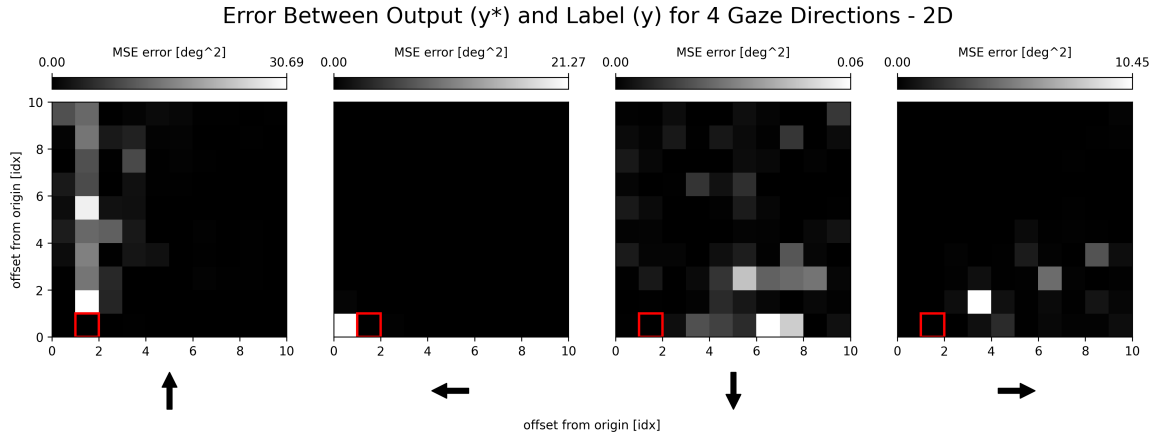


Figure 7.2: MSE error between network output and true angle for nest location (0,1) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location.

Moving one spot row-wise on the grid, Figure 7.2 can be generated. The observations in the Northern gaze direction are similar to the ones in Figure 7.1, where the highest error is adjacent to the nest location. Such patterns can be seen for all locations located at the edge of the grid. Moving only one location, the error in the Eastern looking orientation is suddenly lower than looking West. In addition, the error associated with the Western looking direction is the highest in the corner of the grid. The highest error for the Eastern gaze direction is not directly adjacent to the nest anymore, this can only be explained by the nondeterministic nature of the CNN. Having the nest close to the edge of the grid results in considerable bias towards certain gaze directions. It is assumed that the observations at these edge locations are the effect of training sample imbalances, where training samples in all considered gaze directions are not evenly distributed over the grid.

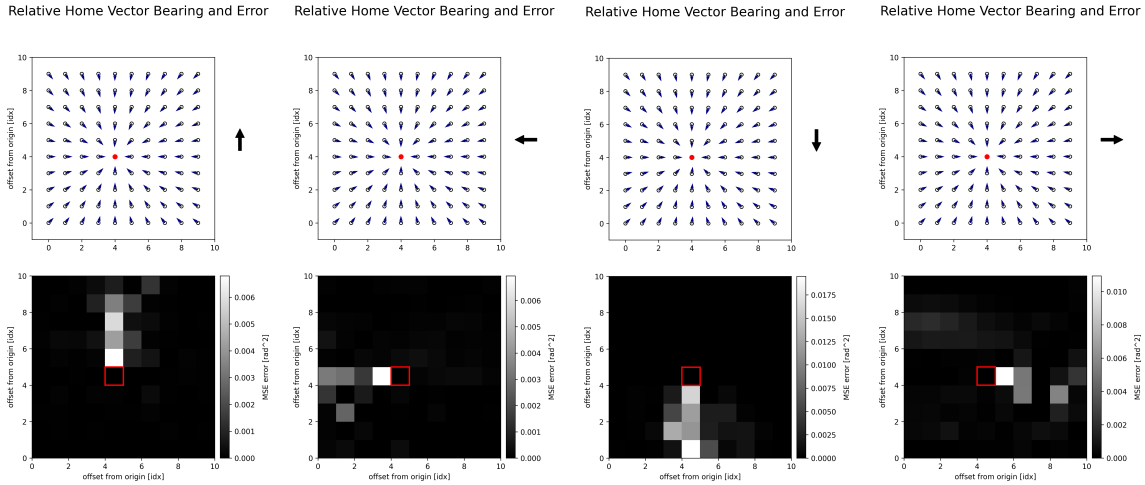


Figure 7.3: Relative home vector bearing and MSE error for nest location (4,4) on a 10x10 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle.

When moving along row 0 towards the middle, the errors in Western and Eastern gaze direction start to balance out. Repeating the experiment for every single grid location, it can be found that the errors start to balance evenly whenever the nest location is chosen about 3 positions away from any edge. Another cause for the higher errors when the simulated agent looks away from the nest could be related to the discontinuity of the rectified images. The distributed error is shown in Figure 7.3 along with the relative home vector bearings to the nest. Direction approximation is handled quite accurately by the bearing CNN. Peculiarly, it can be noted that the overall error for the nest at grid location (4,4) is higher than when the nest location is at a corner point (0,0), about 0.228 rad^2 for (4,4) compared to 0.060 rad^2 for the corner point. These results would support honey bee TBL observations in nature, where the bee conducts her learning flight while gazing directly at the nest and explores one direction at a time.

To investigate whether the errors adjacent to the nest are largest because of the small distance for which the angle is evaluated at, the error scaled with distance is plotted in Figure 7.4. It is clear that the error vanishes at more than one cell from the hive. This is taken as a strong indication that the aforementioned intuitive conclusion was correct (check discussion Figure 7.1). The highest errors are observed right next to the nest because of the smaller error margin associated with flying close to the nest. In other words, the home vector changes more rapidly close to the nest than farther away.

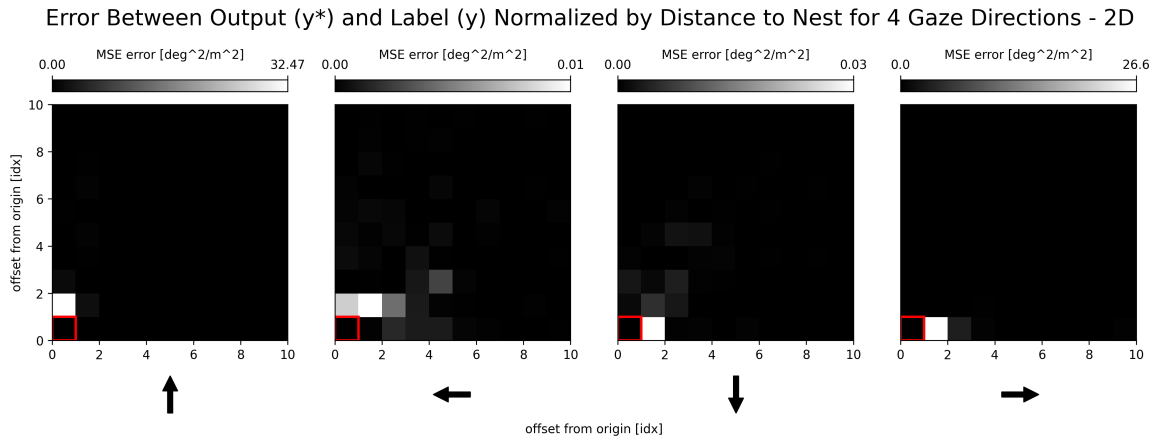
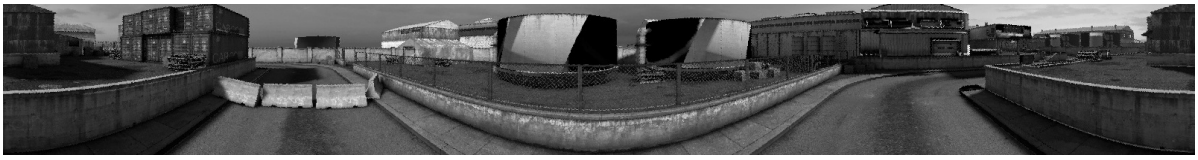
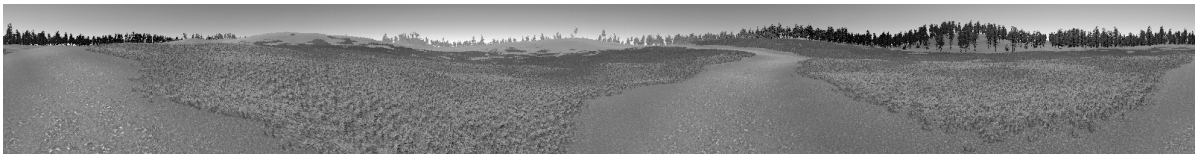


Figure 7.4: MSE error between network output and true angle, scaled with distance, for nest location (0,0) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location.

Note that in the ”Industrial” environment, a lot of landmarks are present to the North and South of the agent. Less so to the West and East, considering that every vertical object in the scene would be interpreted as a landmark, which is currently an unknown. The definition of what a landmark entails will be studied in further thesis work.



(a) The Flightmare Industrial environment



(b) The created Forest environment

Figure 7.5: The two environments used in the preliminary simulation experiments.

Conclusions regarding landmark distribution are to be made in section 7.2. Landmark density may be another viable reason for the anomalous observations in Figure 7.2 for Western looking directions or the higher errors may still be due to an imbalance of data samples. For the next experiment, a dedicated forest environment was prepared with a large open plane in the middle surrounded by trees in the distance. Analyses in this environment may shed further light on this conundrum. The reader can observe both scenes in Figure 7.5.

7.2. Experiment 2: Sensitivity Analysis

This section deals with the performance investigation of the algorithm, namely what the maximum evaluation distance is and how well it generalizes to unseen locations. Together these tests are seen as a trial-and-error approach in determining the catchment area for the bearing CNN. The catchment area can thus be defined as the maximum area around the nest location for which an acceptable performance can be achieved over all locations, not only the trained grid locations. The experimental overview is very similar to experiment 1, with the exception that agent instantiation must be performed multiple times. This means that the simulation environment must be invoked every time a larger distance between grid points is evaluated.

1. Calculate relative home vector bearings and write to file.
2. Run Flightmare and acquire training images.
3. Rectify catadioptric images.
4. Remove nest image from training image directory.
5. Train the bearing CNN.
6. Log training results and network parameters.
7. Load network parameters and evaluate.
8. Log performance figures.
9. Move nest image back to image directory.
10. Change grid distance.
11. Repeat.

All CNN settings and calculations are identical to experiment 1. The total distance of the grid is varied from the previous 10 units to at most 200 units, correlating with a 20 unit distance between grid nodes. From Figure 7.6 it can be concluded that, for the same 10×10 grid and (4,4) nest position, having a distant landmark panorama compared to only having close landmarks (Figure 7.3) decreases the error significantly. The overall error during training on the ”Industrial” environment was 0.228 rad^2 compared to 0.016 rad^2 for the forest environment.

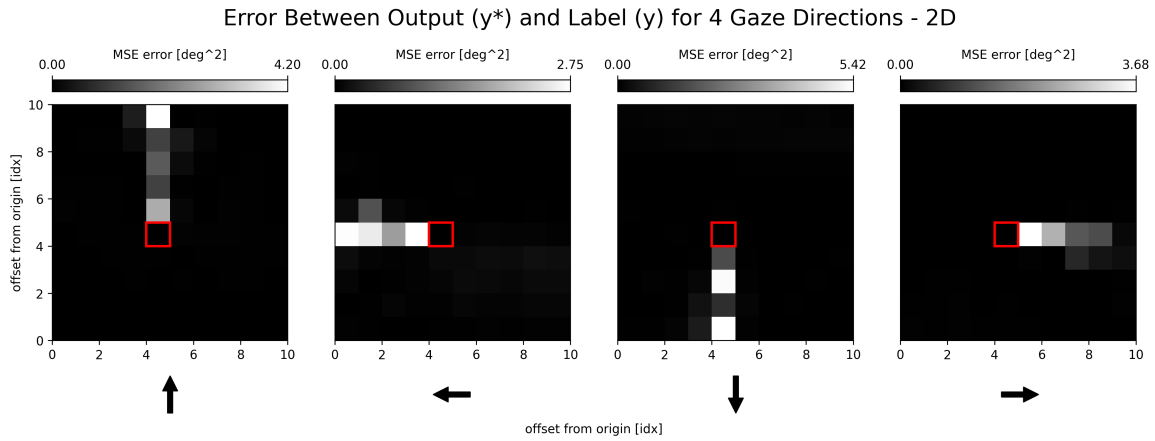


Figure 7.6: MSE error between network output and true angle for nest location (4,4) on a 10x10 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location.

Now, the distance of the grid is increased to investigate what kind of impact it would have on the approximation error. The error starts to shoot up for the 100×100 grid (see Figure 7.7). For the first time, the highest error is occurring farther away from the nest location. The overall error jumps up to about 0.083 rad^2. The algorithm still exhibits remarkable home vector inference at grid locations. Ambiguity in the landscape intuitively results in less certainty in the estimate. The consequences of ambiguity in the scene need to be studied in further thesis work, where the forest scene's open plane is enriched with more trees.

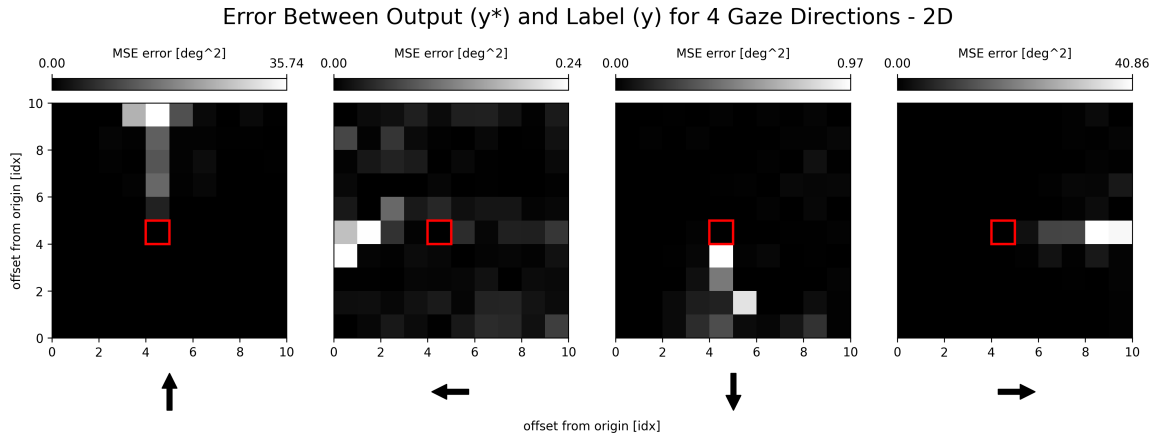
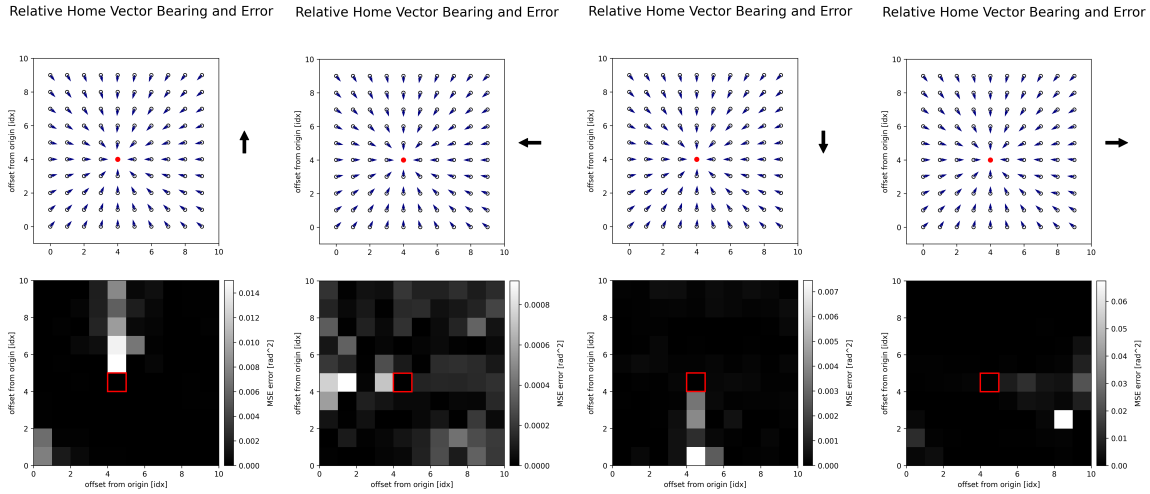


Figure 7.7: MSE error between network output and true angle for nest location (4,4) on a 100x100 grid. Black arrows give the gaze direction of the agent and the red box indicates the nest location.

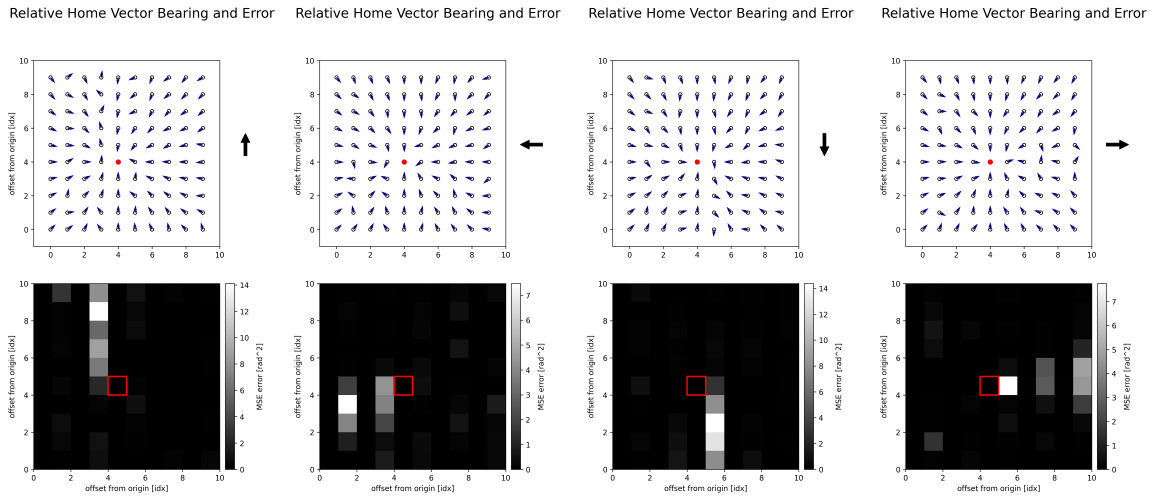
The final distance experiment was performed on a 200×200 grid with a 20 unit distance between every grid point. The errors associated with this test are given in Figure 7.8. Even tough the error keeps increasing, the agent would still be able to home effectively according to the presented relative home bearing maps. The question remains whether this performance is retained even if the network needs to evaluate grid locations it has never encountered before.



(a) North **(b)** West **(c)** South **(d)** East

Figure 7.8: Relative home vector bearing and MSE error for nest location (4,4) on a 200x200 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle.

Learning to infer relative home vector bearings at large distance does not have any value unless the interpolation between grid points is successful as well. This is where the current algorithm starts to show its limitations and performance starts to deteriorate. An inclusive set of grid distances, compared to the altering the grid distances, is utilized to test the generalization of the algorithm (1 m, 10 m, 20 m). Then, the network is presented with images acquired at every other grid point (even-numbered image identifiers). Whereas it is evaluated on all the images, regardless whether the image was used for training or not. This resulted in a very large increase in the errors, especially at the locations the network had not experienced before (see Figure 7.9). This can be explained by the network either overfitting or having a lack of generalization capabilities. Both could be mitigated by the use of image augmentation and random sampling of training data at every training epoch. Whenever the network would still fail to generalize to intermediate grid positions after these modifications, the network architecture will be deemed insufficient for the task and alternate configurations will be explored.



(a) North **(b)** West **(c)** South **(d)** East

Figure 7.9: Relative home vector bearing , only trained on every other grid point, and MSE error for nest location (4,4) on a 200x200 grid. Black arrows give the gaze direction of the agent; the red box and circle indicate the nest location; and the blue arrows show the relative angle.

Due to the erratic approximations at odd-numbered columns (starting at 0) in the relative home vector map, the agent may get stuck in local loops or may leave the experimental area completely. As

mentioned before, this is to be mitigated by presenting the CNN with directions that were previously unseen; randomly select directions from captured omni-directional imagery (more than four gaze directions); and experiment further with the grid size. This way, it is believed that the algorithm will be able to successfully home within pre-determined bounds.

8

Conclusion

This work has shown that learning home vectors with modern neural architectures is not only possible, but can be a preferred navigational method compared to existing bio-inspired alternatives. The bearing convolutional neural network exhibits very viable homing performance for photo-realistic simulated and real targets. The bearing net needs to be supplemented by compass cues during its learning cycle. Usage of a compass cue is a prerequisite for many insect-inspired navigation substitutes.

We proposed an insect-inspired visual homing approach that relies exclusively on inferred home vector bearings from single rectified catadioptric images. The main hypothesis posits that bees retain directional cues to the nest following their initial learning flight. Subsequently, we have demonstrated the feasibility of training an artificial neural network in either a simulated or actual forest environment to effectively execute the homing procedure in a bee-like manner. Moreover, this has been realized by utilizing comparable resources as those employed by the actual bee, i.e. visual and odometric senses.

Snapshot-related approaches suffer from deteriorating performance in noisy realistic scenarios or in the presence of distant landmarks. Familiarity-based methods mitigate many of these shortcomings, but evince repetitive operations to achieve the same essential bearing estimate. In a parsimonious paradigm, the simplest solution, made possible with the subject's available means, must tell us the most about its natural behavior. Therefore, bearing estimation, as performed in this thesis, can be seen as the next step in understanding the inner workings of navigating foragers.

Overall squared error during training of various preliminary simulation experiments (in the same environment) are given, accumulated for all locations on a square 10×10 agent grid, where the nest location is situated in the middle of the grid:

- 1 unit inter-agent distance: 0.016 rad^2
- 10 units inter-agent distance: 0.083 rad^2
- 20 units inter-agent distance: 0.329 rad^2

When generalizing to views that were not experienced before, the network attained an overall squared error of 0.469 rad^2 with 20 units inter-agent distance. This accounts to a 40% error increase compared to the case where the network was only evaluated on experienced views.

We aspire to conduct further research on Hymenopteran navigation, thereby assisting zoologists and biologist to explore its intricate mechanisms. Future efforts would involve extra tuning of the sample acquisition trajectories and other network hyperparameters. Furthermore, we envision the potential integration of supplemental flight time training labels as a distance cue.

Bibliography

[1] Benhamou, S., and Séguinot, V., "How To Find One's Way in the Labyrinth of Path Integration Models," *Journal of Theoretical Biology*, Vol. 174, No. 4, 1995, pp. 463–466.

[2] Lehrer, M., and Bianco, G., "The Turn-Back-and-Look Behaviour: Bee Versus Robot," *Biological Cybernetics*, Vol. 83, 2000, pp. 211–229.

[3] Cartwright, B. A., and Collett, T. S., "Landmark Learning in Bees," *Journal of Comparative Physiology*, Vol. 151, 1983, pp. 521–543.

[4] Lambrinos, D., Möller, R., Labhart, T., Pfeifer, R., and Wehner, R., "A Mobile Robot Employing Insect Strategies for Navigation," *Robotics and Autonomous Systems*, Vol. 30, 2000, pp. 39–64.

[5] Berenguel-Baeta, B., Bernudez-Cameo, J., and Guerrero, J. J., "OmniSCV: An Omnidirectional Synthetic Image Generator for Computer Vision," *Sensors*, Vol. 20, No. 7, 2020.

[6] Wystrach, A., "Movements, Embodiment and the Emergence of Decisions. Insights from Insect Navigation," *Biochemical and Biophysical Research Communications*, Vol. 564, 2021, pp. 70–77.

[7] Hafner, V. V., "Robots as Tools for Modelling Navigation Skills - A Neural Cognitive Map Approach," *Robotics and Cognitive Approaches to Spatial Mapping*, Vol. 38, 2008.

[8] de Croon, G. C. H. E., Dupeyroux, J. J. G., Fuller, S. B., and Marshall, J. A. R., "Insect-Inspired AI for Autonomous Robots," *Science Robotics*, Vol. 7, No. 67, 2022.

[9] von Frisch, K., *The Dance Language and Orientation of Bees*, 1st ed., Harvard University Press, Cambridge, MA., 1967.

[10] Möel, F. L., and Wystrach, A., "Opponent Processes in Visual Memories: A Model of Attraction and Repulsion in Navigating Insects' Mushroom Bodies," *PLoS Computational Biology*, Vol. 16, No. 2, 2020, pp. 1–24.

[11] Hafner, V., and Möller, R., "Learning of Visual Navigation Strategies," *Proceedings of the European Workshop on Learning Robots*, Vol. 9, 2001, pp. 47–56.

[12] Moël, F. L., and Wystrach, A., "Towards a Multi-Level Understanding in Insect Navigation," *Current Opinion in Insect Science*, Vol. 42, 2020, pp. 110–117.

[13] Capaldi, E. A., Smith, A. D., Osborne, J. L., Fahrbach, S. E., Farris, S. M., Reynolds, D. R., Edwards, A. S., Martin, A., Robinson, G. E., Poppy, G. M., and Riley, J. R., "Ontogeny and Orientation Flight in the Honeybee Revealed by Harmonic Radar," *Nature*, Vol. 403, 2000, pp. 537–540.

[14] Émile Zola, *La Fortune des Rougon*, 5th ed., G. Charpentier et C., Paris, France, 1875.

[15] Darwin, C., *On the Origin of Species*, 1st ed., John Murray, London, England, 1859.

[16] Hafner, V. V., "Adaptive Homing – Robotic Exploration Tours," *Adaptive Behavior*, Vol. 9, No. 3-4, 2001, pp. 131–141.

[17] Haferlach, T., Wessnitzer, J., Mangan, M., and Webb, B., "Evolving a Neural Model of Insect Path Integration," *Adaptive Behavior*, Vol. 15, No. 3, 2007, pp. 273–287.

[18] Srinivasan, M. V., Zhang, S. W., Lehrer, M., and Collett, T. S., "Honeybee Navigation En Route to the Goal: Visual Flight Control and Odometry," *The Journal of Experimental Biology*, Vol. 199, No. 1, 1996, pp. 237–244.

[19] Zeil, J., “Visual Homing: an Insect Perspective,” *Current Opinion in Neurobiology*, Vol. 22, No. 2, 2012, pp. 285–293.

[20] Franz, M. O., and Mallot, H. A., “Biomimetic Robot Navigation,” *Robots and Autonomous Systems*, Vol. 30, No. 1, 2000, pp. 133–153.

[21] Heinze, S., “Unraveling the Neural Basis of Insect Navigation,” *Current Opinion in Insect Science*, Vol. 24, 2017, pp. 58–67.

[22] Scaramuzza, D., and Fraundorfer, F., “Visual Odometry, Part I: The First 30 Years and Fundamentals,” *IEEE Robotics & Automation Magazine*, Vol. 18, No. 4, 2011, pp. 80–92.

[23] Lehrer, M., “Why do Bees Turn Back and Look,” *Journal of Comparative Physiology*, Vol. 172, 1993, pp. 549–563.

[24] Lehrer, M., “The Adaptability of the Homecoming Honeybee,” *The British Journal of Animal Behaviour*, Vol. 1, No. 2, 1953, pp. 59–66.

[25] Zeil, J., “Orientation Flights of Solitary Wasps (Cerceris; Sphecidae; Hymenoptera),” *Journal of Comparative Physiology*, Vol. 172, 1993, pp. 189–205.

[26] Collett, T. S., and Zeil, J., “Insect Learning Flights and Walks,” *Current Biology*, Vol. 28, No. 17, 2018, pp. R984–R988.

[27] Stürzl, W., Zeil, J., Boeddeker, N., and Hemmi, J. M., “How Wasps Acquire and Use Views for Homing,” *Current Biology*, Vol. 26, No. 4, 2016, pp. 470–482.

[28] Woodgate, J. L., Makinson, J. C., Lim, K. S., Reynolds, A. M., and Chittka, L., “Life-Long Radar Tracking of Bumblebees,” *PLoS ONE*, Vol. 11, No. 8, 2016.

[29] McGuire, K. N., de Croon, G. C. H. E., and Tuyls, K., “A Comparative Study of Bug Algorithms for Robot Navigation,” *Computing Research Repository*, Vol. abs/1808.05050, 2018.

[30] van Dijk, T., McGuire, K. N., de Croon, G. C. H. E., Campoy, P., and Jonker, P., “Low-Memory Visual Route Following for Micro Aerial Vehicles in Indoor Environments,” , 2017.

[31] Baddeley, B., Graham, P., Husbands, P., and Philippides, A., “A Model of Ant Route Navigation Driven by Scene Familiarity,” *PLoS Computational Biology*, Vol. 8, 2012, pp. 1–16.

[32] Möller, R., “Insect Visual Homing Strategies in a Robot with Analog Processing,” *Biological Cybernetics*, Vol. 83, 2000, pp. 231–243.

[33] Stürzl, W., Boeddeker, N., Dittmar, L., and Egelhaaf, M., “Mimicking Honeybee eyes With a 280° Field of View Catadioptric Imaging System,” *Bioinspiration & Biomimetics*, Vol. 5, 2010.

[34] Seidl, R., and Kaiser, W., “Visual Field Size, Binocular Domain and the Ommatidal Array of the Compound Eyes in Worker Honey Bees,” *Journal of Comparative Physiology*, Vol. 143, 1981, pp. 17–26.

[35] Singh, A. S., Beer, K., Ben-Shahar, Y., and Basu, P., “The Bee Brain: From Neuroanatomy to Social Behavior,” *(submitted, unpublished) Frontiers in Behavioral Neuroscience*, 2022.

[36] Bartol Jr., T. M., ad Justin Kinney, C. B., Chirillo, M. A., Bourne, J. N., Harris, K. M., and Sejnowski, T. J., “Nanoconnectomic Upper Bound on the Variability of Synaptic Plasticity,” *eLife*, Vol. 4, 2015.

[37] Zeil, J., Hofmann, M. I., and Chahl, J. S., “Catchment Areas of Panoramic Snapshots in Outdoor Scenes,” *Journal of the Optical Society of America*, Vol. 20, No. 3, 2003, pp. 450–469.

[38] Stürzl, W., and Mallot, H. A., “Efficient Visual Homing Based on Fourier Transformed Panoramic Images,” *Robotics and Autonomous Systems*, Vol. 54, No. 4, 2006, pp. 300–313.

[39] Baddeley, B., Philippides, A., Graham, P., Hempel de Ibarra, N., Collett, T., and Husbands, P., “What Can Be Learnt From Analysing Insect Orientation Flights Using Probabilistic SLAM?,” *Biological Cybernetics*, Vol. 101, 2009, pp. 169–182.

[40] Wang, S., Clark, R., Wen, H., and Trigoni, N., "DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks," *Computing Research Repository*, Vol. abs/1709.08429, 2017.

[41] Atsuzawa, K., Nilwong, S., Hossain, D., ichiro Kaneko, S., and Capi, G., "Robot Navigation in Outdoor Environments using Odometry and Convolutional Neural Network," *IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization*, 2019.

[42] Watkins-Valls, D., Xu, J., waytowich, N., and Allen, P., "Learning Your Way Without Map or Compass: Panoramic Target Driven Visual Navigation," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[43] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A., "Target-Driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[44] Palossi, D., Conti, F., and Benini, L., "An Open Source and Open Hardware Deep Learning-Powered Visual Navigation Engine for Autonomous Nano-UAVs," *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019.

[45] Wehner, R., "Desert Ant Navigation: How Miniature Brains Solve Complex Tasks," *Journal of Computational Physiology*, Vol. 189, No. 8, 2003, pp. 579–588.

[46] Aqel, M. O. A., Marhaban, M. H., Saripan, M. I., and Ismail, N. B., "Review of Visual Odometry: Types, Approaches, Challenges, and Applications," *SpringerPlus*, Vol. 5, No. 1897, 2016.

[47] Scaramuzza, D., and Fraudorfer, F., "Visual Odometry, Part II: Matching, Robustness Optimization, and Applications," *IEEE Robotics & Automation Magazine*, Vol. 19, No. 2, 2012, pp. 78–90.

[48] Hussaini, S., Milford, M., and Fischer, T., "Ensembles of Compact, Region-Specific & Regularized Spiking Neural Networks for Scalable Place Recognition," , 2022.

[49] Ma, J., Zhao, J., Jiang, J., Zhou, H., Zhou, Y., Wang, Z., and Guo, X., "Visual Homing and Guided Locality Preserving Matching," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7254–7261.

[50] Tron, R., and Daniilidis, K., "An Optimization Approach to Bearing-Only Visual Homing with Applications to a 2-D Unicycle Model," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4235–4242.

[51] Aranda, M., López-Nicolás, G., and Sagüés, C., "Angle-Based Homing from a Reference Image Set Using the 1D Trifocal Tensor," *Autonomous Robotics*, Vol. 34, 2013, pp. 73–91.

[52] Vardy, A., "A Simple Visual Compass With Learned Pixel Weights," *2008 Canadian Conference on Electrical and Computer Engineering*, 2008, pp. 001581–001586.

[53] Duisterhof, B. P., Li, S., Burgués, J., Reddi, V. J., and de Croon, G. C. H. E., "Sniffy Bug: A Fully Autonomous and Collaborative Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments," *Computing Research Repository*, Vol. abs/2107.05490, 2021.

[54] Milford, M. J., Wyeth, G. F., and Prasser, D., "RatSLAM: a Hippocampal Model for Simultaneous Localization and Mapping," *2004 Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2004, pp. 403–408.

[55] Fleer, D., and Möller, R., "Comparing Holistic and Feature-Based Visual Methods for Estimating the Relative Pose of Mobile Robots," *Robotics and Autonomous Systems*, Vol. 89, 2017, pp. 51–74.

[56] Fiala, M., and Basu, A., "Robot Navigation using Panoramic Tracking," *Pattern Recognition*, Vol. 37, No. 11, 2004, pp. 2195–2215.

[57] Sturm, J., and Visser, A., "An Appearance-Based Visual Compass for Mobile Robots," *Robotics and Autonomous Systems*, Vol. 57, No. 5, 2009, pp. 536–545.

[58] Lind, L., “Deep Learning Navigation for UGVs on Forests Paths,” *Degree Project in Engineering Physics*, 2018.

[59] Lee, Y., Jeong, J., Yun, J., Cho, W., and Yoon, K.-J., “SpherePHD: Applying CNNs on a Spherical PolyHeDron Representation of 360°Images,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9173–9181.

[60] Süderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., and Corke, P., “The Limits and Potentials of Deep Learning for Robotics,” *Computing Research Repository*, Vol. abs/1804.06557, 2018.

[61] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L., “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions,” *Journal of Big Data*, Vol. 8, No. 53, 2021.

[62] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D., “Unity: A General Platform for Intelligent Agents,” *Computing Research Repository*, Vol. abs/1809.02627, 2018.

[63] Song, Y., Naji, S., Kaufmann, E., Loquercio, A., and Scaramuzza, D., “Flightmare: A Flexible Quadrotor Simulator,” *Computing Research Repository*, Vol. abs/2009.00563, 2020.