Multi-modal Vision Transformers For Data Efficient Visual Odometry In Embodied Indoor Navigation

Master thesis by Marius Memmel (Student ID: 2817893)
Date of submission: May 13, 2022

1. Review: Professor Amir Roshan Zamir (Swiss Federal Institute of Technology Lausanne)
2. Review: Professor Stefan Roth (Technical University of Darmstadt)
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Swiss Federal Institute of Technology Lausanne

Visual Intelligence and Learning Lab

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Marius Memmel, die vorliegende Masterarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 13. Mai 2022

Marius Memmel

Abstract

Successful indoor navigation is a crucial skill for personal robots. This fundamental ability has been extensively studied through the task of PointGoal navigation in simulated environments. While these advancements allow for large-scale training, the conditions are idealistic. In the real world, indoor localization gets impeded by sensor inaccuracies and actuation noise. Visual Odometry has shown to be an efficient substitute for GPS and compass and can effectively localize the agent from visual observations. However, recent methods can not deal with multiple modalities and privileged information, *i.e.*, irregular access to one of them during test time. Furthermore, they do not sufficiently exploit the global structure of the Visual Odometry problem leading to inferior localization performance. As these properties become increasingly significant when moving toward autonomous systems and self-driving, this work presents a novel approach to multi-modal Visual Odometry. The proposed Visual Odometry Transformer combines the flexibility of Vision Transformers with exploiting geometric and structural properties of the Visual Odometry problem. It deploys a unified model, large-scale pre-training, and action prior, outperforming the state-of-the-art while using only 25% of the data. Additionally, the large receptive field of the attention mechanism uncovers and exploits the global nature of Visual Odometry in indoor navigation. Finally, the experiments demonstrate that the Visual Odometry Transformer implicitly learns invariance to the input modalities and privileged information.

Notation

General	
x	scalar
$\boldsymbol{x}$	vector
X	matrix
$\hat{\cdot}$	estimate
$I_{n \times m}$	identity matrix with n rows and m columns
Agent	
$\boldsymbol{o}$	observation
$\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$	observation tuple of time step t and $t+1$
$\boldsymbol{v}$	visual features
`RGB`	sensor reading from RGB camera
`Depth`	sensor reading from depth sensor
`GPS+Compass`	sensor reading from GPS and compass
a_t	action taken by the agent
`fwd`	move forward action
`left`	turn left action
`right`	turn right action
$\boldsymbol{g}$	relative goal position
Visual Odometry	
C_t	coordinate system at time step t
$\mathcal{R}_{C_t \to C_{t+1}}$	rotation (matrix) from C_t to C_{t+1}
$\mathcal{H}_{C_t \to C_{t+1}}$	transformation (matrix) from C_t to C_{t+1}
$\beta_{C_t \to C_{t+1}}$	angle in the rotation matrix $\mathcal{R}_{C_t \to C_{t+1}}$
$\boldsymbol{\xi}_{C_t \to C_{t+1}}$	translation (vector) from C_t to C_{t+1}
Dataset	
$\mathcal{D}$	dataset with datapoints d
$\widetilde{\mathcal{D}}$	augmented dataset with datapoints d
Model	
f_ϕ	function f parameterized by ϕ
$\mathcal{L}$	loss function

Contents

Abstract i

Notation ii

Acronyms vi

1. Introduction 1

2. Related Work 3
2.1. Embodied Visual Navigation . . . 3
2.1.1. Embodied Artificial Intelligence . . . 3
2.1.2. SLAM- and Learning-based Navigation . . . 3
2.1.3. AI Habitat Simulator . . . 5
2.1.4. Realistic PointGoal Navigation . . . 6
2.1.5. Camera Pose Estimation and Visual Odometry . . . 7
2.1.6. Visual Odometry for Realistic PointGoal Navigation . . . 8
2.2. The Transformer . . . 9
2.2.1. Attention Mechanism . . . 9
2.2.2. Multi-head Attention . . . 11
2.2.3. Attention Is All You Need . . . 12
2.3. The Vision Transformer . . . 13
2.3.1. An Image is Worth 16x16 Words . . . 13
2.3.2. Comparison to Convolution Neural Networks . . . 14
2.3.3. Transfer Learning . . . 15
2.3.4. Multi-Modality . . . 16

3. Methodology 17
3.1. Preliminaries . . . 17
3.1.1. Problem Formulation . . . 17
3.1.2. Policy Search . . . 17
3.1.3. Agent and Simulator . . . 19
3.1.4. Evaluating PointGoal Navigation Performance . . . 20
3.1.5. Visual Odometry Formulation . . . 21
3.2. The Visual Odometry Transformer . . . 22
3.2.1. Transformers for Visual Odometry . . . 22
3.2.2. Architecture . . . 23
3.2.3. Training Loss . . . 25

4. Experimental Evaluation **27**
4.1. Dataset . . . 27
4.1.1. AI Habitat Setup . . . 27
4.1.2. Data Collection Procedure . . . 27
4.1.3. Data Exploration . . . 28
4.2. Training Setup . . . 29
4.3. Baselines . . . 32
4.3.1. Trivial Baselines . . . 32
4.3.2. State-Of-The-Art Baselines . . . 32
4.4. Results . . . 33
4.4.1. Ablation Study . . . 33
4.4.2. State-Of-The-Art Comparison . . . 35
4.4.3. Backbone Initialization . . . 36
4.4.4. Attention Maps . . . 37
4.4.5. Action Prior Dependency . . . 40
4.4.6. Privileged Modalities . . . 41
4.5. Discussion & Outlook . . . 42

5. Conclusion **44**

References **45**

Appendix **53**
A. Dataset Statistics . . . 53
B. Experiment Configurations . . . 54
C. Extended Experiment Results . . . 56
D. Additional Attention Maps . . . 61
E. Auxiliary Depth Estimation Loss . . . 64

Acronyms

$[ACT]$ Action Token. 24, 34, 36, 37, 39–41, 55, 57, 59, 60, 63

$[CLS]$ Classification Token. 13, 14, 23, 24, 30

AI Artificial Intelligence. 1, 3, 5, 16

BERT Bidirectional Encoder Representations from Transformers. 13

BN Batch Normalization. 23

CBAM Convolutional Block Attention Module. 8, 22

CBN Conditional Batch Normalization. 23

ConvNet Convolution Neural Network. 1, 7–9, 13, 15, 16, 19, 20, 22, 23, 30, 31, 33, 35, 36, 41, 43, 58

CV Computer Vision. 3, 13, 15, 23, 30

DD-PPO Decentralized Distributed Proximal Policy Optimization. 18

DINO Self-DIstillation and NO Labels. 15, 16, 30, 32, 36–39, 41, 42, 57, 59–62

GELU Gaussian Error Linear Unit. 23, 64

Habitat AI Habitat. 5–7, 18–21, 27, 30, 31

IMU Inertial Measurement Unit. 23

IN-1k ImageNet-1k. 14

IN-21k ImageNet-21k. 14, 15, 30, 35

KNN K-nearest Neighbors. 16, 33

LSTM Long Short-Term Memory. 12, 20, 31

MAE Masked Autoencoder. 16

MDP Markov Decision Process. 17

MHA Multi-head Attention. 11, 12, 30, 37, 38, 61, 62

MHSA Multi-head Self-Attention. 12–15

MLM Masked Language Model. 13

MLP Multi-layer Perceptron. 13, 14, 23, 33, 36, 64

MultiMAE Multi-modal Multi-task Masked Autoencoder. 16, 25, 30, 31, 36–42, 55, 59–63

NLP Natural Language Processing. 3, 10

NSP Next Sentence Prediction. 13

POMDP Partially Observable Markov Decision Process. 17–19

PPO Proximal Policy Optimization. 18, 20, 31

ReLU Rectified Linear Unit. 23

ResNet Residual Network. 14, 15, 20, 30–33, 35, 55, 58

RL Reinforcement Learning. 4, 5, 17

RNN Recurrent Neural Network. 8, 12, 23

SA Self-Attention. 14, 15

Seq2Seq Sequence to Sequence. 12

SLAM Simultaneous Localization and Mapping. 4, 5

SPL Success weighted by (normalized inverse) Path Length. 20, 31, 34–37, 41, 42, 57–60, 65

SSL Self-supervised Learning. 15, 16

SSPL Soft Success Path Length. 20, 21, 31, 34–37, 41, 42, 57–60, 65

timm PyTorch Image Models. 30

ViT Vision Transformer. 1, 2, 9, 13–16, 23–25, 29, 30, 32–34, 36–40, 42, 60–64

VO Visual Odometry. 1, 2, 8, 9, 18, 22–25, 27–37, 39, 41–43, 64

VOT Visual Odometry Transformer. 2, 23, 25, 26, 30–43, 55, 57, 59–65

1. Introduction

Artificial Intelligence (AI) has found its way into many commercial products that simplify our lives by providing helpful services. However, most of its applications are still in the digital world revolving around retrieving and processing information. When talking about AI, most people still think of humanoid robots or other materializations of intelligence. Intending to make this fiction a reality, embodied AI aims to put intelligent programs into bodies that can move in the real world or interact with it. The most prominent representatives, *e.g.*, robot vacuum cleaners already found their way into our homes. Progress is still to be made before manipulators can help us clean, cook, or do laundry.

One of the most fundamental skills such embodied agents must learn is to effectively traverse the environment around them, allowing them to move past stationary tasks and provide services in multiple places [1]. Here, the ability of an agent to locate itself in an environment is vital to navigating it successfully [2, 3]. Anderson *et al.* [4] formulate this notion into a benchmark suite where an agent, equipped with an `RGB-D` camera and a `GPS+Compass` sensor, has to navigate to goals in an unseen environment. With extended data access through simulators [1, 5–8] and large-scale parallel training, recent approaches reached a tremendous 99.6% success rate on the specific task of PointGoal navigation [9]. However, the setup is still idealistic with noiseless observations and perfect actuation [10]. In a realistic setting, the success rate of this agent degrades to a mere 0.3% [3, 11].

The most significant change causing the agent to fail is the removal of `GPS+Compass` due to their inaccurate sensor readings in indoor environments [3]. Deploying a separate Visual Odometry (VO) model has shown to be beneficial when localizing the agent from visual observations only [2, 3]. These methods build on the advancements in deep learning to achieve impressive results. However, the underlying Convolution Neural Network (ConvNet) architecture assumes a constant channel size of the input. This reliance limits their flexibility and makes dealing with multiple modalities and privileged access to those sheer impossible.

For instance, consider the case of a failing sensor. If access to an input channel, *e.g.*, `Depth`, is not given or varies, *i.e.*, fails once in a while, it is unclear how the model should react. Should an additional model be trained to cover the edge case? Should the input be replaced with fixed values? Could this cause the model to fail catastrophically as it tries to make sense of the dummy input? While those questions are unanswered, they clearly show the need for a robust and flexible architecture capable of handling varying modalities and inconsistent access to those. As a bi-product, this also allows assessing the importance of the modalities for different design choices.

Furthermore, the focus in VO for embodied AI lies more on pushing the benchmarks than on understanding the underlying problem itself. Based on the findings that humans use global structures to localize and orient themselves, this work hypothesizes a global nature of the VO problem in indoor navigation caused by the large turning angles, which lead to enormous displacements of pixels in the image plane. If this assumption is proven, recent architectures based on ConvNets are ill-equipped to solve the task due to their locality bias. The recently proposed Vision Transformer (ViT) can exploit such global nature through its larger receptive field [12].

On the flip side, ViTs suffer from the absence of biases which sparks the need for a lot of training data which is difficult to obtain in VO settings [13–15]. This work investigates various supervised and unsupervised pre-training methods to deal with this requirement. It further hypothesizes that the action taken by the agent between two observations acts as a strong prior on its displacement. Injecting this additional information into the architecture could further stabilize training and reduce the data requirements.

To summarize, this work investigates whether the VO problem in embodied indoor navigation is global and whether the ViT architecture can exploit this property with its large receptive field. Furthermore, the roles of the input modalities for VO get explored and privileged access evaluated. This thesis presents the *Visual Odometry Transformer (VOT)*. It is a flexible model that can deal with multiple privileged modalities while having a large receptive field through the integrated attention mechanism. Furthermore, it uses supervised and unsupervised pre-training for the backbone and a practical action prior on the task to train on significantly fewer data than previous methods. Finally, the VOT achieves state-of-the-art performance on VO as a true swap-in for `GPS+Compass` in PointGoal navigation.

2. Related Work

2.1. Embodied Visual Navigation

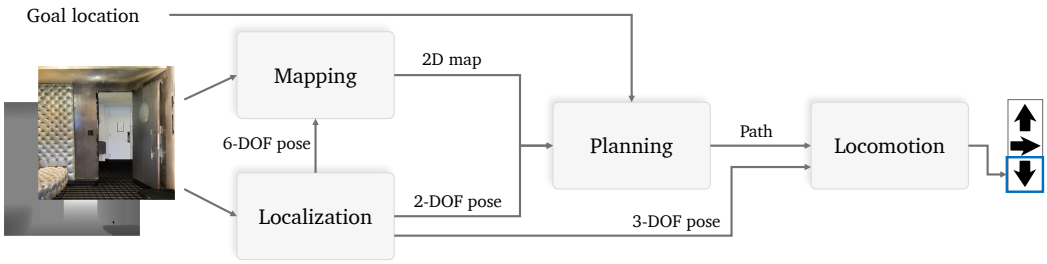
2.1.1. Embodied Artificial Intelligence

The rise of deep learning and the curation of larger datasets like ImageNet [16] has led to remarkable progress in Natural Language Processing (NLP) [17], Computer Vision (CV) [18], and various other fields related to Artificial Intelligence (AI). These advancements manifest themselves in superhuman performance on passive tasks where the model acts as an observer and does not interact with its environment. Working on static datasets, this branch of AI is often referred to as *Internet AI* due to the origin of the data [16, 19, 20]. Embodied AI aims to bridge the gap between virtual worlds and the real world by giving AI systems the capability to move in and interact with their environment, hence, embodiment. The resulting embodied agents can take many forms like mobile robots, *e.g.*, vacuum cleaners, manipulators, and combinations of both. While this paradigm shift allows embodied platforms in, *e.g.*, robotics, to leverage large-scale learning techniques, it can also benefit the perception aspect.

James J. Gibson, a psychologist of the 19th century, argues that in perceiving the world, acting in it is a crucial component [21]. For instance, imagine a person looking at a painting of a chair. If painted sufficiently realistic, there is no way of telling whether the object is a painting or a chair without moving to get an additional viewing angle of the scene. Obtaining such viewpoints is one of the benefits of examining perception and action combined. Furthermore, James J. Gibson believes that this relationship also holds in the other direction. Consider an animal searching for food. Without any means of perception, it is blind to its surroundings. Acting would not increase the chances of getting food but might even increase accidents, *e.g.*, falling off a cliff. Only through perception does action yield benefits. In this case, the animal could move towards food sources or avoid cliffs. To summarize, perception requires action, or else it is incomplete, *i.e.*, it misses out on information. This view on action-perception is still limited and referred to as *active perception*, *i.e.*, the agent can only act in the environment but cannot alter it. If the agent's capability extends to interacting with objects, *e.g.*, displacing them, the term evolves to *interactive perception* [22]. This setup is more challenging since the agent can uncover information through interaction that would've not been present by simple observation. Because object manipulation is still an active field of research, this work will focus on active perception instead.

2.1.2. SLAM- and Learning-based Navigation

Due to the limited action capability of an agent in the active perception setting, navigation emerged as a substantial task to solve and pave the way toward autonomous real-world interaction. To tackle this broad problem, various types of navigation tasks emerged. These tasks include navigating to goals [4], *e.g.*, objects [24], points, and areas, and embodied question answering, where the agent has to navigate to a goal



(a) SLAM-based pipeline for navigation split into mapping, localization, planning, and control (here: locomotion).

(b) Learning-based, end-to-end navigation using a deep (neural) network.

Figure 2.1.: Abstract SLAM- *vs.* learning-based navigation approaches. Figures from [23].

first before answering a question prompt [25, 26]. Even more complex tasks that require high-level reasoning require low-level navigation capability.

In the past, hand-crafted pipelines based on Simultaneous Localization and Mapping (SLAM) dominated navigation by splitting the task into the four components of mapping, localization, planning, and control [27]. First, a localization module estimates the agent's pose in the environment, while the mapping module simultaneously builds a 2D obstacle map. This map, represented as a grid, is created from `Depth` and contains probabilities of each cell occupied by an object, *i.e.*, the agent can not navigate through. In the second step, the planner plans a trajectory given the agent's pose, the 2D map, and a goal location relative to the agent. Finally, the controller takes the planned trajectory and outputs an action the agent takes to move towards the goal position. Figure 2.1 visualizes such a navigation pipeline and contrasts it to a learning-based solution. One downside of hand-crafted approaches is that they require extensive hand-engineering to find a set of hyperparameters that make the method perform well [23]. Furthermore, the system depends on `Depth` as input modality when building an occupancy map of the environment and localizing the agent. If it only receives `RGB` as input, it experiences catastrophic failure [23].

On the other hand, learning-based methods offer the potential to learn the navigation pipeline entirely from data and exploit its regularities. Figure 2.1b shows the general concept of a deep Reinforcement Learning (RL) agent for navigation. RL utilizes a reward signal to reinforce desired or penalize avoidable behavior and trains the policy with this signal. In the case of navigation, *e.g.*, the agent receives a terminal reward when it reaches the goal, a small reward whenever it moves towards it, and a penalty for taking any action, incentivizing reaching the goal with fewer steps. In contrast to supervised learning, unsupervised or self-supervised learning, RL describes learning from experience instead of a fixed dataset. The agent rather acts in the environment to collect its own dataset of observations and rewards.

For some time, hand-crafted approaches were superior, outperforming their learning-based counterparts [23, 28] by performing well in cluttered environments [23] and experiencing better collision avoidance [28].

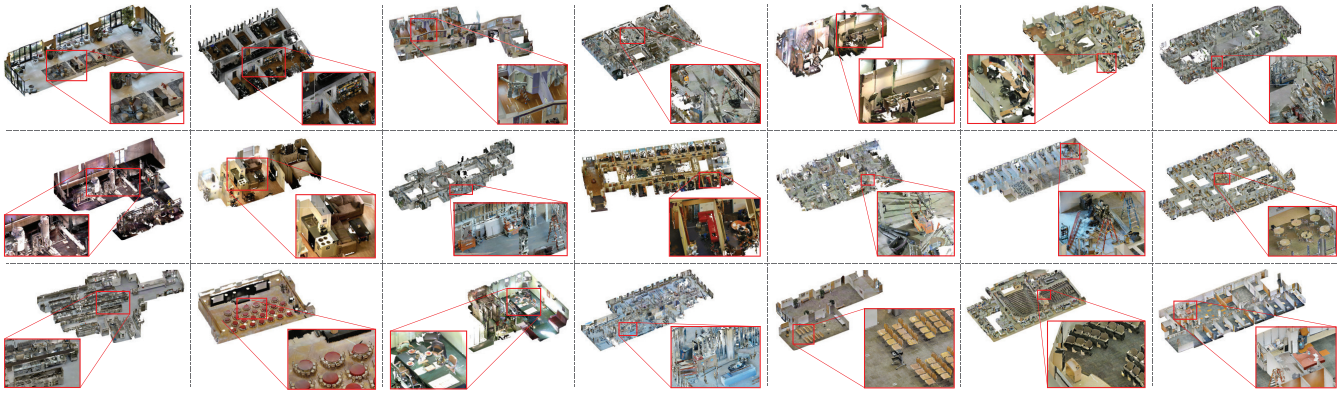


Figure 2.2.: Sample 3D models from the Gibson database. Note the difference in size, visual appearance, and domain (*e.g.* offices, living space, *etc.*). Figure from [36].

However, Savva *et al.* [1] discovered that scaling the amount of training data by an order of magnitude leads to opposing results. Furthermore, learning-based methods become more robust to ambiguity, noise, and limited sensor suites [23, 28].

While both methods have their advantages and drawbacks, learning-based methods nowadays dominate the field of indoor navigation. Because the search space of deep RL methods is considerable, and extensive data collection is required, combining both approaches is an active field of research and shows promising results [29–33]. Vice versa, deep learning has also recorded accomplishments in extending SLAM approaches [34, 35].

2.1.3. AI Habitat Simulator

The research community moved towards simulated environments instead of directly training agents in the real world to examine visual navigation tasks at a larger scale. For that matter, they developed several indoor simulators like *Habitat* [1, 5], *Gibson* [6], *MINOS* [7], and *AI2-THOR* [8] that can serve as environments to train navigation agents. In contrast to running experiments in the real world, those platforms provide scalability, speed, safety, and reproducibility [1]. On the flip side, however, agents can exploit irregularities, artifacts, and quirks of the simulators that make it difficult to transfer the behavior to the real world. This resulting performance difference is called *Sim2Real* gap [38]. To keep this gap small, the community turns to photo-realistic scans of 3D environments [8, 36, 39–41] that aim to represent the real world as accurately as possible. One of the most prominent simulators is AI Habitat (Habitat) by Savva *et al.* [1]. The project aims to standardize the embodied AI software stack (tasks, simulators, datasets) into a single accessible product. For this purpose, the developers include a configurable 3D simulator and generic 3D dataset support to allow maximum flexibility in the research process. At the start of every simulation, Habitat loads a 3D (indoor) scene, like the ones in Figure 2.2, from one of the previously mentioned datasets and places the agent at a predefined or random location in the scene. The agent is embodied as a $0.1m$ cylinder equipped with an `RGB` camera and a `Depth` sensor. Furthermore, access to `GPS+Compass` provides the agent's position and orientation relative to its initial location. The action space consists of move forward $0.25m$ (`fwd`), turn left (`left`), and turn right (`right`) by $30°$ each. Additionally, `stop` signalizes that the agent has reached the goal. In the standard case, `GPS+Compass` is sufficient to determine the end of an episode. However, in its absence, the agent has to decide whether it has reached the goal, requiring the `stop` action. Figure 2.3 provides

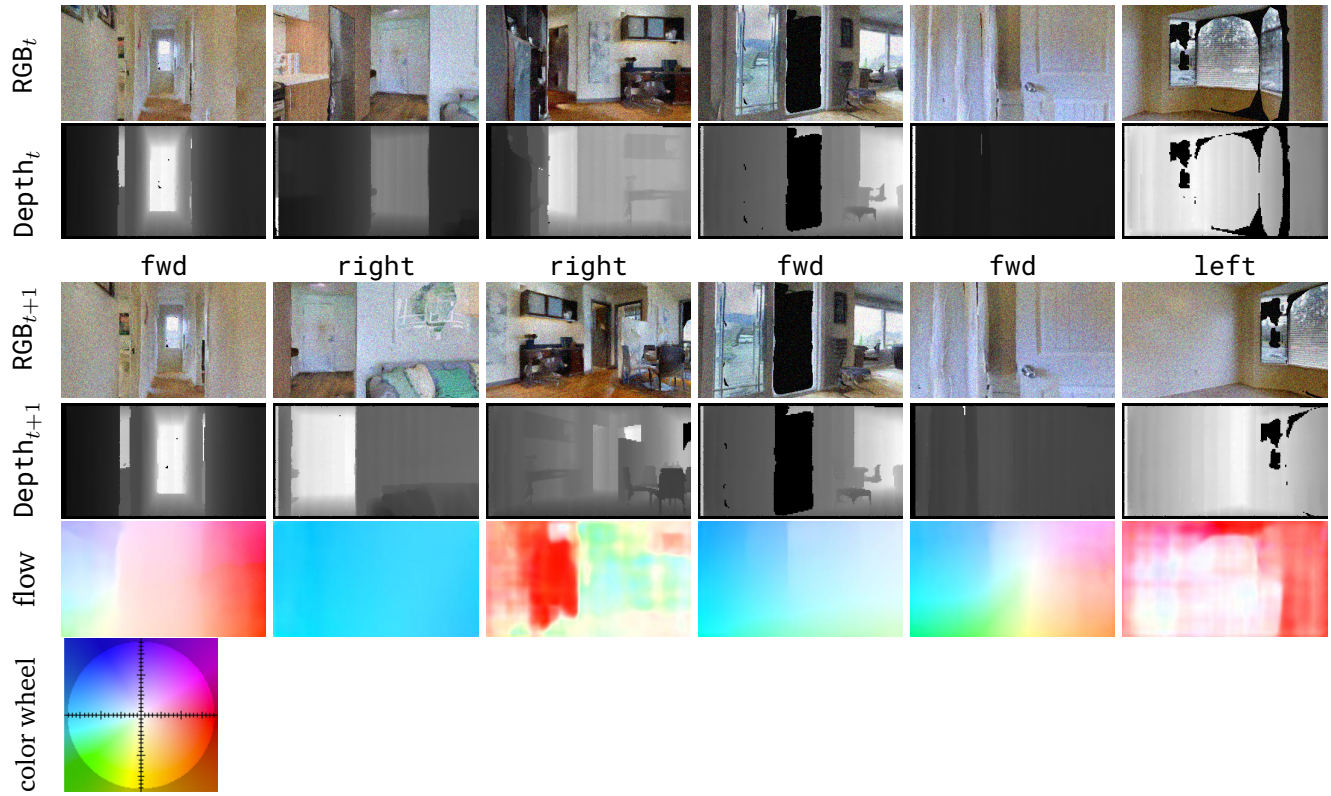


Figure 2.3.: Observations (RGB, `Depth`) and action taken by the agent. Samples from different 3D scene of the Gibson-4+ dataset loaded in AI Habitat (Habitat). Notice that even in the revised Gibson dataset, the scans in the 2nd and 5th column experience artifacts in both RGB, and `Depth`. Optical flow field color coding from [37].

example observations and actions from such an agent. This configuration is exemplary, and sensor and action space are further configurable, which allows for additional observations and actions.

2.1.4. Realistic PointGoal Navigation

In the *PointGoal* Navigation task by Anderson *et al.* [4], an agent spawns at a random position in an unseen environment and uses a set of discrete actions (`fwd`,`left`,`right`,`stop`) to navigate to a goal defined *w.r.t.* to its starting position and orientation. Wijmans *et al.* [9] achieved near-perfect results of 99.6% success in this idealized setting. However, the authors assume noiseless egocentric vision (noise-free RGB camera and `Depth` sensors), noise-free actuation (`fwd` moves the agent forward by exactly $0.25m$, `left` turns the agent by exactly $30°$, *etc.*), and perfect localization via `GPS+Compass`. These assumptions are unrealistic since the real-world actuation is inherently noisy due to varying surface friction, perception assumes consistent lighting conditions, and GPS sensors are less accurate in indoor environments [3]. Finally, as stated in [1], Habitat allows for *sliding*, *i.e.*, the agent can slide alongside walls when colliding with such, perturbing the expected outcome position. Therefore, Murali *et al.* [10] updated the original task definition to bring the setup closer to reality. The authors first propose to include noise models for actuation to simulate the real-world perturbations of the action space. Second, they add RGB and `Depth` noise models from Choi *et al.* [43] to simulate real-world cameras. Finally, and most importantly, they removed the `GPS+Compass` such that the agent only has access

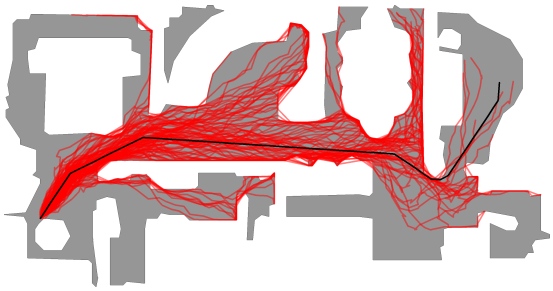


Figure 2.4.: Actuation noise model by Murali *et al.* [10]. Black trajectory indicates perfect actuation. Red trajectories are different rollouts of the same policy under actuation noise. Figure from [42].

visual sensors. Including these proposals into the Habitat environment and fine-tuning the parameters has shown to produce a setting better predictive of real-world performance [42]. Improvements in the simulation, therefore, better translate to the real world and reduce the Sim2Real gap.

With the increased difficulty of the task, the near-perfect policy trained on the idealized case deteriorates to a mere 0.3% success rate [11], which makes intuitive sense because noisy actuation makes navigation a stochastic process where the previous perfect actions rarely lead to the goal. Figure 2.4 visualizes the effects of the actuation noise on the resulting trajectory. Furthermore, the policy trained on noise-free `RGB` and `Depth` now has to deal with additional noise in the input to decide which action to take. Finally, deciding to call the `stop` action can no longer be done by comparing its current position with the true relative goal position but an estimate of it.

To facilitate progress in such a realistic PointGoal Navigation setting, the Habitat team hosts yearly challenges like the most recent *Habitat PointNav Challenge 2020/2021* [38]. Participants train agents on the training split from the Gibson Database of 3D Spaces [36]. The agents are then evaluated on an evaluation split to determine the one that generalizes best to the unseen environments. Figure 2.2 shows examples of the spaces contained in the dataset. Because not all scans are of the same quality, the challenge further focuses on *Gibson-4+*[1], a subset of 106 scenes from the initial 572. This new dataset only contains scans with a rating of 4 or higher, indicating scans with the highest reconstruction quality, further facilitating realism and reducing possible exploits through unrealistic imagery.

2.1.5. Camera Pose Estimation and Visual Odometry

Traditionally, the task of pose estimation, *i.e.*, recovering the absolute camera pose, was tackled by searching 2D-3D correspondences in the image [44, 45]. Finding those requires a 2D image and a 3D reconstruction of the scene. Points in both representations are mapped to a shared feature space. The features can either be hand-crafted, *e.g.*, *SIFT* [46] or *SURF* [47], or be learned, *e.g.*, *SuperPoint* [48]. Subsequently, candidate poses are computed from the correspondences, and the best one is selected and refined [49–51]. Searching for correspondences is computationally expensive, and access to both 2D images and 3D scenes is not always given. Nonetheless, this strategy still observes competitive results when combined with deep learning approaches for feature extraction [52].

As one of the first, Kendall *et al.* [15] proposed *PoseNet*, an approach to regress camera pose in an end-to-end fashion using a Convolution Neural Network (ConvNet) architecture. Their model consists of a modified truncated *GoogLeNet* [53], which they pre-train on an image classification task to deal with the absence of large-scale training data for pose estimation. In general, ground truth acquisition still poses a challenge, and

the small amount of training data leads to most approaches overfitting it [3, 15, 54]. The introduction of a temporal dimension then prepares the task based on static scenes for an interactive setting where Visual Odometry (VO) deals with estimating the relative change between two poses from images. While pose estimation only involves a single view, VO can be divided into stereo and monocular estimation. Most methods nowadays focus on monocular estimation because a single camera is cheaper, lighter, and more widespread on end-user devices than a stereo rig [55]. When the ratio of stereo baselines to `Depth` is minimal, the stereo case of VO degenerates to the monocular one [55]. At this juncture, the following will assume a monocular VO and point out stereo cases if those arise.

Inspired by PoseNet, many VO architectures move towards ConvNets for VO [56, 57]. One of those deep learning-based architectures is *VLocNet* [54]. This network leverages the inherent similarities across pose and relative pose change to learn an auxiliary task. The authors use weight sharing and argue that it facilitates a constraint on the search space as both encoders have to be consistent *w.r.t.* each other. Additionally, the auxiliary task acts as regularization, avoiding overfitting. Building on this line of work, *VLocNet++* [58] extends it by introducing semantic segmentation. Partially sharing weights between all three tasks allows the network to make even more consistent predictions taking into account spatial and semantic information about the scene. While using auxiliary learning of the pose might benefit pose change accuracy, pose change is sufficient to solve a navigation task.

DeepVO [59] and *ESP-VO* [60] further exploit the temporal aspect of the VO problem by learning from videos. Their models deploy a Recurrent Neural Network (RNN) to learn the motion dynamics and relations of the sequence. Similar to [15], they also use a pre-trained architecture, the *FlowNet* [61].

Most recently, Zhu *et al.* [55] observed that the optical flow fields of navigation sequences differ *w.r.t.* each quadrant, and pixel movements become more intense towards the edges. These differences particularly appear in the car trajectories of the *KITTI* [62] dataset, where turning angles are relatively low between two frames. The authors propose *DeepAVO*. This architecture first computes the optical flow between two input images using a *PWC-Net* [63] and subsequently splits it into four quadrants. Then, it processes each quadrant with a ConvNet backbone and concatenates the output to compute the final pose change. Each backbone also includes several Convolutional Block Attention Modules (CBAMs) [64] that allow the model to decrease the focus on the foreground and blurred parts of the image, as they are irrelevant for the pose change and focus on pixels in distinct motion. This assumption only holds for `fwd` actions in indoor navigation tasks. When turning, most of the optical flow is occupied by a single direction. Furthermore, this suggests that VO is indeed a global problem as as all pixels get shifted by a significant margin. Figure 2.3 visualizes the optical flow for some observation pairs, computed using the *PyTorch* implementation [65] of *PWC-Net* [63].

2.1.6. Visual Odometry for Realistic PointGoal Navigation

In realistic PointGoal navigation, the lack of `GPS+Compass` sparks the need for some form of localization. As the agent receives a relative goal position, it has to compute its own relative pose change to update the goal position accordingly. In an indoor environment, subsequent frames share almost no overlap and experience far more occlusions due to the large translation and rotation caused by the actions. This circumstance drastically reduces the effectiveness of multiple input frames [3]. Datta *et al.* [2], therefore, propose a VO model to estimate the egomotion, *i.e.*, pose change of the agent from consecutive frames. Even though noisy, this egomotion estimate is sufficient to update the relative position and reach the goal. The authors decouple learning the system dynamics via VO from the task-specific navigation policy allowing them to retrain the modules when dynamics change or actuation experiences noise.

Zhao *et al.* [3] justify end-to-end approaches by showing that classic feature-based methods fail for realistic settings where observation noise is present. They mainly follow Datta *et al.* [2] but focus on improving its robustness to actuation and observation noise. Their proposal includes geometric invariance losses similar to [66] that ensure rotation and translation invariance. Additionally, they train separate models for moving (`fwd`) and turning (`left,right`). Finally, they deploy Dropout [67] to mimic the behavior of an ensemble of models that deal with uncertainty in the egomotion estimate. Additional pre-processing of the `Depth` input via discretization and projection into a top-down view then handle the noisy observations and induce bias.

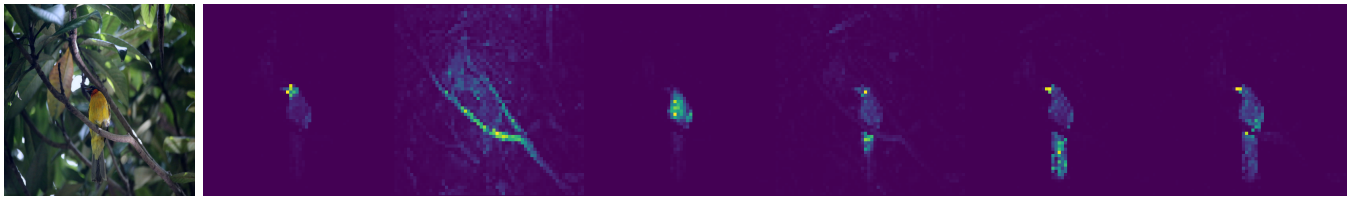
Both presented methods display their VO model as a swap-in component, *i.e.*, not requiring re-training the navigation policy trained on ground truth localization. However, the resulting agent still does not reach the original performance suggesting that the VO model is not yet good enough to be a complete replacement. Furthermore, using different models for moving forward and turning leads to a static approach. It is still unclear whether a single model for all actions could leverage the interplay between them and allow for a broader application that is not limited to discrete action spaces. Finally, the heavy reliance on `Depth` of robust VO cause problems in some real-world applications, *e.g.*, rescue operations. If the `Depth` sensor fails and no redundant system is present, an estimation using solely `RGB` data could lead to a much more robust system.

A general direction of recent approaches is also to use more data made available through simulation (72k [2], 1M [3], 2M [68]). While this presents a logical progression, it appears to be infeasible to re-train the VO module whenever dynamics or sensor configurations change due to the computational effort.

2.2. The Transformer

2.2.1. Attention Mechanism

The Transformer architecture relies on the attention mechanism, which re-appears throughout all its components Vaswani *et al.* [70]. It is a learnable layer that in contrast to the local information aggregation of ConvNets, aggregates information globally, *i.e.*, across the whole input at once [71]. Informally, attention can be described best in the example of human attention. Assume the task of classifying the object in Figure 2.5a. Humans scan the image for distinctive features, which we then assign to the corresponding class [72]. For instance, in the given image, we identify wings, head, beak, and tail, which leads to detecting a bird. The object also sits on a tree branch, further supporting our decision. We discard the rest of the image since it



(a) Input image. (b) Individual attention maps of each attention head.

Figure 2.5.: Attention maps were computed with a ViT-S pre-trained on ImageNet with the DINO method and a patch size of 8×8. As proposed in [69], the original image Figure 2.5a is passed through the ViT, and the last attention layer visualized. Attention maps corresponding to the individual attention heads are provided. Brighter colors indicate a higher weighting of the patches by the Vision Transformer (ViT).

does not provide useful information to solve the task [72]. The attention mechanism behaves quite similarly. First, the image is split into patches, embedded into *tokens*, and encoded into a set of *key-value* pairs [13]. The mechanism then uses a *query* to access this encoding, similar to looking for relevant information in the image. Because deciding which patches of the image are relevant for the task is nontrivial and highly dependent on the downstream task, constructing the key-value pairs and the query is usually learned in an end-to-end fashion [70]. Inspired by Caron *et al.* [69], Figure 2.5 shows so-called attention maps for the described example, giving an idea of what the model focuses on in the image. Notice the overlap with human attention. How to create those maps will be discussed later. While this example focuses on an image, the attention mechanism first got popular in NLP [70]. In this domain, tokens do not represent image patches but words, characters, *etc.* [73]

Formally, an attention function is defined as the mapping from a query and a set of key-value pairs to an output [70]. Query $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$, and values $V \in \mathbb{R}^{n \times d_v}$ are obtained by a linear transformation of an input matrix $X \in \mathbb{R}^{n \times m}$, containing n input tokens of dimensionality m. The corresponding transformation matrices are $W_Q, W_K \in \mathbb{R}^{m \times d_k}$, and $W_V \in \mathbb{R}^{m \times d_v}$, leading to the following equations:

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \end{aligned} \tag{2.1}$$

Inside the attention function, a weighting $A \in \mathbb{R}^{n \times n}$ of the values is computed from query and keys via a compatibility function. Vaswani *et al.* [70] propose to use the *dot-product*, defining the function as

$$\text{compatibility}(Y, Z) = \text{softmax}\left(YZ^T\right) \tag{2.2}$$

resulting in the following attention formulation

$$\begin{aligned} A &= \text{compatibility}\left(Q, K\right) \\ \text{Attention}\left(Q, K, V\right) &= AV \end{aligned} \tag{2.3}$$

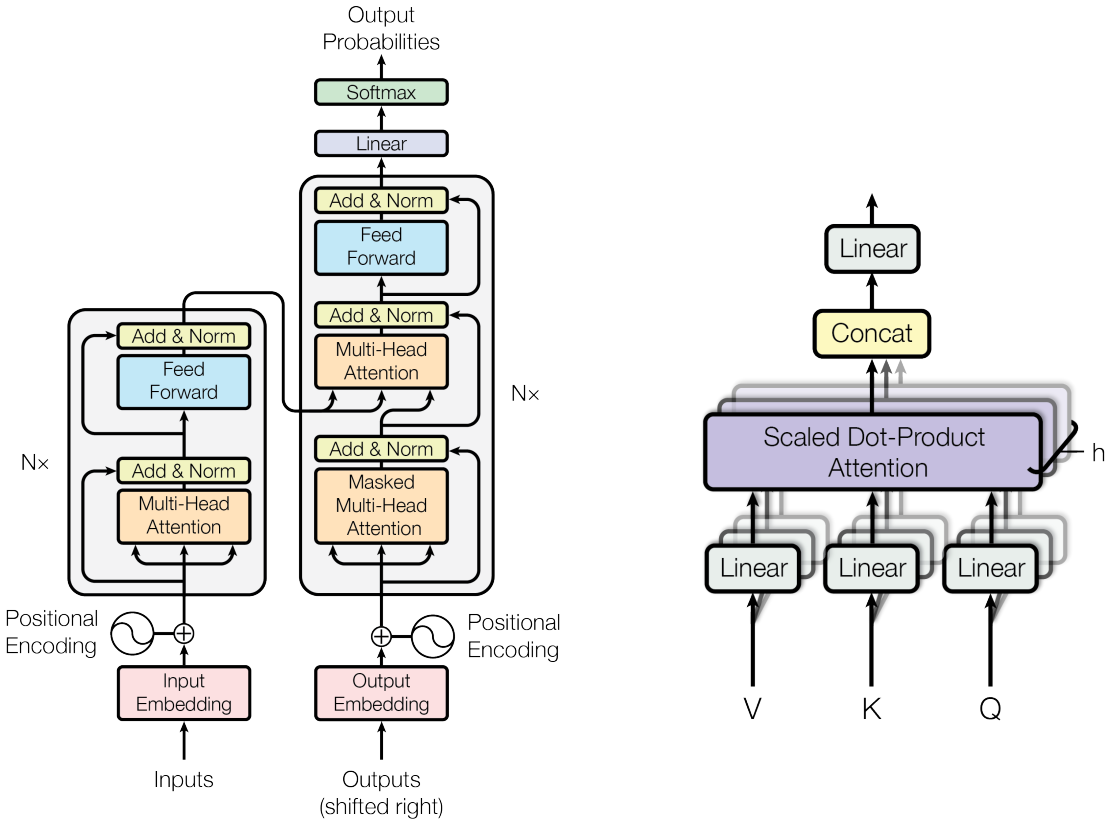
The attention weights A then contain the bilateral *compatibility* of each token with one another. Multiplying A with value matrix V results in the final output of the attention mechanism. A *softmax* over the last dimension of A ensures the weights sum up to 1 before multiplying them with the values to avoid exploding values.

The most common compatibility functions are additive attention and dot-product attention [70]. Vaswani *et al.* [70] stress that while both are similar in theoretical complexity, they differ in practice, with dot-product attention being much faster and more space-efficient due to its reliance on matrix multiplications which are highly optimized today and run on GPUs. Additionally, the authors introduce a scaling factor $\frac{1}{\sqrt{d_k}}$ to the dot-product that prevents small gradients for large d_k, which they found to be an issue in their work.

Putting it all together, the attention mechanism as proposed by Vaswani *et al.* [70] is as follows:

$$\text{Attention}\left(Q, K, V\right) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.4}$$

In this attention formulation, Q, K, V are all computed from linear projections of X. This particular case is called Self-Attention and is used to encode a sequence by letting its tokens attend to each other. However, the transformer also features attention mechanisms where query and key-value pairs get computed from different inputs. This case strikes more intuitively as producing an external query to select information from an internal key-value representation.



(a) Transformer architecture.

(b) Multi-head Attention

Figure 2.6.: a) The Transformer as presented by Vaswani *et al.* [70]. The input sequence is embedded, a positional encoding added, and passed into the encoder (Figure 2.6a left). The produced encoding acts as additional input for the decoder that processes the previous output sequence (Figure 2.6a right). Finally, a linear layer and a *softmax* compute the next token in the output sequence. Here, N denotes the number of encoder and decoder blocks.
b) Hyperparameter h denotes the number of attention heads in each Multi-head Attention layer. The inputs V, K, Q represent values, keys, and query. Figures from [70].

2.2.2. Multi-head Attention

Multi-head Attention (MHA) extends the concept of attention to multiple attention heads, *i.e.*, parallel attention mechanisms, that allow the model to attend to different token positions simultaneously [70]. While the attention weights of a single token can get dominated by obvious key-value selections, *e.g.*, itself, a model with multiple randomly initialized heads may provide a more comprehensive output by putting attention on a more diverse token selection. Further, applying learnable linear projections to W_Q, W_K, W_V allows the model to attend to information in different representation subspaces. Figure 2.6b visualizes the structure of MHA.

Following Vaswani *et al.* [70], $Q, K, V \in \mathbb{R}^{n \times d_{model}}$ are redefined with d_{model} the dimensionalities after the linear projection into the subspace. The corresponding transformation matrices are $P_Q^{(i)}, P_K^{(i)} \in \mathbb{R}^{d_{model} \times d_k}$ and $P_V^{(i)} \in \mathbb{R}^{d_{model} \times d_v}$ with d_k, d_v the previous dimensionalities. Every attention head i has its own transformation matrices with $i \in [0, 1, ..., I-1]$ and I the total number of attention heads. A final linear projection $F \in \mathbb{R}^{hd_v \times d_{model}}$ transforms the concatenated outputs of the attention heads to the output size of a single head. The

authors suggest choosing $d_k = d_v = d_{model}/h$ with h the number of parallel attention heads to compensate for the additional computational cost. In the case of Multi-head Self-Attention (MHSA), *i.e.*, Multi-head Attention with Self-Attention, the mapping of input X looks like this:

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \\ \text{MultiHead}\,(Q,K,V) &= \text{concatenate}\left(head_0, ..., head_{(I-1)}\right) F \\ \text{where} \quad head_i &= \text{Attention}\left(QP_Q^{(i)}, KP_K^{(i)}, VP_V^{(i)}\right) \end{aligned} \tag{2.5}$$

An interesting side-effect of the attention mechanism is that it can be used to interpret incomprehensible model behavior [71]. As shown by Caron *et al.* [69], the attention weights A from Equation (2.3) can contain information on how much the model attends to each token to arrive at its final decision. Fitting those to the image produces an *attention map* visualizing how much each token is weighted by the model. Figure 2.5 shows the input image and the corresponding attention maps for their proposed DINO method.

2.2.3. Attention Is All You Need

Language is inherently sequential, *i.e.*, it consists of a sequence of elements like characters, words, sentences, paragraphs, sections, *etc.* A subset of language tasks, *e.g.*, translation and question answering, are Sequence to Sequence (Seq2Seq) tasks where both the input and output of the model consist of a sequence of, *e.g.*, words or characters [74]. Historically, researchers have used RNNs to model such Seq2Seq tasks [70]. Tragically, those networks suffer from vanishing gradients with increasing sequence lengths, making them ill-equipped to learn long-range dependencies required to solve most language tasks [75]. Hochreiter *et al.* [75] proposed the Long Short-Term Memory (LSTM) that maintains an explicit memory to store information on those dependencies and solves the vanishing gradient problem. However, those units still do not completely eliminate the information loss, especially when sequences become even longer [70]. Aiming to solve this forgetting, Vaswani *et al.* [70] present the Transformer architecture, which utilizes an attention mechanism over all input elements at once, circumventing the need for explicit memory. Figure 2.6a visualizes this architecture composed of stacked encoder-decoder structures.

In the first step, the sequence elements are embedded into tokens and passed to the encoder. The encoder consists of multiple stacks that deploy MHSA-layers to compute a lower-dimensional representation of the input sequence. This representation serves as an input to the decoder stack, which computes a set of key-value pairs from it. The decoder then creates a query from its previous predictions utilizing MHSA-layer to retrieve information from the encoder output in a MHA-layer. With the final output of the encoder stack, a linear layer then predicts the next token of the output sequence. Since the decoder only predicts a single output token at every pass, multiple forward passes are required to produce the final output sequence.

Tokenizing the input sequence removes the information about its element's order. In contrast to recurrent architectures or convolutions that explicitly model the position of the input tokens, the attention mechanism presented in Equation (2.4) is *permutation invariant*, *i.e.*, any permutation of the input sequence produces the same output [70, 76]. Therefore, the Transformer does not know anything about the original sequence order. As some tasks require this information, Vaswani *et al.* [70] propose an inductive bias in the form of a positional encoding that they add to each token. This encoding can either be fixed or learned and helps the model distinguish the token's positions [77]. The presented fixed embedding consists of sine and cosine

functions of different frequencies. Plugging in the position of the token, an embedding is computed for each token dimension. The authors hypothesize that the model can learn to use this embedding to identify a token's position. Due to the nature of the sine and cosine functions, it is further assumed that the model can extrapolate to longer sequence lengths than seen during training. In contrast to a fixed function, a positional embedding can also be learned. Dosovitskiy *et al.* [13] propose a learnable 1D embedding and utilize 2D interpolation to extrapolate to longer sequence lengths at test time. However, it is unclear which positional embedding to use, and empirical studies show that the positional information is not fully exploited yet [78].

While the Transformer was presented for sequence modeling and applied to language translation [70], its capabilities reach beyond that. For instance, Bidirectional Encoder Representations from Transformers (BERT) [17] uses an encoder-based Transformer to learn representations that are useful for a variety of downstream tasks. To achieve such general-purpose language representation, the authors pre-train BERT with a Masked Language Model (MLM) objective and Next Sentence Prediction (NSP). The former describes masking out past and future tokens in the input sequence and asking the model to predict those. While some language tasks go beyond single tokens, the NSP objective learns relationships between sentences. As this objective is not directly associated with a specific input token, an empty token, the so-called Classification Token ($[CLS]$), is passed to the model to extract the desired information. After processing the input sequence, a Multi-layer Perceptron (MLP) takes the $[CLS]$ token as input to solve NSP during pre-training and downstream tasks during fine-tuning. By design of the attention mechanism, the $[CLS]$ token aggregates information from all other tokens and lets it serve as a rich representation for task transfer.

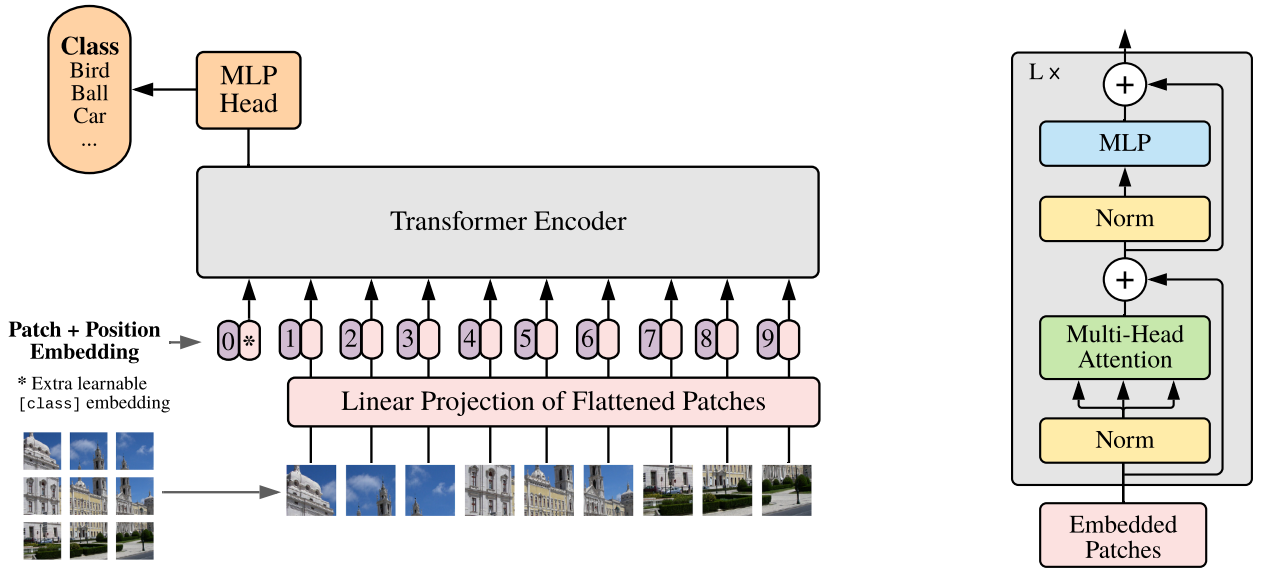
2.3. The Vision Transformer

2.3.1. An Image is Worth 16x16 Words

Most recently, Dosovitskiy *et al.* [13] introduced the Vision Transformer (ViT), a Transformer architecture for CV. The authors interpret images as sequences by dividing them into fixed-sized patches (proposed 16×16 pixels) and encode those into 1D token representations. In practice, their architecture follows Vaswani *et al.* [70] to a great extent, which manifests itself in the similar encoder block architecture (*cf.* Figure 2.6a and Figure 2.7b). As visualized in Figure 2.7a, they first tokenize the images via a flatten operation followed by a linear projection. Then, they encode each patch with a stack of encoders whose architecture is similar to the one in Figure 2.6a. Note that in contrast to Vaswani *et al.* [70], the layer normalization [79] is moved from following to preceding the MHSA and MLP as part of a design choice. The authors utilize an additional $[CLS]$ token that aggregates information into a representation that they then pass into an MLP to solve an image classification task. The learned $[CLS]$ token serves as a rich representation for transfer to new tasks.

The computation of ViTs scales quadratically with the sequence length due to the dot-product in the compatibility function (*cf.* Equation (2.2)) and, hence, inverse proportional to the square of the patch size. This means that the smaller the patch size, the higher the computational cost due to a longer sequence, and vice versa.

Since ViTs lack the inductive biases of ConvNets, they require vast amounts of training data to show sufficient generalization capabilities [13]. Because such data is rarely available, heavy augmentation and regularization techniques are often used in practice [14]. Using ConvNets to encode the image patches has also been shown to introduce some inductive bias to the architecture [13, 80]. Another approach to circumvent excessive data requirements is the teacher-student strategy proposed by Touvron *et al.* [81], which uses a ConvNet as a



(a) ViT architecture.

(b) Encoder block of a ViT.

Figure 2.7.: a) The Vision Transformer as presented by Dosovitskiy *et al.* [13]. Each input image is split into patches, flattened, and projected towards a specified token dimension. Similar to the Transformer, a positional embedding is added before passing the tokens to the encoder that produces a representation for each token via MHSA.
b) Here, L denotes the number of encoder blocks. The encoding of a special $[CLS]$ token (here: $[class]$) acts as condensed representation used to compute the final prediction through an MLP head. Figures from [13].

teacher and a ViT that tries to mimic its predictions. Finally, introducing a locality bias to the ViT allows for training on smaller datasets [82].

In large data regimes, however, ViTs outperform the classic Residual Networks (ResNets) [18] in a classification task [13, 83] on the open source ImageNet-1k (IN-1k) (1.4M images, 1k classes) [16], ImageNet-21k (IN-21k) (14M images, 21k classes) [16] and Google in-house datasets JFT-300M (300M images, 18k classes) [84], JFT-3B (3B images, 30k classes) [83]. ViT further achieved state-of-the-art results on various tasks like semantic segmentation [85, 86], instance segmentation [87], panoptic segmentation [76], object detection [76, 87], and dense prediction [86] like, *e.g.*, surface normal [88] and monocular `Depth` prediction.

2.3.2. Comparison to Convolution Neural Networks

However, many argue that the success of ViTs can not be attributed directly to the Self-Attention (SA) operation but the advancements in training settings and architecture details. Liu *et al.* [89] modify the original ResNet architecture to represent recent findings, and Wightman *et al.* [90] present a more demanding training setting. Both approaches significantly increase the performance of ResNet-based architectures on ImageNet, and the former even achieves state-of-the-art results. Tolstikhin *et al.* [91] present another approach that replaces convolutions and SA operations with MLPs. Their method achieves competitive results to the state-of-the-art in similar training settings and computational investment.

Most recent works have investigated the impact of the ViT's building blocks and compared it to classic ConvNet-based approaches. In contrast to the attention mechanism, the convolution operation is *translation equivariant*, which means that mapping an input to an output preserves the algebraic structure of the transformation. Put differently, mapping a translated input also produces a translated output. This property can also be seen as a result of the shared weights of ConvNets [92]. Another inductive bias of ConvNets is the *locality* caused by the convolutional operation, which only takes into account pixels in proximity to one another. Interestingly, Cordonnier *et al.* [93] show that the attention layer can learn convolution-like operations and, in practice, actually tends to. Park *et al.* [94] find that convolutions and MHSAs complement each other representing high-pass and low-pass filters, respectively.

Another difference is, that ViTs process the entire image at every layer and, therefore, keep the internal representations at the same size. This characteristic has shown to be beneficial for dense prediction tasks [86].

Because ViTs scale computationally poorly with larger image sizes, Liu *et al.* [87] propose a shifting window approach. By limiting SA to non-overlapping local windows, they achieve linear complexity *w.r.t.* to the image size. The model can further act as a drop-in replacement for ResNet backbones due to its adapted feature resolutions. Finally, Raghu *et al.* [12] compare the receptive fields and find that ViTs attend to global and local information in lower layers. In contrast, ConvNets are hardcoded to attend locally [12].

2.3.3. Transfer Learning

Transfer learning deals with transferring knowledge gained from solving one task to another [95]. In the domain of CV, this manifests in training a model on one task, *e.g.*, surface normal prediction, and transferring it to another, *e.g.*, depth estimation. The intuition behind this transfer is that both tasks share structure, and solving one can help solve the other [95]. Zamir *et al.* [95] map out those relationships between various perceptual tasks. Using a pre-trained model as initialization to learn a new task can reduce training time and the need for large datasets. One can interpret this practice as initializing the model closer to a minima of the loss surface. Due to that reason, it is common practice to fine-tune the pre-trained model, *i.e.*, training with a lower learning rate then used in the initial training to avoid negating the benefits gained from pre-training but utilize additional data from the target task. A common approach is to train a classification network, *e.g.*, a ResNet or a ViT, on supervised IN-21k data. To transfer the learned representations, the prediction head is replaced by a task-dependent network of the new downstream task. Subsequently, the full network can be fine-tuned to solve the new unseen task or unseen dataset.

However, pre-training in a supervised fashion trades off the need for labeled data of one task to another. Self-supervised Learning (SSL) mitigates this tradeoff by learning transferable representations from unlabeled data. Earlier methods focused on hand-designed proxy tasks such as image colorization [96], rotation prediction [97], solving a jigsaw puzzle [98], and various others [56, 99, 100]. More recent techniques rely on contrastive loss formulation [101] and augmentation methods [102–106]. These approaches force the learned representation to be similar for augmented views of the same image (positive examples) and dissimilar for different images (negative examples). The negative examples are crucial to avoid a representation collapse to a single point, *i.e.*, the model converging to the trivial solution of a single representation for all inputs. However, it lacks clear justification or interpretation [107]. Therefore, Bardes *et al.* [107], Grill *et al.* [108], and Zbontar *et al.* [109] propose practices to circumvent negative samples while showing similar performance to contrastive methods.

Caron *et al.* [69] propose a self-supervised teacher-student approach named Self-DIstillation and NO Labels (DINO). They demonstrate that the features learned by a ViT through their method contain explicit information

about the segmentation of an image (*cf.* Figure 2.5b). In contrast, training a ViT in a supervised fashion or using a ConvNet results in less salient features. While DINO is architecture agnostic, it synergizes well with ViTs and can help them reach competitive accuracy on ImageNet.

To evaluate representations learned with SSL, the community turns to linear probing. This technique involves training a linear classifier or a K-nearest Neighbors (KNN) on the representations. The accuracy on a validation set then determines their quality. While most works focus on ImageNet, others explore the concept of linear probing to evaluate the explainability of representations [110].

2.3.4. Multi-Modality

The previous approaches learn their representations from RGB images. However, the availability of multi-modal data, *e.g.*, depth, video, and audio, poses an opportunity to utilize additional information to learn richer representations. The ability to process a token sequence of arbitrary length allows the Transformer to digest any modality as long as the tokens have the same dimensionality. Jaegle *et al.* [111] push this to the extreme, presenting a general-purpose architecture that can handle data from arbitrary modalities. Likhosherstov *et al.* [112] explore this idea using video, audio, and images to co-train their models. They propose to train a model on single or multiple input modalities solving multiple downstream tasks. Girdhar *et al.* [113] propose to use multiple input modalities to solve a classification task. The model trains jointly on images, videos, and single-view 3D geometry. The learned representations generalize well across modalities and reach state-of-the-art results after fine-tuning on new tasks.

Finally, Bachmann *et al.* [114] extend the Masked Autoencoder (MAE) [115] to multi-modal, *i.e.*, multiple input modalities, and multi-task, *i.e.*, multiple output tasks. MAEs aim to solve a reconstruction task from a masked version of the input image. A standard procedure in training a ViT is to split images into patches. Learning to reconstruct the original image from only a low proportion of those (*e.g.*, 25%) poses a nontrivial and meaningful SSL task that requires the model to learn something about the underlying image content. Due to the quadratic complexity *w.r.t.* the input sequence length, discarding the masked input tokens further allows for a lightweight encoder than would be required if processing all patches. Combining a nontrivial but meaningful task with the lightweight encoder structure reduces training time significantly while producing representations that outperform those learned in a supervised fashion on ImageNet. Bachmann *et al.* [114] explore training MAE by reconstructing multiple input domains. They focus on RGB, depth estimation, and semantic segmentation. The impressive progress of dense prediction using Transformers [86], advancements in the dataset construction [88], and training procedure [116] for monocular depth estimation provide remarkably accurate and reliable estimates. The authors, therefore, propose to utilize pseudo labels, *i.e.*, predictions from a model, as a replacement for when the ground truth of an input modality is not available. Such a supplement works surprisingly well and allows large-scale training of such multi-modal models on, *e.g.*, ImageNet, which only provides RGB images. The proposed Multi-modal Multi-task Masked Autoencoder (MultiMAE) showcases strong transfer performance when additional input modalities are available through ground truth, *i.e.*, sensor readings or pseudo labels. Ablations with sensory depth suggest even higher performance gains than pseudo labeling, especially in contrast to models solely trained on RGB. When no RGB is available, MultiMAEs still perform well, showcasing an invariance to the input modalities. An overlooked application of their work poses the domain of embodied AI and robotics, where it is common to rely on depth sensors or RGB. The findings suggest 1) using estimated depth estimation in case sensors are privileged, *i.e.*, not always available, malfunctioning, or are turned off to save power, and 2) utilizing multi-modal input to extract better representations.

3. Methodology

3.1. Preliminaries

3.1.1. Problem Formulation

As discussed in Section 2.1.2, the task of point goal navigation can be formulated as a Reinforcement Learning (RL) problem. This section largely follows the definitions of Memmel *et al.* [117] and extends them to the PointGoal navigation task. First, the environment is modeled as a Markov Decision Process (MDP) defined as a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is a set of conditional transition probabilities between states, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. A property of an MDP is that each state s_{t+1} is conditionally independent of all previous states and actions given a_t, s_t, such that the state transitions satisfy the *Markov property*. However, in the navigation task, the observation the agent receives, *e.g.*, `RGB` and `Depth`, does *not* fully specify the underlying state. We, therefore, extend the MDP tuple by observation space Ω, and observation kernel $\mathcal{O} : S \times A \to \Omega$ that maps states and actions to observations. The resulting 7-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma \rangle$ then defines a Partially Observable Markov Decision Process (POMDP).

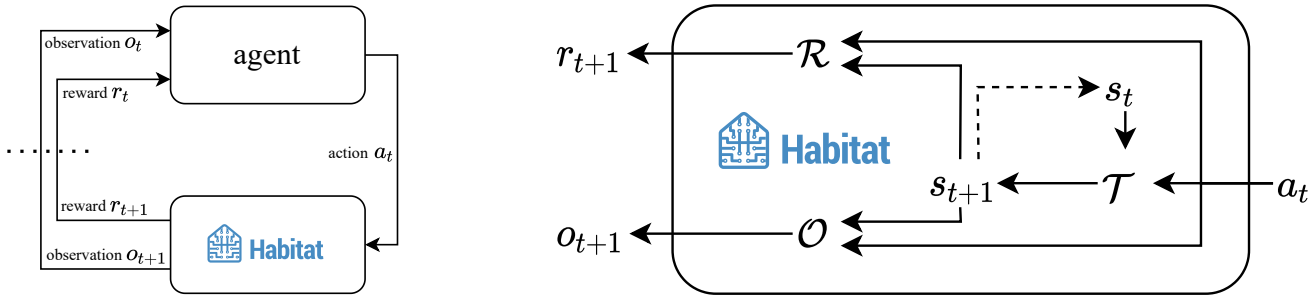
The agent, in the following referred to as policy $\pi_\theta(\cdot|\boldsymbol{o}_t, \boldsymbol{g}_t)$ parameterized by θ, computes a distribution over the action space $\mathcal{A}$, taking as input an observation from the environment $\boldsymbol{o}_t$ at time step t, and the current relative point goal position $\boldsymbol{g}_t$. In the environment, the agent makes an observation $\boldsymbol{o}_t$, takes an action a_t, transitions to a new state $s_{t+1} \sim \mathcal{T}(s_t, a_t)$, and receives a reward r_t. For a sequence $\tau = \{\langle \boldsymbol{o}_t, a_t, r_t \rangle\}_{t=0}^{T}$, the optimal policy π^* is then defined as

$$\begin{aligned} \pi^* &= \arg\max_\pi \mathbb{E}_{\tau \sim \pi} \left[\mathcal{R}_T \right], \\ \text{where} \quad \mathcal{R}_T &= \sum_{t=0}^{T-1} \gamma^t r_t, \end{aligned} \tag{3.1}$$

and T is the length of an *episode*. $\mathcal{R}_T$ is the cumulative sum over the discounted reward of an episode. Discount factor γ reduces future reward and is defined as $\gamma \in [0, 1)$ such as the sum in $\mathcal{R}_T$ converges when $T \to \infty$, which is not the case for $\gamma = 1$. On the other extreme $\gamma = 0$ only regards the reward at the current time step. Therefore, the factor is often set between $\gamma = 0.99$ or $\gamma = 0.999$, promoting a focus on reward closer in time while maintaining relatively high incentives for the future.

3.1.2. Policy Search

To find π^*, π_θ is randomly initialized, and several episodes are rolled out over it, *i.e.*, let the agent act in the environment, to obtain a dataset of $\langle \boldsymbol{o}_t, a_t, r_t \rangle$. At the start of training, the policy will act almost arbitrarily



(a) Agent environment interaction

(b) Habitat's role in the POMDP

Figure 3.1.: Interaction between the agent and its environment in the light of the defined POMDP.

and is updated to take better actions in the future. One update possibility is to compute the gradient of the reward *w.r.t.* the policy parameters θ and use gradient ascent to maximize $\mathbb{E}[\mathcal{R}_T]$ [118]. Methods following this paradigm are termed *policy gradient* methods [118]. Intuitively, updating the policy to maximize the reward increases the likelihood of better actions and avoids undesired behavior.

However, keeping some stochasticity in the action selection is desired as the policy could otherwise get stuck in local minima. Consider the example of navigating to a goal position in a maze. The agent finds itself at a crossroads and must choose between going left or right. While going right would immediately bring it closer to the goal, it runs into a dead-end eventually. Going left would move it further away for now but brings it directly to it after a short detour. Since going right poses an immediate reward signal, traversing the dead-end comes with a momentarily decrease in reward and is not incentivized. Taking a random action once in a while helps the agent discover potentially better paths to the goal, even though not intended by the policy. This process is called *exploration*, while the process of maximizing the reward at any given time is called *exploitation*. Finding a good policy requires to balance exploration and exploitation in the famous *exploration-exploitation tradeoff* [119].

Another pitfall is updating the policy too much [120]. Policy gradient methods are *on-policy* methods, *i.e.*, they use samples obtained from the current policy for the update [119]. In contrast, *off-policy* describes finding a policy from offline data such that the policy cannot explore the environment but is limited to exploiting given a fixed set of observations. Being on-policy means that updates in the policy immediately reflect on the samples collected for the next update. If the update moves the policy too far away from the previous one, it might cause catastrophic failures when the new policy returns low-quality samples. As recovering from this situation can be impossible, a trust-region constraint can force the update to stay close to the previous policy and ensure the new policy will collect informative samples [120, 121]. This constraint often takes the form of a KL term added to the optimization in Equation (3.1).

Proximal Policy Optimization (PPO) [122] is a well-established policy gradient method that strikes a balance between performance and ease of implementation. Decentralized Distributed Proximal Policy Optimization (DD-PPO) [9] modifies the PPO algorithm to run on multiple machines without a centralized server while keeping it easy to implement. This large-scale training with PPO shows almost perfect results in an idealized setting of PointGoal navigation. A more detailed description of these algorithms is disregarded since it is not subject to the presented work but used for the VO model evaluation.

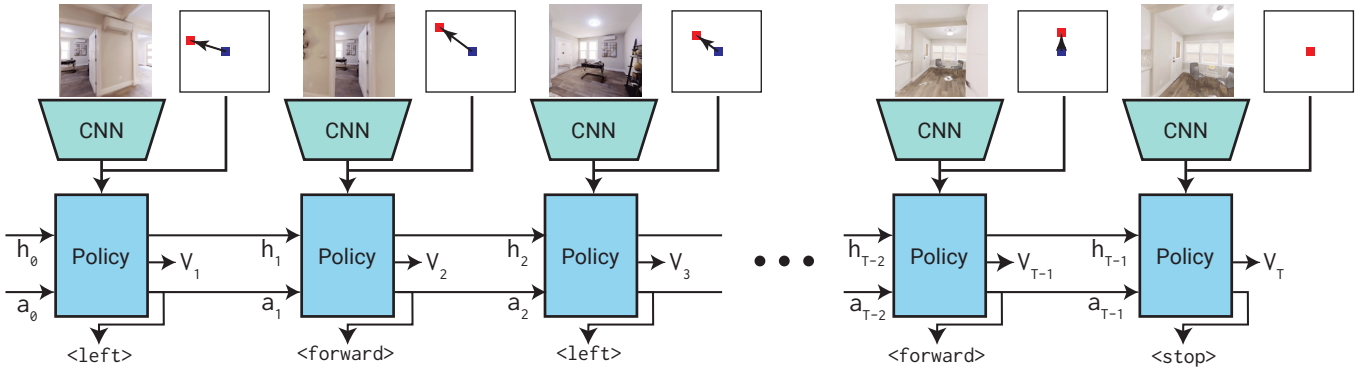


Figure 3.2.: Composition of an agent consisting of a ConvNet (here: CNN) to process the visual observations and a recurrent policy taking as input visual features, relative goal position, previous hidden state h_{t-1}, and previous action a_{t-1} to compute the next action a_t. Figure from [9].

3.1.3. Agent and Simulator

Figure 3.1 visualizes the interaction of an agent and the role of Habitat in the defined POMDP. First, Habitat simulates the environment dynamics, *i.e.*, state transitions $\mathcal{T}$. Therefore, it takes the agent's action and computes its new position and orientation based on the ground truth position, orientation, and (optional) actuation noise model. It then acts as an observation kernel $\mathcal{O}$ in that it returns observations corresponding to the new state, loaded 3D scene, equipped sensors, and (optional) observation noise models. Finally, it implements the reward function $\mathcal{R}$ and returns a reward regarding the true position and orientation of the agent.

Defining a reward function that leads to the desired behavior is a nontrivial task. A known issue is the *credit assignment problem* which describes the difficulty of assigning a reward to the actions responsible for achieving it [123]. Consider the example of PointGoal navigation, where the agent's objective is to call the `stop` action on reaching the goal. Succeeding within a given episode length counts as a success $S = 1$, and $S = 0$ otherwise. A reward given on task completion is called a *terminal reward*. Given the following reward function

$$r_t(a_t, s_t) = \begin{cases} 1 \cdot S & \text{if} \quad a_t = \texttt{stop} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

an agent would not receive a reward if it calls `stop` prematurely or finishes close to the goal due to, *e.g.*, actuation noise. Even though most of its actions brought it closer to the goal, the episode still counts as a failure $S = 0$. Another example is an agent reaching the goal but taking a detour before finally calling the `stop` action. Since the episode is considered a success $S = 1$, the actions leading to the detour would also be reinforced even though they are not the desired behavior.

These issues go hand-in-hand with *sparse reward*, *i.e.*, the agent only receives a reward once in a while [124]. This sparsity is especially problematic when collecting samples with a randomly initialized agent. Due to the considerable number of states in an environment, *i.e.*, search space of possible action sequences, it is unlikely that the agent will ever get a positive signal in the first place. Without a reward, the agent cannot improve, and training plateaus. *Reward shaping* [124] alleviates those downfalls by constructing a *denser* reward signal that provides more frequent intermediate feedback. For instance, one solution is to evaluate

each state-action pair separately and compute its corresponding reward. For the task of PointGoal navigation in Habitat, Wijmans *et al.* [9], therefore, formulate the following reward function

$$r_t(a_t, s_t) = \begin{cases} 2.5 \cdot S & \text{if} \quad a_t = \texttt{stop} \\ -\Delta_{geo\_dist}(a_t, s_t) - 0.01 & \text{otherwise} \end{cases}, \tag{3.3}$$

where $\Delta_{geo\_dist}(a_t, s_t)$ is the change in *geodesic distance* when performing action a_t in state s_t. The shortest path between two nodes in a graph is the geodesic path, and its length is the shortest or geodesic distance. In contrast to, *e.g.*, the Euclidean distance, the geodesic distance presents a more appropriate distance measure in an indoor navigation setup as it takes the structure of the environment into account [4]. For instance, an agent that is close to the goal but whose path is blocked by a wall experiences a low Euclidean distance which hinders learning a good navigation policy. In contrast, the geodesic distance considers the shortest path and facilitates staying close to it. If the shortest path is a straight line, both distances are equal. Penalizing the agent by its geodesic distance reinforces behavior that moves the agent closer to the goal. Additionally, a negative reward of 0.01 penalizes each action and incentivizes the agent to reach its goal more efficiently, *i.e.*, in fewer steps.

Following Savva *et al.* [1], an agent can learn using the reward signal in Equation (3.3) and utilize PPO to update its policy. Such an agent consists of an *encoder* and a *policy*. The encoder takes as input the observations and compresses them into a lower-dimensional representation. Since observations often take the form of `RGB` images or `Depth` maps, ConvNets, specifically, ResNets are the predominant encoder architecture. The representation and a goal location serve as input for the policy, which computes the action a_t the agent takes in the environment. Such policies often use an LSTM to deal with the long-term dependencies required for planning. This design choice implies that the policy curates a hidden state h_t that keeps track of past behavior and facilitates implicit localization, mapping, and planning. Figure 3.2 visualizes such a rollout over T time steps.

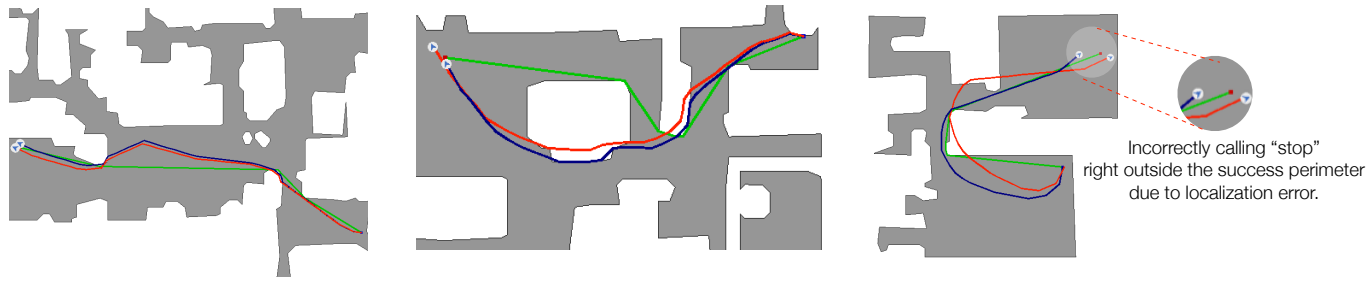
3.1.4. Evaluating PointGoal Navigation Performance

To evaluate agents in a PointGoal or ObjectGoal navigation setting, Anderson *et al.* [4] propose the Success weighted by (normalized inverse) Path Length (SPL). A crucial component of this metric is the success of an episode, denoted as a binary value $S = 1$ for successful completion, and $S = 0$ for a failure. With l the shortest path distance from the starting position and p the length of the path taken by the agent, the SPL over N episodes is defined as:

$$SPL = \frac{1}{N} \sum_{i=0}^{N-1} S^{(i)} \frac{l^{(i)}}{\max(p^{(i)}, l^{(i)})} \tag{3.4}$$

While SPL depends on the success of an episode, Datta *et al.* [2] propose the Soft Success Path Length (SSPL) that provides a more holistic view of the agent's navigation performance. The authors replace the binary success S of an episode with a soft value consisting of the ratio between the (geodesic) distances to target upon start d_{init} and termination of an episode d_g. The resulting metric is then:

$$SoftSPL = \frac{1}{N} \sum_{i=0}^{N-1} (1 - \frac{d_g^{(i)}}{d_{init}^{(i)}}) \cdot \frac{l^{(i)}}{\max(p^{(i)}, l^{(i)})} \tag{3.5}$$

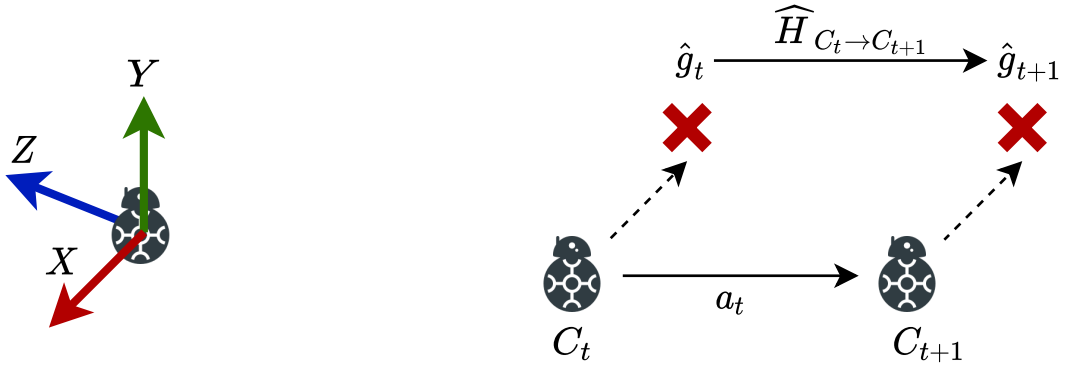


SPL=0.96, SoftSPL=0.95, geo_d_T=0.12 (a)

SPL=0.95, SoftSPL=0.93, geo_d_T=0.12 (b)

SPL=0.0, SoftSPL=0.89, geo_d_T=0.26 (c)

Figure 3.3.: Comparison of SPL and SSPL for different agent trajectories. Shortest path is denoted in green. In a) and b), the predicted pose (red) stays close to the ground truth pose (blue), hence SPL and SSPL are also close to each other. However, as the pose estimation in c) differs stronger from the ground truth, the agent calls `stop` prematurely, leading to a low SPL. Because it is still close to the goal, SSPL stays high. Geodesic distance to goal on termination of an episode d_g is denoted here as geo_d_T. Figure from [2].



(a) Agent coordinate system C_t

(b) Transformation between the agent's coordinate systems $C_t \to C_{t+1}$ and corresponding transformation of the goal position estimates $\widehat{g}_t \to \widehat{g}_{t+1}$

Figure 3.4.: Agent and goal positions *w.r.t.* the coordinate system transformation. Figures inspired by [2].

If the geodesic distance is chosen, $d_{init} = l$. The closer the agent gets to the goal, the higher the SSPL, even if the episode is unsuccessful. This softening allows distinguishing agents that fail to complete a single or multiple episodes but move significantly close to the goal from ones that move away from it. Without access to `GPS+Compass`, SSPL becomes significantly more important as an agent could call `stop` prematurely due to noisy localization. Consider Figure 3.3 for specific examples and failure cases. For agents that diverge from their starting position away from the goal, the SSPL can also become negative as $d_g > d_{init}$. However, Habitat forces the soft success to be the $\max(0, (1 - \frac{d_g}{d_{init}}))$.

3.1.5. Visual Odometry Formulation

In a realistic PointGoal navigation setting, the agent receives a goal position $\boldsymbol{g}_t$ at the beginning of an episode. This goal is defined relative to its starting position and is updated throughout the episode as the agent changes its position and orientation with each action. With access to `GPS+Compass`, the agent can directly compute

the transformation of its coordinates caused by the action taken and apply it to the goal to update it *w.r.t.* its new location. When GPS+Compass is unavailable, the transformation has to be estimated solely from the agent's observations to obtain an estimate of the relative goal position $\widehat{\boldsymbol{g}}_t$. This section largely follows the work of Datta *et al.* [2] and Zhao *et al.* [3].

The agent's coordinate system at time step t is defined as C_t and illustrated in Figure 3.4a. When the agent takes an action a_t, its coordinate system transforms into C_{t+1}. Such transformation from $C_t \to C_{t+1}$ is denoted as $H_{C_t \to C_{t+1}} \in SE(2)$ with $SE(2)$ the group of rigid transformations in a 2D plane. Because the agent can only navigate planar, and scenes only consist of one level, the 3rd dimension gets discarded. The resulting transformation is defined as

$$\begin{aligned} H_{C_t \to C_{t+1}} &= \begin{bmatrix} R_{C_t \to C_{t+1}} & \boldsymbol{\xi}_{C_t \to C_{t+1}} \\ 0 & 1 \end{bmatrix} \\ \text{with} \quad R_{C_t \to C_{t+1}} &= \begin{bmatrix} \cos(\beta_{C_t \to C_{t+1}}) & -\sin(\beta_{C_t \to C_{t+1}}) \\ \sin(\beta_{C_t \to C_{t+1}}) & \cos(\beta_{C_t \to C_{t+1}}) \end{bmatrix} \in SO(2) \end{aligned} \tag{3.6}$$

where $\beta_{C_t \to C_{t+1}}$ is an angle, $R_{C_t \to C_{t+1}}$ is a rotation matrix from the special orthogonal group, and $\boldsymbol{\xi}_{C_t \to C_{t+1}} \in \mathbb{R}^2$ is a translation vector. Without GPS+Compass, the true transformation $H_{C_t \to C_{t+1}}$ cannot be computed but instead it can be tried to find an estimate $\widehat{H}_{C_t \to C_{t+1}}$ of it. Zhao *et al.* [3] propose to regress the transformation parameters $\beta_{C_t \to C_{t+1}}$ and $\boldsymbol{\xi}_{C_t \to C_{t+1}}$ using a VO model f_ϕ resulting in

$$\begin{aligned} \widehat{H}_{C_t \to C_{t+1}} &= \begin{bmatrix} \widehat{R}_{C_t \to C_{t+1}} & \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \\ 0 & 1 \end{bmatrix} \\ \text{with} \quad \widehat{R}_{C_t \to C_{t+1}} &= \begin{bmatrix} \cos(\widehat{\beta}_{C_t \to C_{t+1}}) & -\sin(\widehat{\beta}_{C_t \to C_{t+1}}) \\ \sin(\widehat{\beta}_{C_t \to C_{t+1}}) & \cos(\widehat{\beta}_{C_t \to C_{t+1}}) \end{bmatrix} \in SO(2) \\ \langle \widehat{\beta}_{C_t \to C_{t+1}}, \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \rangle &= f_\phi(\boldsymbol{o}_t, \boldsymbol{o}_{t+1}) \end{aligned} \tag{3.7}$$

where f_ϕ is parameterized by ϕ and taking egocentric observations $\boldsymbol{o}_t$ and $\boldsymbol{o}_{t+1}$ as input.

Estimating $H_{C_t \to C_{t+1}}$ would be sufficient for the task of VO. Dealing with PointGoal navigation, the goal has to be updated since the agent position itself is not sufficient to solve the task. Therefore, the relative goal $\widehat{\boldsymbol{g}}_t$ is transformed to the new agent coordinate system C_{t+1} resulting in the new goal coordinates $\widehat{\boldsymbol{g}}_{t+1} = \widehat{H}_{C_t \to C_{t+1}} \cdot \widehat{\boldsymbol{g}}_t$ as visualized in Figure 3.4b.

3.2. The Visual Odometry Transformer

3.2.1. Transformers for Visual Odometry

Most methods so far estimate the VO using ConvNet architectures [2, 3, 58]. Recently, Zhu *et al.* [55] proposed to integrate a Convolutional Block Attention Module (CBAM) into their ConvNet to allow the model to focus on pixels with distinct motion. CBAM has shown improvements in VO estimation and makes model put less weight on foreground, noisy, or blurred features. In RGB- and Depth-based visual navigation, another challenge is the distribution of information across the receptive field of the images. Consider the samples in Figure 2.3. While in the RGB images, the walls on both sides appear to be textureless and, therefore, provide less information about the agent's movement, the Depth map shows a smooth gradient of values. Vice versa, for faraway objects and surfaces, *e.g.*, windows, Depth does not provide accurate measurements, while RGB

contains visual textures and unique features, *e.g.*, window frames. With the widespread adaptation of ViTs in CV, the question arises whether a new architecture based solely on the attention mechanism can outperform recent ConvNet-based VO approaches.

It is believed that biological eyes use global structure to determine angular and translation changes [125]. Especially when the magnitude of translation and rotation between frames is large, the locality bias of ConvNets becomes disadvantageous. With its receptive field limited to a neighborhood grid of the convolutional kernel size, the ViT could better exploit such features through its receptive field spanning the full image [93].

Another disadvantage of ConvNet architectures is the lack of recurrent information processing. Recent multi-frame approaches use ConvNets as backbones and deploy an RNN to process the information [59, 60]. Transformers can be easily extended to multiple frames due to the flexible number of input tokens. Furthermore, they could make use of recurrence in the two consecutive images.

In most real-world cases, VO does not solely have to rely on visual observations because additional information is available through an Inertial Measurement Unit (IMU), noisy GPS, or even the noise-free action taken by the agent. Such circumstances spark the need for an easy integration of external information into the VO model. For ConvNet-based architectures, Conditional Batch Normalization (CBN) [126] is a widespread method that conditions the network by predicting the normalization parameters of Batch Normalization (BN) [127] from it. Other approaches turn to a simple concatenation of the information embeddings to the final or latent feature representations [3]. The reliance of the Transformer on token inputs allows for a more straightforward approach to integrating additional information and modalities. By simply embedding them into token-sized representations, the Transformer can use them without changing the architecture itself. This property is specifically beneficial in pre-trained models where integrating CBN is nontrivial.

Finally, the visualizations of the attention maps demonstrated by Caron *et al.* [69] offer an out-of-the-box capability to interpret which regions the ViT attends to and what matters in VO. Such interpretation enables a confirmation or a rejection of the hypotheses on information distribution in `RGB` and `Depth`, and the global nature of the VO problem.

3.2.2. Architecture

This work proposes the Visual Odometry Transformer (VOT), an architecture to regress VO parameters based on a ViT backbone. With the benefits of a ViT over ConvNets caused by the attention mechanism, the hypothesis arises that it can exploit global structure [12], leverage multi-modal inputs [114], and deal with noisy or blurry inputs [55]. Figure 3.5 shows the different components of the architecture. At the core of the VOT stands a ViT backbone which adopts different pre-training methods. As shown in PoseNet [15] and DeepVO [59], using a pre-trained backbone like GoogLeNet [53] and FlowNet [61] enormously benefits the task of VO. Furthermore, the lack of inductive bias, and limited access to extensive training data suggest that pre-training is indispensable for training a ViT backbone.

Similar to an image classification task, the $[CLS]$ token is used for the VO parameter estimation. Therefore, the token is fed into an two-layer MLP with parameters ψ composed into $W_0 \in \mathbb{R}^{dim \times dim_h}, \boldsymbol{b}_0 \in \mathbb{R}^{dim_h}$, and $W_1 \in \mathbb{R}^{dim_h \times 3}, \boldsymbol{b}_1 \in \mathbb{R}^3$ with token dimensions $dim = 384$ (ViT-S) or $dim = 768$ (ViT-B), and hidden dimensions $dim_h = dim/2$. A Gaussian Error Linear Unit (GELU) [128] is used as an activation function between the two layers as it has shown to outperform the Rectified Linear Unit (ReLU) [129] in most CV tasks. Revisiting Equation (3.7), the VO model is defined as a function $f_{\phi,\psi}(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle)$ taking as input observations $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$ which correspond to either `RGB`, `Depth`, or `RGB-D` and predicting the VO parameters

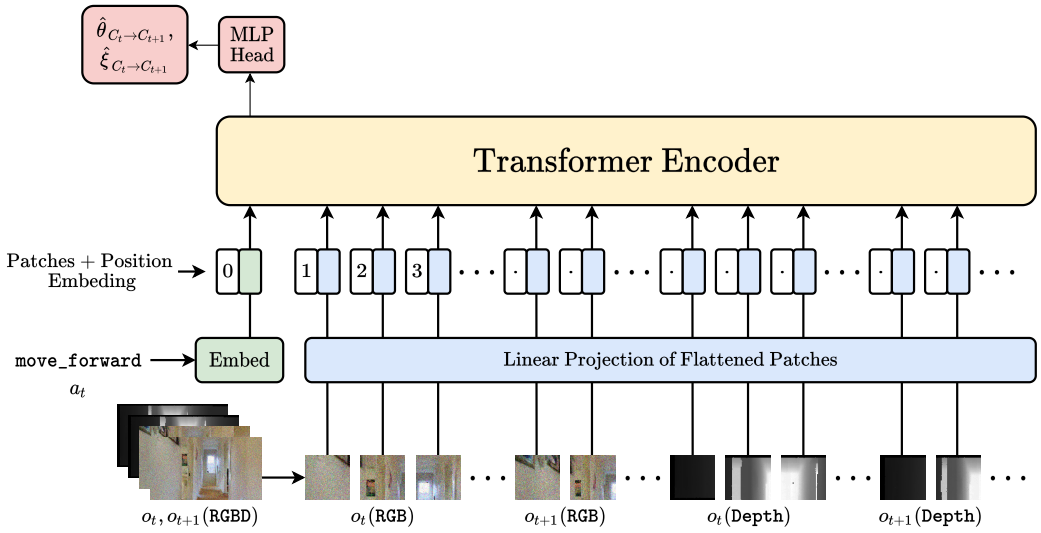


Figure 3.5.: Vanilla Visual Odometry Transformer. Visualization inspired by Dosovitskiy *et al.* [13].

$\langle \widehat{\beta}_{C_t \to C_{t+1}}, \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \rangle$. Simplifying the ViT backbone as $b_\phi(\boldsymbol{o}_t, \boldsymbol{o}_{t+1})$ that returns the encoded $[CLS]$ token as visual features $\boldsymbol{v}_{t \to t+1} \in \mathbb{R}^{1 \times dim}$, and governed by parameters ϕ, the following equations are obtained:

$$\begin{aligned} \boldsymbol{v}_{t \to t+1} &= b_\phi(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle) \\ \text{MLP}_\psi(\boldsymbol{v}) &= \text{GELU}(\boldsymbol{v}W_0 + \boldsymbol{b}_0)W_1 + \boldsymbol{b}_1 \\ f_{\phi,\psi}(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle) &= \text{MLP}_\psi(b_\phi(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle)) \\ &= \langle \widehat{\beta}_{C_t \to C_{t+1}}, \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \rangle \end{aligned} \tag{3.8}$$

Additionally, the model has access to the action a_t taken by the agent to get from s_t to s_{t+1} and resulting in the observations $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$. Since in the realistic PointGoal task, these observations are subject to noise from [43], the action is assumed to be a strong prior on the VO parameter estimation. To provide this information to the model, the action represented as an integer (`fwd` =1,`left` =2,`right` =3) is embedded using an embedding layer [130]. This layer acts as a learnable lookup for each action, mapping it to a fixed-size embedding. With the embedding size equal the token dimensions, the embedding replaces the $[CLS]$ token, resulting in the Action Token ($[ACT]$). This conditioning facilitates directed information accumulation without alternating the model architecture and hurting pre-training possibilities. Learning the embedding even if the pre-trained weights of the ViT are frozen allows for the best possible information extraction from the backbone. Extending Equation (3.8) by the action results in:

$$\begin{aligned} \boldsymbol{v}_{t \to t+1} &= b_\phi(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle, a_t) \\ \text{MLP}_\psi(\boldsymbol{v}) &= \text{GELU}(\boldsymbol{v}W_0 + \boldsymbol{b}_0)W_1 + \boldsymbol{b}_1 \\ f_{\phi,\psi}(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle, a_t) &= \text{MLP}_\psi(b_\phi(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle, a_t) \\ &= \langle \widehat{\beta}_{C_t \to C_{t+1}}, \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \rangle \end{aligned} \tag{3.9}$$

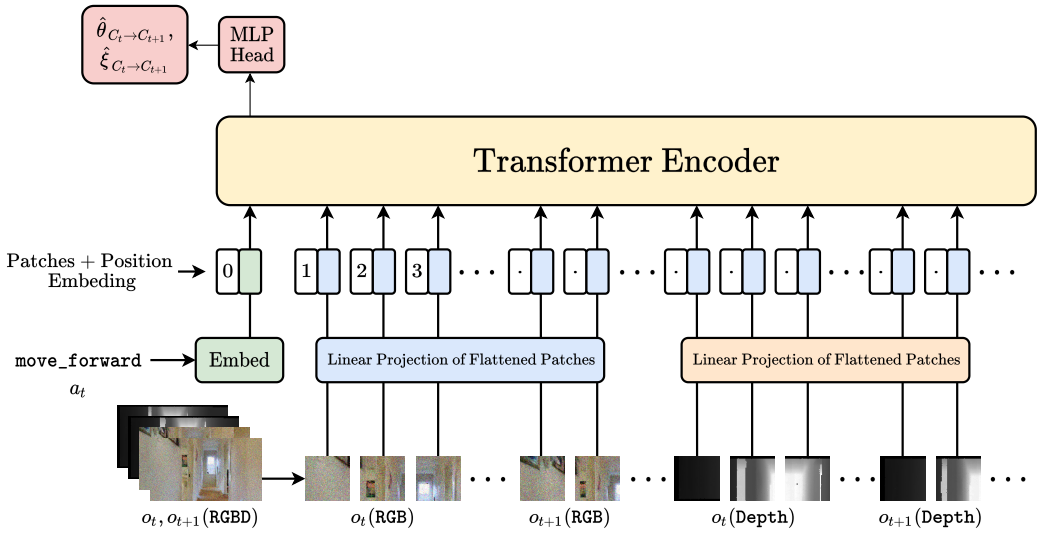


Figure 3.6.: Multi-modal VOT with a MultiMAE backbine and separate projection layers for RGB and Depth. Visualization inspired by Dosovitskiy *et al.* [13].

The independence of the backbone on the number of input tokens allows for the derivation of various uni-modal and multi-modal designs of the VOT. While Figure 3.5 visualizes the multi-modal case, Figure E4 shows a VOT with RGB-only input.

As proposed in the original ViT paper [13], the VOT uses a learnable positional embedding and following linear projections to embed the patches into the corresponding latent dimensionality. When using the pre-trained MultiMAE as a backbone, however, the linear projection of the input tokens is replaced by separate projection layers for RGB and Depth [114]. Furthermore, a fixed positional embedding replaces its learnable counterpart, as the network is assumed to distinguish the modalities with the adapted projection layers [114]. While these changes hurt comparability of the different approaches, they are necessary to avoid re-training the MultiMAE and the other initializations from scratch. Figure 3.6 shows the alternative approach with a MultiMAE backbone.

3.2.3. Training Loss

The regression loss suggested by Zhao *et al.* [3] is adopted to train the model. It is based on the $L2$-norm between the ground truth transformation parameters $\langle \beta_{C_t \to C_{t+1}}, \boldsymbol{\xi}_{C_t \to C_{t+1}} \rangle$ and their estimated counterparts $\langle \widehat{\beta}_{C_t \to C_{t+1}}, \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \rangle$ as derived in Equation (3.7). The loss is then defined as follows:

$$\mathcal{L}^{reg}_{C_t \to C_{t+1}} = \| \boldsymbol{\xi}_{C_t \to C_{t+1}} - \widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} \|_2^2 + \| \beta_{C_t \to C_{t+1}} - \widehat{\beta}_{C_t \to C_{t+1}} \|_2^2 \tag{3.10}$$

Since the regression loss in Equation (3.10) is insufficient to estimate the VO parameters in a noisy setting, Zhao *et al.* [3] propose additional *geometric invariance losses*. They build on the intuition that an agent that subsequently receives the observation tuple $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$ and $\langle \boldsymbol{o}_{t+1}, \boldsymbol{o}_t \rangle$ would geometrically end up where it started, *i.e.*, $\widehat{H}_{C_t \to C_{t+1}} \cdot \widehat{H}_{C_{t+1} \to C_t} = I_{3 \times 3}$

A rotation by an angle $\beta_{C_t \to C_{t+1}}$ causing a coordinate transformation from $C_t \to C_{t+1}$, *e.g.*, can be undone by its inverse, *i.e.*, $-\beta_{C_t \to C_{t+1}}$ transforms $C_{t+1} \to C_t$. This means that $\beta_{C_{t+1} \to C_t} = -\beta_{C_t \to C_{t+1}}$ and $\widehat{\beta}_{C_t \to C_{t+1}} + \widehat{\beta}_{C_{t+1} \to C_t} = 0$ should be encouraged for the model predictions to reflect this relationship. The resulting rotation invariance loss is self-supervised since it only uses model predictions and no external information. Similar to Equation (3.10), it is based on the $L2$-norm:

$$\mathcal{L}^{inv,rot}_{C_t \to C_{t+1}} = \|\widehat{\beta}_{C_t \to C_{t+1}} + \widehat{\beta}_{C_{t+1} \to C_t}\|^2_2 \tag{3.11}$$

The translation invariance loss builds on the same intuition, *i.e.*, if the transformation from $C_t \to C_{t+1}$ is a pure translation $\boldsymbol{\xi}_{C_t \to C_{t+1}}$, then the transformation can be reversed by $\boldsymbol{\xi}_{C_{t+1} \to C_t} = -\boldsymbol{\xi}_{C_t \to C_{t+1}}$. The loss then is defined as:

$$\mathcal{L}^{inv,trans,only}_{C_t \to C_{t+1}} = \|\widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} + \widehat{\boldsymbol{\xi}}_{C_{t+1} \to C_t}\|^2_2 \tag{3.12}$$

In case the transformation is subject to both rotation and translation, the loss as of [3] becomes:

$$\mathcal{L}^{inv,trans}_{C_t \to C_{t+1}} = \|\widehat{\boldsymbol{\xi}}_{C_t \to C_{t+1}} + \widehat{R}_{C_t \to C_{t+1}} \cdot \widehat{\boldsymbol{\xi}}_{C_{t+1} \to C_t}\|^2_2 \tag{3.13}$$

The training procedure of the VOT uses a combination of Equation (3.10), Equation (3.11), and Equation (3.13):

$$\mathcal{L}_{C_t \to C_{t+1}} = \mathcal{L}^{reg}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,rot}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,trans}_{C_t \to C_{t+1}} \tag{3.14}$$

Assuming a dataset $\mathcal{D}$ consists of datapoints $d_{C_t \to C_{t+1}} = (\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle, \langle \boldsymbol{\xi}_{C_t \to C_{t+1}}, \beta_{C_t \to C_{t+1}} \rangle)$ with $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$ an observation tuple, and $\langle \boldsymbol{\xi}_{C_t \to C_{t+1}}, \beta_{C_t \to C_{t+1}} \rangle$ the ground truth translation and rotation parameters, the training objective is defined as:

$$\min_{\phi,\psi} \mathcal{L}_{C_t \to C_{t+1}} = \sum_{d_{C_t \to C_{t+1}} \in \mathcal{D}} \left[\mathcal{L}^{reg}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,rot}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,trans}_{C_t \to C_{t+1}} \right] \tag{3.15}$$

Note that the hyperparameters scaling the different components of the loss, as suggested by Zhao *et al.* [3] are all set to 1.0. Finally, an optimizer of choice updates the parameters ϕ, ψ to minimize the objective.

4. Experimental Evaluation

4.1. Dataset

4.1.1. AI Habitat Setup

The dataset is collected in Habitat following the *Habitat PointNav Challenge 2020* [38]. Because the baseline policy was trained using the 2020 version, this will be the configuration used throughout this work to facilitate comparability to prior work and to avoid robustness issues that could require re-training or fine-tuning. The guidelines include an action space of `fwd` (move forward $0.25m$), `left` (turn left by $30°$, `right` (turn right by $30°$), and `stop` (indicate the agent reached its goal), and a sensor suite of `RGB-D` camera, and `GPS+Compass` (not used in the PointGoal task). The `RGB` observations get returned in a $[0, 255]$ range while the `Depth` map is scaled to $[0, 10]$. Both sensors experience noisy actuation and observations with noise models from [10] and [43], respectively. Furthermore, collision dynamics are updated to prevent *sliding*, a behavior that allows the agent to slide along walls on collision. The agent can exploit this mechanic, which does not reflect reality. Finally, some cosmetic changes bring the simulation closer to *LoCoBot* [131], the low-cost robotic platform. For instance, agent radius and height become $0.18m$ and $0.88m$, and optical sensor resolution is 341×192 (width $\times$ height), emulating an Azure Kinect camera. Figure 4.1 shows two LoCoBot platforms. An episode is successful if the agent calls `stop` in a radius two times its own, *i.e.*, $0.36m$ around the point goal.

By specification, the 3D scenes loaded into Habitat are from the Gibson [6] dataset, more precisely, a subset of 72 scenes with a rating of $4+$ [1]. The validation set contains 14 scenes, which are not part of the training set. Table A1 lists the scene names for both training and validation split.

4.1.2. Data Collection Procedure

To train the VO model, a training and a validation dataset $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ are collected that consists of datapoints $d_{C_t \to C_{t+1}} = (\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle, a_t), \langle \boldsymbol{\xi}_{C_t \to C_{t+1}}, \beta_{C_t \to C_{t+1}} \rangle)$. In contrast to previous work, a datapoint contains the observation tuple $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$ and action a_t, which will act as a prior over the VO parameter estimation. $\langle \boldsymbol{\xi}_{C_t \to C_{t+1}}, \beta_{C_t \to C_{t+1}} \rangle$ are the ground truth translation and rotation parameters retrieved from a perfect `GPS+Compass` sensor.

The collection procedure follows [3] and is described as follows: 1) Initialize the Habitat simulator and load a scene from the Gibson dataset 2) Place the agent at a random location within the environment with a random orientation 3) Sample a navigable PointGoal the agent should navigate to 4) Compute the shortest path and let the agent follow it 5) Randomly sample data points $d_{C_t \to C_{t+1}}$ along the trajectory. This work collects a dataset of only 250k samples, which is significantly less than the previous methods performing well in the most recent *Habitat PointNav Challenge 2021* (1M [3], 2M [68]). The 2021 challenge also follows the 2020 configuration.

Figure 4.1.: LoCoBot (left) and LoCoBot-Light (right). Both are low-cost robotic platforms equipped with an RGB-D camera, a gripper mounted to a 5-DoF arm that can be used for manipulation, and a mobile base for locomotion. Image from [10].

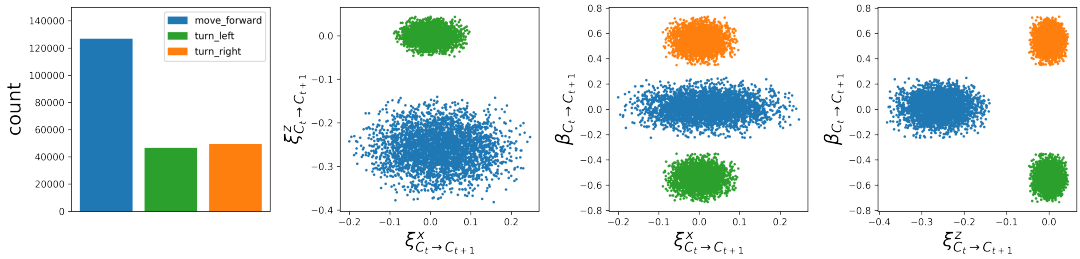
Table 4.1.: Sample statistics of the training and validation sets $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ with data augmentation of `left` and `right`. Augmented datasets are denoted as $\widetilde{\mathcal{D}}$.

dataset	`fwd`	`left`	`right`	total
$\mathcal{D}_{train}$	143697	53140	53163	250000
$\widetilde{\mathcal{D}}_{train}$	143697	106303	106303	356303
$\mathcal{D}_{val}$	13990	5542	5468	25000
$\widetilde{\mathcal{D}}_{val}$	13990	11010	11010	36010

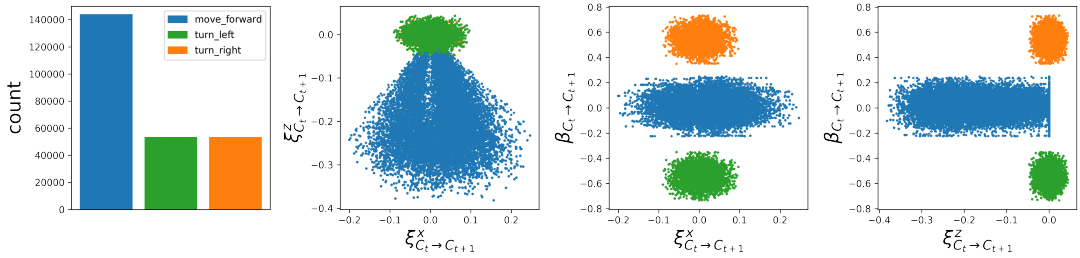
4.1.3. Data Exploration

Since this manuscript proposes a single model to regress the VO parameters for the full action space, the datasets also contain samples from all actions. Table 4.1 lists the distribution of action samples contained in $\mathcal{D}_{train}$. In total 4891 rollouts are collected, and in $\sim 11.3\%$ of the samples, the agent collides with the environment boundaries. The high occurrence rate stresses the importance of including collisions in the training data. Observe the difference in translation and rotation distributions between the data with collision (*cf.* Figure 4.2b) and without collisions (*cf.* Figure 4.2a). The collisions manifest itself in the visualizations as bounds on the parameters of a specific action, *e.g.*, $\boldsymbol{\xi}^{Z}_{C_t \to C_{t+1}}$ of `fwd` at 0.0. The collision's effect on the `fwd` action is the greatest as running into a wall causes the agent to stop immediately. The influence of `left` and `right` is less distinct as the agent does not move but rotates on the spot.

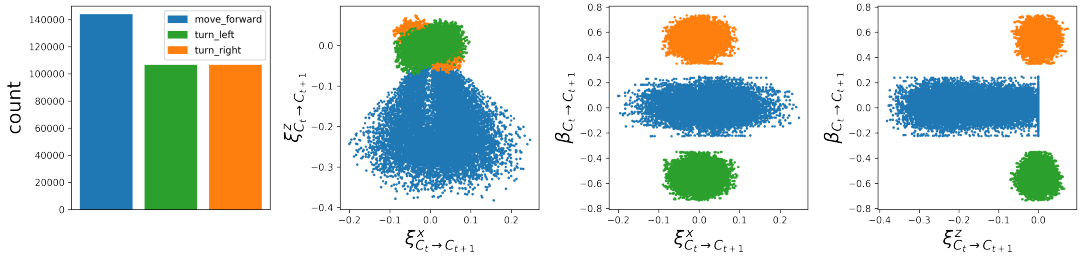
Zhao *et al.* [3] propose to train separate models for each action due to the differences in noise distributions. For their approach, separate datasets that contain `fwd`, and `left`, `right`, respectively, will be collected to improve read/write access to the disk. Dataset statistics are kept close to the main dataset and are provided in Table A4. However, the assumption of separate models restricts the approach to small discrete action spaces, which hinders scalability to continuous or larger action spaces. Supporting this argument, Figure 4.3 visualizes the distribution of an action space extended by turning actions of 15° in the left and right direction. As can be seen, the lower turning angles move the distribution closer to the `fwd` action if collision is enabled. Furthermore, left and right turns experience a similar noise distribution but differ through a flipped sign of the parameters. Therefore, one could argue that learning all actions at once lets the model generalize better to new and even make accurate predictions for continuous action spaces. Therefore, this work will focus on



(a) Training and validation sets $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ without collisions.



(b) Training and validation sets $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ with collisions.



(c) Augmented training and validation sets $\widetilde{\mathcal{D}}_{train}$ and $\widetilde{\mathcal{D}}_{val}$ with collisions.

Figure 4.2.: Dataset statistics of the training and validation sets (*cf.* Table 4.1).

training a single VO model to capture as much overlapping information as possible and maintain the possibility of learning continuous action spaces in the future.

Because `left` and `right` show the unique geometric relationships presented in Section 3.2.3, it stands to reason to use them for the data collection process. As common in recent work [3, 68], the presented approach also uses data augmentation based on this observation. During training, visual observations $\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle$ corresponding to actions `left` and `right` get flipped, and the corresponding ground truth VO parameters get computed (*cf.* Section 3.2.3). Actions `fwd` get omitted since the agent can not move backward. Table 4.1 displays the new sample distributions marked as $\widetilde{\mathcal{D}}$ while Figure 4.2c visualizes the new parameter distributions for $\widetilde{\mathcal{D}}_{train}$.

4.2. Training Setup

The following will present the training setup to facilitate reproducibility and comparability. In all cases, the ViT acts as a generic backbone to study the impact of various pre-training methods. The ablation study uses a ViT-S(mall) to keep the computational footprint in check. In contrast, the study of pre-training benefits

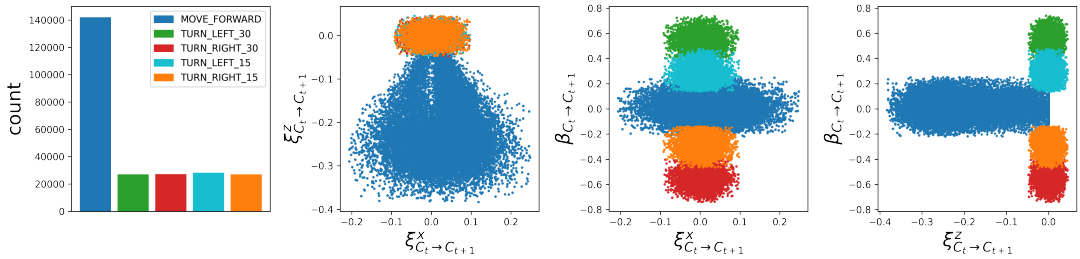


Figure 4.3.: Dataset statistics of a separate dataset with collisions and extended action space (*cf.* Table A2). This dataset is not used for training.

Table 4.2.: Backbone configuration using the timm [132] library. Number of encoder blocks N (blocks, *cf.* Figure 2.6a). Channel or token dimension of the output features (dim). Number of attention heads h in the MHA-layers (heads, *cf.* Figure 2.6b).

model	blocks	dim	heads	#params
ViT-S	12	384	6	22M
ViT-B	12	768	12	86M
ResNet-18	-	512	12	11M
ResNet-50	-	2048	12	25M
ResNet-101	-	2048	12	44M

turns to the ViT-B(ase), for which more pre-trained weights are available. For the sake of readability, the VOT adopts the terminology of the ViT with VOT-S, and VOT-B referring to the size of the backbone. This work opts for a patch size of 16×16 and ViTs pre-trained on 224×224 images balancing computational effort and availability of pre-trained weights.

Both ViT-S and ViT-B consist of 12 encoder blocks but vary in terms of attention heads in the MHA-layers with 6 and 12, respectively. The number of the attention heads, and an increase in the latent, *i.e.*, token, dimensions from 384 to 768, accounts for a parameter growth from 22 Million to 86 Million. Table 4.2 compares the ViT to the ResNet backbones. Note that while only the channel dimension of the ResNets are listed, they commonly return a 12×12 map for each channel due to the reliance on ConvNets. In contrast, the ViTs return one vector of the channel dimension for each input token. Common practice is to use the $[CLS]$ token for the downstream task resulting in a feature dimension equal to the channel dimension.

The model implementations use *PyTorch* [130], which provides an automatic differentiation engine and allows one to focus on the model architecture by abstracting away gradient computations and parameter updates. Additionally, PyTorch Image Models (timm) [132] provides a large number of CV models and corresponding pre-trained weights to support the implementation of the visual backbones. Specifically, this work uses the ResNet-50 (supervised), ViT-S (supervised, DINO), and ViT-B(supervised, DINO) implementations and corresponding pre-trained weights trained on IN-21k. Note that even though the weights are all integrated into timm, they might stem from other projects like the original repository, *e.g.*, from DINO [69]. Pre-trained MultiMAE weights and model configurations are taken from the official repository [133] of Bachmann *et al.* [114]. To track the experiments, *Weights & Biases* [134] is utilized.

The code bases itself on the pipeline presented by Zhao *et al.* [3], which integrates the VO models into Habitat agents and implements data collection, model training, and PointGoal evaluation. This thesis provides an

extension of the codebase by the proposed VOT models, corresponding training functionalities like gradient norm clipping [135] (set to 1.0 for all VOT models), pre-training, and attention map visualizations.

All models are trained on an NVIDIA A100-SXM4-40GB GPU with the exception of VOT-B with `RGB-D` input and no frozen weights, which is trained on two instances via distributed training implemented using PyTorch's distributed data-parallel. Furthermore, automatic mixed-precision is enabled in PyTorch to reduce memory footprint and increase training speed.

The VOT and unified ConvNet models are trained on $\widetilde{\mathcal{D}}_{train}$, and the separate ConvNet models for each action proposed by [3] are trained on the splits in Table A4. Both trainings consist of 150 epochs using the *Adam* [136] optimizer ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1.e^{-8}$) with 10 warm-up steps where the learning rate increases linearly from 0.0 to $2e.^{-4}$ marking the learning rate for the remaining 140 epochs. Table B5 provides an overview of the hyperparameters per model. In the VOT, observations are reshaped to 160×80 (width $\times$ height), `RGB` and `Depth` normalized separately, and the linear projection weights of the patch embedding are shared between both modalities. The `Depth` channel is repeated three times to match the number of `RGB` channels. The VOT-B with a MultiMAE backbone uses two linear projections which process `RGB` and `Depth` separately and do not require this channel repetition. Furthermore, the architecture uses a fixed positional embedding as the model can already distinguish `RGB` from `Depth` through the projection layers.

VO model performance is reported over the validation set $\widetilde{\mathcal{D}}_{val}$ or the splits in Table A4 for unified and separate models, respectively (*cf.* Table 4.1). In the following, those will jointly be referred to as $\widetilde{\mathcal{D}}_{val}$. The evaluation loss in Equation (3.14) is then defined over all actions as

$$\mathcal{L}_{val} = \sum_{d_{C_t \to C_{t+1}} \in \widetilde{\mathcal{D}}_{val}} \left[\mathcal{L}^{reg}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,rot}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,trans}_{C_t \to C_{t+1}} \right] \tag{4.1}$$

and minimized as proposed in Equation (3.15) using an optimizer. If appropriate, training loss $\mathcal{L}_{train}$ is computed accordingly and reported to assess possible overfitting. To reflect the difference in the action distribution, the regression loss on the validation datasets $\mathcal{D}_{val}$ is also reported separately for each action type (`fwd`, `left`, `right`) as $\mathcal{L}^{forward}_{val}, \mathcal{L}^{left}_{val}, \mathcal{L}^{right}_{val}$. Subscript $C_t \to C_{t+1}$ is omitted for readability purposes in all cases. Decimal places are truncated.

Furthermore, the models act as a replacement for the `GPS+Compass` sensor of a navigation policy trained by [137] with ground truth localization as proposed in [9]. The navigation policy consists of an LSTM with two recurrent layers that process 1) a 512-dimensional vector of egocentric observations, 2) a 32-dimensional embedding of the previous action, and 3) a 32-dimensional embedding of the goal position. The observation vector gets obtained by passing the visual observations through a ResNet-18 backbone, flattening the resulting feature map to dimensionality 2052, and projecting it to dimensionality 512 with a fully-connected layer. Finally, the output of the LSTM is fed through another fully-connected layer to produce a distribution over the action space and an estimate of the value function. The policy is trained using PPO. In contrast to [3], this work does not fine-tune the navigation policy on the trained VO models. As suggested by the Habitat challenges SPL, SSPL, success S, and (geodesic) distance to goal on termination d_g get reported and decimals are truncated.

4.3. Baselines

4.3.1. Trivial Baselines

An extensive evaluation of the presented approach calls for baselines reflecting upper and lower performance bounds. The lower performance bound is represented by a `Blind` VOT-S that does not have access to any information, meaning it must randomly guess the displacements caused by the agent's actions. Another simplistic baseline is a VOT-S that only uses the previous action to estimate the translation and rotation. When the actuation is not noisy, learning a deterministic mapping from actions to displacement allows the agent to localize itself, assuming it avoids collisions as those would introduce noise-like errors to the location. As soon as actuation becomes noisy, a mapping is no longer accurate enough because a specific action no longer leads to a fixed change in position and orientation. Without access to visual features, it is impossible to estimate the subtleties of the actuation noise. For instance, the displacement of corners, edges, and objects in the scene provides additional information that can benefit the VOT when making its prediction. Furthermore, it allows detecting collision, when `Depth` values become low or `RGB` registers mostly uniform color, an indication for running into a wall. However, the approach can still lead to success when the agent is placed in an overly easy setting. For instance, starting close to the goal reduces the accumulated action noise over an episode, and whenever the shortest path is a straight line, the collision probability decreases.

On the other extreme, the *oracle* acts as the upper performance bound. Through access to ground truth localization, the oracle agent can update its goal perfectly and use visual observations to plan its path around obstacles or corners. The goal is to get as close to the resulting performance as possible through a VO-based navigation approach. A small gap between them would indicate a minimal performance drop in using VO over ground truth `GPS+Compass` information.

Training models from scratch, *i.e.*, not initialized with pre-trained weights, sets a baseline for comparing unsupervised, multi-modal, and supervised pre-training strategies. Additionally, freezing the pre-trained weights of the backbone gives an even deeper insight into the usefulness of the learned features for the downstream task of VO.

4.3.2. State-Of-The-Art Baselines

The approach presented by Zhao *et al.* [3] represents the state-of-the-art approach and is used to place the proposed method in the context of current research. The authors propose to extend the VO formulation of Datta *et al.* [2] by improving its robustness against observation noise. They present several pre-processing methods mainly based on `Depth` input. For instance, they discretize the continuous sensor `Depth` map into N channels and compute a top-down projection of the scene that injects additional external knowledge about the VO problem. Furthermore, they introduce the geometric invariance losses described in Section 3.2.3. Finally, they make similar observations as in Section 4.1.3 and propose to train one model per action type. They also show results for a *unified* baseline model but find it prone to overfitting, which the experimental evaluation will closely monitor. For completeness, this work will investigate both approaches, the impact of pre-training, and their robustness to privileged modalities. Figure 4.4 shows their original model pipeline.

The authors use a ResNet-18 backbone with a two times wider encoder for the unified model to match the number of trainable parameters. This work replaces the backbone by a ResNet-50 to 1) obtain a similar number of model parameters as the VOT-S (*cf.* Table 4.2), 2) and approximately match the ImageNet top-1 accuracy of ViT-S with DINO pre-training as a proxy for the model expressiveness (linear probing results from

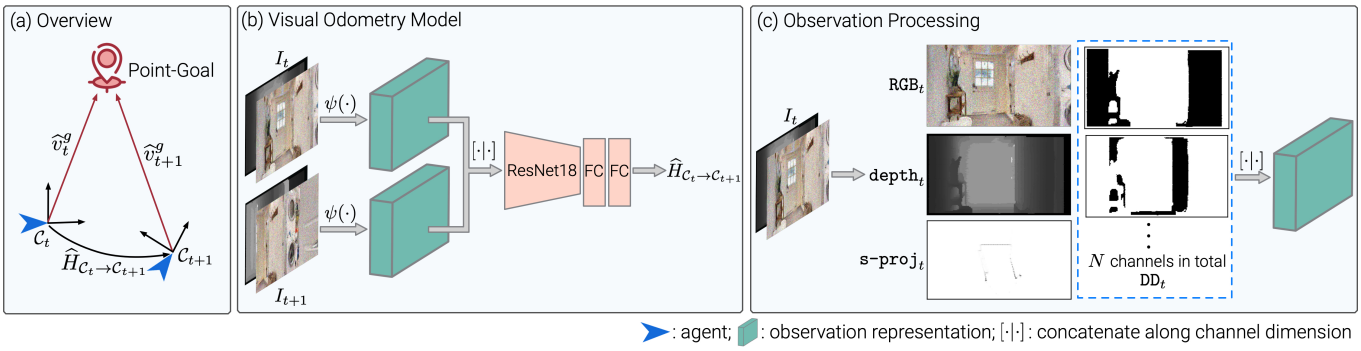


Figure 4.4.: a) This work follows large parts of the problem formulation from Zhao *et al.* [3], *e.g.*, updating the agent's goal with an estimate of the transformation matrix $\widehat{H}_{C_t \to C_{t+1}}$ computed from observations (here: I_t, I_{t+1}). b) The architecture is based on a ConvNet backbone (ResNet-18) and utilizes various pre-processing steps ψ to deal with the noise in RGB and `Depth`. c) `Depth` is discretized into DD with N channels and used to produce a top-down projection `s-proj`. Figure from [3].

[69]: ResNet-50 (linear) 75.3%, (KNN) 67.5%; ViT-S (linear) 77.0%, (KNN) 74.5%). As shown by Dehghani *et al.* [138], the comparison based only on parameters is flawed, sparking the need for this substitution.

The official code repository [137] suggests to train separate models for `left` and `right` with dataset augmentation, a learning rate of $2.0e^{-4}$, and without invariance losses. Subsequently, both models should be fine-tuned with a learning rate of $1.5e^{-4}$ and with invariance losses. This work follows the procedure by training separate models for 75 epochs (with 10 warm-up steps), and fine-tuning them for 75 epochs (with 10 warm-up steps). The `fwd` action model is trained for 150 epochs (with 10 warm-up steps), a learning rate of $2.e^{-4}$ and without data augmentation or invariance losses. The authors also suggest a dropout rate of 0.2 on the final MLP, which this work also considers. They also provide a unified model consisting of a single ResNet-18 backbone whose visual feature output gets concatenated to an action embedding. The resulting vector is passed to an MLP to regress the VO parameters. Again, this work extends this model to a ResNet-50 but forgoes the widening proposed by [3] to increase model parameters as it already has as many as a comparable VOT-S. In contrast to Zhao *et al.* [3], a fair comparison of separate and unified models is made by also training the latter with proposed geometric invariance losses and data augmentation. Table B5 contains additional details on the training setups.

Comparing the updated and this work's proposal to the original results of Zhao *et al.* [3], *i.e.*, training setup, model size, and training data, Table 4.4 also shows the results from the original publication. While those numbers report results over three random seeds, this work will only evaluate a single one due to computational constraints.

4.4. Results

4.4.1. Ablation Study

An ablation study identifies the impact of different input modalities and model design choices. Table 4.3 shows the results for a VOT-S trained from scratch, *i.e.*, without pre-training. First, observe the `Blind` methods that do not use any observation from the environment as input. Without any information about the action or the

Table 4.3.: Ablation study for a VOT-S. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category. Full results including pre-training in Table C6.

obs o	$[ACT]$	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$	$\mathcal{L}_{val} \downarrow$	$\mathcal{L}_{train} \downarrow$
oracle	✘	**97.8**	**74.8**	**73.1**	**29.9**	–	–
`Blind`	✘	0.0	0.0	5.4	398.7	47.258	48.770
`RGB`	✘	36.8	28.2	61.5	125.2	0.620	0.485
`Depth`	✘	**69.5**	**53.2**	**66.7**	**75.8**	0.226	**0.107**
`RGB-D`	✘	51.5	39.5	62.9	115.0	**0.208**	0.132
`Blind`	✔	13.3	10.0	46.3	251.8	1.641	1.637
`RGB`	✔	12.9	10.0	46.5	243.6	1.645	1.738
`Depth`	✔	**79.7**	**61.3**	**69.5**	**64.0**	**0.107**	**0.006**
`RGB-D`	✔	73.9	56.8	68.1	66.6	0.146	0.014

environment, the agent has to randomly choose an action and infer the corresponding VO parameters. This behavior reflects itself in the success rate of 0%. The extremely low SSPL of 5.4 further clarifies that the agent does not even get close to its goal. Providing the `Blind` model with the proposed $[ACT]$ token turns out to be a strong prior that brings the agent closer to the goal, reaching an SSPL of 46.3. However, since the agent can not react to the actuation noise, it only reached the goal and called `stop` 13.3% of the time.

Access to `RGB` or `Depth` allows the VO model to adjust to the unpredictable displacement. These observations further have an information overlap with the $[ACT]$ token, meaning the action can get recovered from the observations. This overlap manifests itself in the results of models that only use visual observations and outperform the `Blind` model with $[ACT]$ token. A combination of both leads to stabler training with a smoother learning curve. With the additional knowledge, the model can better learn from ambiguous or corner cases where the visual observations do not provide explicit information about the action. For example, a `fwd` action colliding with the wall and experiencing actuation noise to the left is difficult to distinguish from a `left` that turns less than $30°$ due to noise (*cf.* Figure 4.2b). Because ViTs require a lot of training data which is deliberately limited in this work, the $[ACT]$ token finds its way into all further VOT models.

Throughout all experiments, `Depth` shows the best results. This behavior supports the assumption that `Depth` contains valuable information about the geometric structure of the underlying scene, which benefits the VO task. The VOT-S using only `RGB` and the $[ACT]$ token runs into a local minimum and converges to predicting the mean transformation of each action and, therefore, reaches a similar performance as only relying on the $[ACT]$ token.

Passing `RGB` besides `Depth` even causes the performance to drop below a model trained solely on `Depth`. Intuitively, a model trained on `RGB-D` should be at least as good as one trained on `Depth`. Processing both modalities is assumed to be a too difficult task to learn from scratch, especially with the low-capacity VOT-S and limited training data. Furthermore, using the same patch embedding to process both modalities requires the model to distinguish `RGB` from `Depth` by learning a suitable positional embedding. Additionally, the VOT does not deploy any techniques to guard against the observation noise, which could exacerbate the situation. The collapse of `RGB` input to the action prior disappears when initializing the model with a more sophisticated pre-training technique, supporting this thesis of learning difficulty (*cf.* Table C6).

Table 4.4.: ConvNet baselines. Metrics S, SPL, SSPL, and d_g reported as e^{-2}. *Italic* results are taken from [3]. **Bold** indicates best results for each category.

action	obs o	b_ϕ	#samples	pre-train b_ϕ	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$
unified	RGB	ResNet-50	250k	None	45.1	34.3	63.7	95.2
unified	Depth, DD, s-proj	ResNet-50	250k	None	59.6	45.2	65.2	**81.0**
unified	RGB-D, DD, s-proj	ResNet-50	250k	None	**64.5**	**48.9**	**65.4**	85.3
separate	RGB-D, DD, s-proj	3×ResNet-50	250k	None	**22.4**	**13.8**	**31.5**	**305.3**
unified	RGB	ResNet-50	250k	supervised	33.5	25.4	59.8	136.3
unified	Depth, DD, s-proj	ResNet-50	250k	supervised	**54.7**	**41.4**	64.2	**93.6**
unified	RGB-D, DD, s-proj	ResNet-50	250k	supervised	53.3	40.8	**64.7**	95.1
separate	RGB-D, DD, s-proj	3×ResNet-50	250k	supervised	47.7	36.0	62.6	103.5
DeepVO [59]	*RGB*		*1000k*	*None*	*50.0*	*39.0*	*65.0*	*93.0*
separate [3]	*RGB-D, DD, s-proj*	*3×ResNet-18*	*1000k*	*None*	***81.0***	***62.0***	***70.0***	***51.0***
unified [3]	*RGB-D, DD, s-proj*	*wide ResNet-18*	*1000k*	*None*	*72.0*	*53.0*	*65.0*	*83.0*

4.4.2. State-Of-The-Art Comparison

First, the unified and separate models proposed by Zhao *et al.* [3] are adapted as discussed, and compared to evaluate their performance on the proposed dataset. With the larger backbone and less training data, the unified model outperforms the separate models mainly due to their poor results. This behavior is assumed to stem from the even lower amount of training data that each separate model trains on (*cf.* Table A4 *vs.* `fwd`: 150k, `left,right`: 100k). The hypothesis is strengthened as initializing the backbones with supervised IN-21k pre-training closes the performance gap, suggesting that the models are indeed struggling to learn from the data provided. While the initialization benefits the separate models, they do not match the original performance of the unified approach without pre-training. On the other hand, the unified model does not benefit from supervised pre-training. These results give rise to the assumption that supervised pre-training helps ConvNets deal with less data but hinders them to unlock its full potential. Even though the model is biased towards RGB features due to its pre-training, it can not fully exploit them. It instead focuses on `Depth`, supporting the hypothesis of the generalizability of supervised pre-training on RGB to `Depth` maps. In contrast to [3], the observed behavior suggests that the unified model can indeed leverage the interplay between actions when provided with enough capacity. Furthermore, training on all action types simultaneously reduces the data sample requirements.

Inspecting the original results from the paper [3] (1000k samples, ResNet-18), it becomes clear that the modified models (250k samples, ResNet-50) can not match them. Note, however, that the modifications offer a different view on the relationship between separate and unified VO models, especially in terms of data requirements. Comparing the results to the proposed VOT-S, training from scratch matches performance and outperforms them using plain `Depth` without extensive preprocessing (*cf.* Table 4.3). With appropriate pre-training, the VOT-S can consistently surpass the performance of Zhao *et al.* [3] using fewer data (*cf.* Table C6).

Training the unified models leads to less overfitting indicated by the smaller gap between training and validation loss (*cf.* Table C7). Because the separate model's capacity is $3\times$ the unified model's this behavior is unsurprising. However, as these results stand in contrast to Zhao *et al.* [3]'s work, it is assumed that their

Table 4.5.: Comparison of different pre-training methods on the VOT-B with and without frozen weights. All models use $[ACT]$. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category. Full results in Table C8.

pre-train b_ϕ	train b_ϕ	obs o	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$	$\mathcal{L}_{val} \downarrow$	$\mathcal{L}_{train} \downarrow$
supervised	✘	RGB	**12.6**	**9.3**	45.4	275.96	1.656	1.732
DINO	✘	RGB	11.2	8.4	**46.4**	240.87	1.512	**1.297**
MultiMAE	✘	RGB	11.7	8.4	45.8	**237.56**	**1.466**	1.429
supervised	✘	Depth	13.5	**10.1**	**47.3**	242.21	1.595	1.272
DINO	✘	Depth	**13.6**	9.8	46.6	232.69	**1.331**	**1.149**
MultiMAE	✘	Depth	13.5	9.6	46.6	**229.89**	1.450	1.315
supervised	✘	RGB-D	13.0	9.7	47.5	237.05	1.589	1.371
DINO	✘	RGB-D	**15.1**	**11.3**	**47.8**	**232.91**	**1.419**	1.475
MultiMAE	✘	RGB-D	12.7	9.4	45.8	249.52	1.424	**1.199**
supervised	✔	RGB	**60.2**	**46.0**	**67.2**	68.14	0.350	0.007
DINO	✔	RGB	57.8	43.9	66.3	78.30	0.361	0.004
MultiMAE	✔	RGB	59.3	45.4	66.7	**66.23**	**0.280**	**0.003**
supervised	✔	Depth	82.6	63.9	69.8	59.12	0.064	0.005
DINO	✔	Depth	86.4	66.0	70.6	46.16	0.074	0.013
MultiMAE	✔	Depth	**93.3**	**71.7**	**72.0**	**38.83**	**0.044**	**0.004**
supervised	✔	RGB-D	**91.4**	**69.9**	**71.5**	**38.13**	0.053	0.003
DINO	✔	RGB-D	85.0	65.4	71.1	43.91	0.069	**0.001**
MultiMAE	✔	RGB-D	88.2	67.9	71.3	42.13	**0.051**	0.004

assumption can be traced back to overfitting caused by the model capacities instead of the unified or separate property.

In conclusion, the VOT-S presents a competitive alternative to previous approaches and outperforms them using only 25% of the data. Furthermore, it eliminates the rigidity of the ConvNet approach caused by the predetermined channel size, allowing for an investigation of the model's robustness to input modalities. As hypothesized in Section 4.1.3, the experiments suggest that a unified model can better exploit limited data and achieve better results.

4.4.3. Backbone Initialization

Comparing model performance across different pre-training methods investigates how much the learned features overlap with the VO task. A common practice is to freeze the weights of the visual backbone, i.e., not updating them with the gradient descent procedure, to prohibit the model from alternating the feature extraction. The frozen backbone then encodes the visual observations, while the MLP head learns to regress the VO parameters from the pre-processed representation. Table 4.5 shows the evaluation result of such an experimental setup on the validation and the test set. Furthermore, Table C6 shows corresponding results for a VOT-S with supervised and DINO pre-training. At the time of writing, MultiMAE weights are only available for the ViT-B.

The VOT pre-trained with the DINO method emerges as a good out-of-the-box feature extractor for RGB-D observations. One reason for the observed behavior could be that supervised pre-training based on classification and a multi-modal reconstruction task specializes the representation too much on the downstream task. In contrast, the contrastive approach of DINO does not experience such an external bias. In the domain of RGB and Depth, the pre-training methods do not differ significantly from each other. Surprisingly, the Depth observations leads to even better performance even though the backbone has only seen RGB during training, while the MultiMAE did see RGB-D. These results also hold for supervised pre-training, which generalizes to the new modality. It is assumed that by expanding Depth to 3 dimensions, the models can interpret it as a grey-scale image for which it can still apply the learned patterns. Even though the frozen backbones do not achieve competitive results, the SSPL ≥ 45.0 shows that the features are indeed helpful for navigating towards the goal.

When fine-tuning the visual backbone, opposing behavior arises. The DINO method gets outperformed by both MultiMAE and supervised pre-training. This circumstance suggests that even though the features produced by DINO have the most overlap with the VO task, they do not act as a good initialization for downstream fine-tuning. Especially, MultiMAE with an RGB-D input adapts fast and reaches good performance ($\mathcal{L}_{val} = 0.051$) after only 47 epochs. The fine-tuned models achieve better results and outperform the state-of-the-art models by a significant margin. However, as the VOT-B's capacity increases dramatically by fine-tuning, overfitting also increases. While there is almost no gap between training and validation for the frozen backbones, all fine-tuned versions overfit, indicated by the drastically lower training error. Especially, models trained with RGB input show this behavior caused by the varying visual appearance of the 3D scenes. (*cf.* Figure 2.2). During training, the model remembers distinct visual features and uses those to estimate the VO. When the appearance changes in the unseen validation scenes, the model can no longer extract meaningful features, and performance declines. This issue might also cause the collapse of the VOT-S model trained from scratch to the performance of the Blind model with the $[ACT]$ token (*cf.* Table 4.3). In contrast, Depth nor RGB-D suffer from this problem as the Depth map only contains information about the geometric structure of the scene, which is invariant to the visual appearance. Another issue is the limited availability of ground truth data and the opposing high capacity of the VOT. Because VOT-B and VOT-S overfit on the dataset, the ViT seems to be the root cause for it. The deployment of regularization techniques like weight decay [139, 140] have shown to improve generalization. Even though the VOT-B overfits to training data, validation performance is still higher than the one of VOT-S (*cf.* Table C6).

Among the pre-training methods, the supervised stands out as it outperforms the other approaches on RGB observations. Interestingly, it is also the only approach that surpasses its results on RGB and Depth with the ones achieved RGB-D observations. This behavior only emerges when the model has sufficient capacity, *i.e.*, a ViT-B backbone.

Even though MultiMAE reaches its best performance when passed Depth, it outperforms all other approaches across all experiments. Its resulting SPL of 71.7 and SSPL of 72.0 come close to GPS+Compass performance with SPL of 74.8 and SSPL of 73.1.

4.4.4. Attention Maps

With the availability of pre-trained weights, the ViT-B poses a great baseline to compare the attention maps learned by the different methods. It further allows assumptions about the results achieved by the frozen backbones and the modalities they focus on. Figure 4.5 visualizes the last MHA-layer of the VOT-B and the action left. The attention weights A (*cf.* Equation (2.3)) contain the compatibility of all tokens to one another. Because the presented architecture uses a special token ($[ACT]$) for the VO parameter estimation,

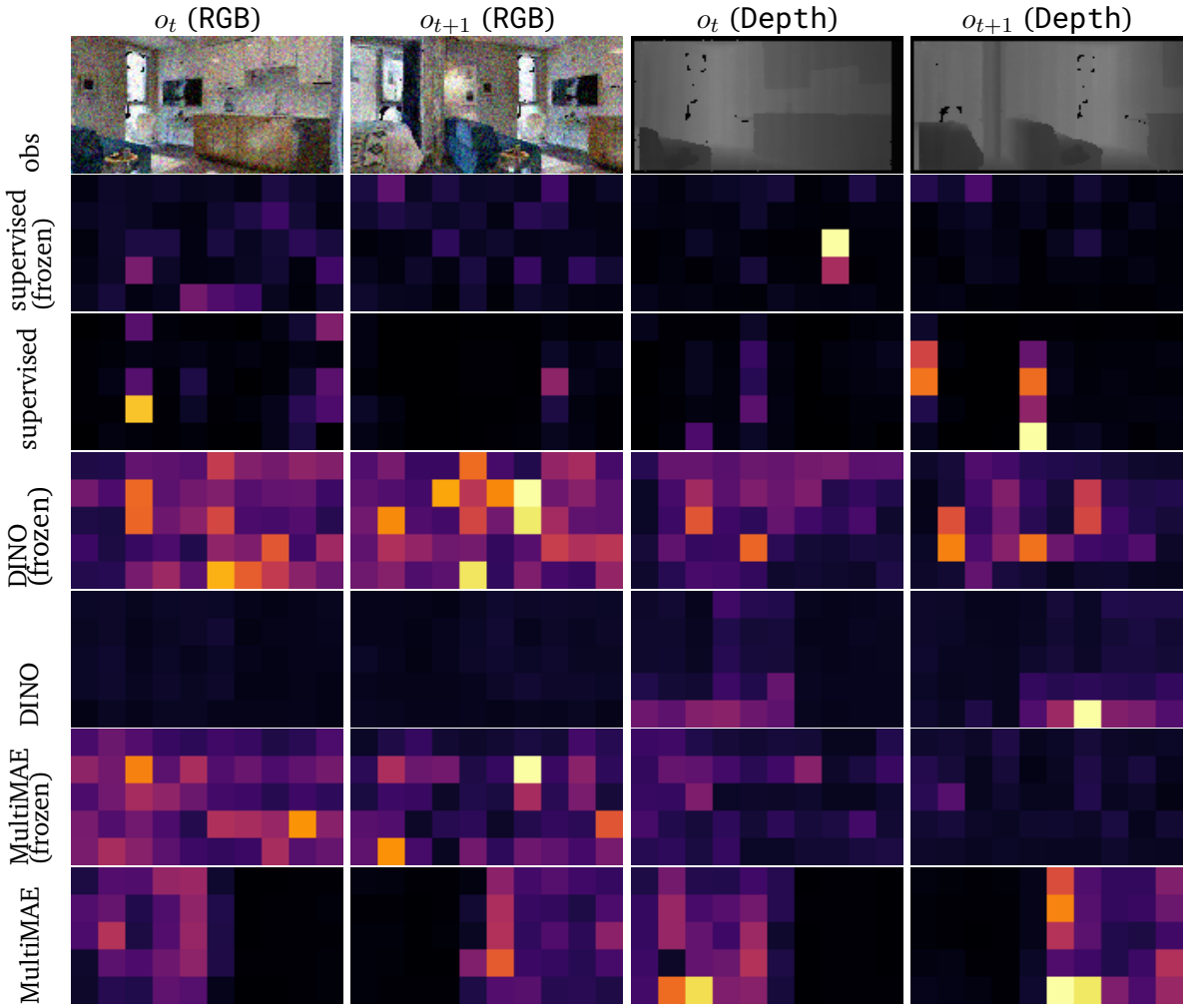


Figure 4.5.: Attention maps of the last MHA-layer of the VOT with ViT-B backbone. The figure shows the visualizations for different pre-training methods and a frozen or fine-tuned visual backbone. Note that MultiMAE shows very intuitive attention masks focusing on regions present in the observations from both time steps $t, t+1$. Action taken by the agent is `left`.

the compatibility of it with the other tokens is relevant for the attention maps. To make the visualization better digestible – VOT-B has 12 attention heads – fusing the heads via the max operator reduces dimensionality and allows for aligning the normalized attention map with the input image. Finally, the maps is normalized to exploit the full range of the color scheme. Attention maps use the *inferno* color map, *i.e.*, the brighter a token's color, the higher the attention put on it by the model.

Inspecting the maps, it becomes evident that interpreting supervised pre-training is less straightforward. This observation coincides with [69], who also find that a classification task leads to less salient attention maps than unsupervised pre-training, *e.g.*, DINO. Most of the attention collapses to a single token, likely caused by its pre-training objective. Looking for distinct objects and related features in the scene might better solve the task of classifying an object into 21k pre-defined classes than attending all over the image. Further, paying close attention to the selected tokens, a hypothesis emerges that the model attends to distinct edges and corners. The frozen backbone also attends less to the `Depth` input, which is intuitive as it was pre-trained on RGB only. After fine-tuning, the model focuses more on `Depth` and attends to moving objects in the scene

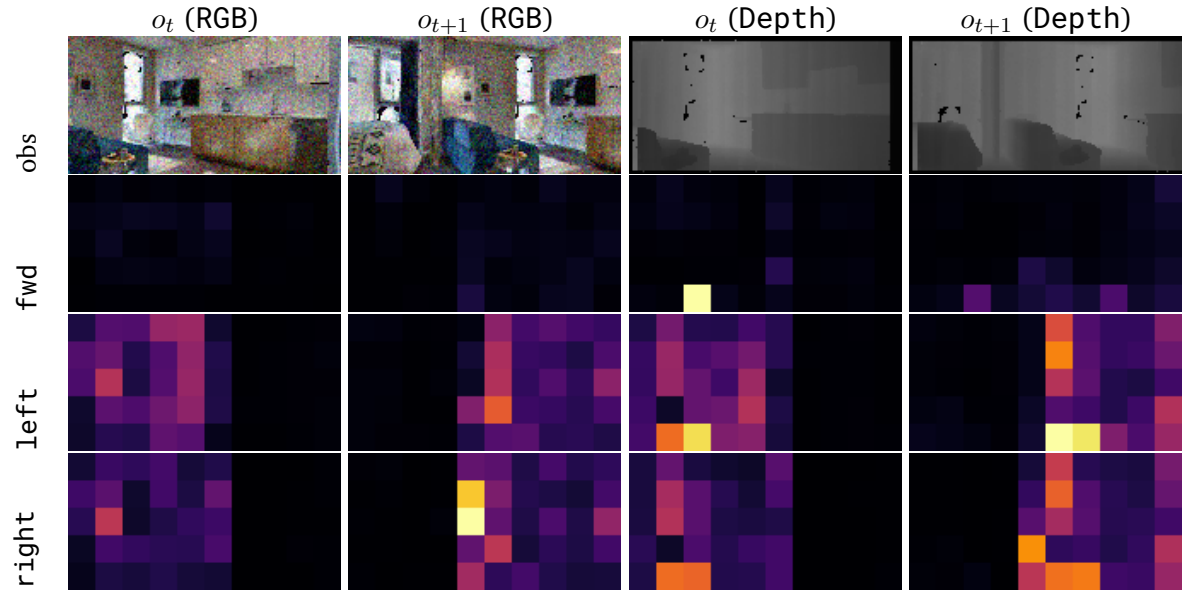


Figure 4.6.: Impact of the $[ACT]$ token on a trained VOT with ViT-B backbone and MultiMAE pre-training. Ground truth action: `left`. Injected actions: `fwd`, `left`, `right`. Embedded `fwd` causes the attention to focus on the center of the image while both `left` and `right` cause a similar behavior of attending the regions of the image consistent across time steps $t, t+1$.

like, *e.g.*, the blue couch.

Trained on RGB only, the frozen DINO backbone attends more to RGB and refocuses the attention towards `Depth` as soon as it gets fine-tuned. In contrast to supervised pre-training, the characteristic attention maps emerge that segment parts, *e.g.*, the window in the image, while also focusing on corners and edges in the scene. As the model gets fine-tuned, the attention becomes more focused on the image features present at both time steps. This refocus makes intuitive sense as those regions provide the most information about the VO while the others become fully occluded or move out of the field of view.

The attention maps become even more distinct for the fine-tuned MultiMAE backbone as the model almost completely neglects the occluded regions. This interpretation is particularly promising as the model has no prior knowledge about the VO task but understands something about the underlying problem structure. For instance, focusing on distinctive edges and features in the image can be interpreted as finding correspondences and estimating the relative pose change. Even though the model was pre-trained on `RGB-D`, the frozen backbone focuses on distinct features in the RGB image, similar to DINO.

While the attention maps for the `right` action are opposing to `left`, `fwd` uniquely attends to the center or outer regions, *e.g.*, the walls and the end of a hallway. Interestingly, the artifacts of the Gibson dataset get ignored by the model. For the visualizations of the `fwd` and `right` observations refer to Figure D1 and Figure D2.

Finally, one should treat these results with caution, as visualizing the attention mechanism of ViTs is subject to active research. So far, no single method has been established. Gildenblat [141] and Gosthipaty *et al.* [142] provide an overview of existing approaches and tricks to make them work.

4.4.5. Action Prior Dependency

The VOT's conditioning on the $[ACT]$ token allows an assessment of the impact of different actions on the attended image regions. The ViT-B backbone trained with MultiMAE showcases the most distinctive attention maps and will be the subject of further examination.

The first example visualized in Figure 4.7 shows that when the model receives the embedding for the `fwd` action, it focuses on the center of the observations in both time steps. Such behavior makes intuitive sense because most visual features will end up in close proximity to their original location due to the nature of a forward movement.

Vice versa, a turning action of $30°$ strongly displaces visual features or even pushes them out of the agent's field of view. This prior knowledge reflects itself in the attention maps when passing action `left` or `right` to the model, visualized in Figure 4.6. The focus shifts to one of the sides of the observations corresponding to the ones that would stay in the vision radius if the agent actually turned. This behavior supports the findings in Figure 4.5 that the model learns to distinguish regions that get occluded from the ones that stay in the agent's field of view. However, it seems the turning direction does not matter for this assessment. The direction might be too obvious, and knowing that the agent rotates provides enough prior information to deal with ambiguous cases. The `right` example in Figure D3 observes a similar behavior.

Finally, whenever the passed action type (move, turn) does not correspond to the ground truth's type, the attention seems to mostly collapse on a single token indicated in the colors bright yellow to white (*cf.* Figure 4.7:`left`,`right`, Figure 4.6: `fwd`, Figure D3: `fwd`). This behavior is similar to the one of supervised pre-training.

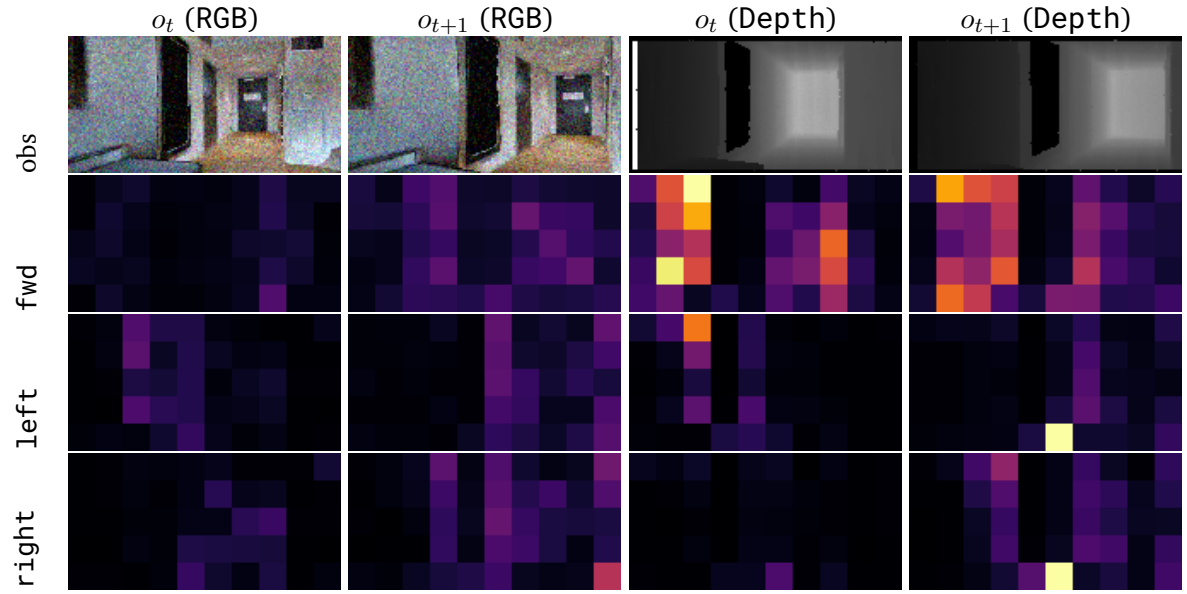


Figure 4.7.: Impact of the $[ACT]$ token on a trained VOT with ViT-B backbone and MultiMAE pre-training. Ground truth action: `fwd`. Injected actions: `fwd`, `left`, `right`. Embedded `fwd` causes the attention to focus on the center of the image while both `left` and `right` try to attend regions of the image that would be consistent across time steps $t, t+1$ in case of a rotation.

4.4.6. Privileged Modalities

Table 4.6.: Modality ablation study for a VOT-B. All models use $[ACT]$, are trained with RGB-D and during test time stripped of one of the modalities (rm obs o) with probability p. Metrics S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category. Pre-training on RGB (supervised, DINO) still shows a strong dependency on it after fine-tuning while multi-modal pre-training (MultiMAE) relies heavily on `Depth`. For most cases, SSPL stays high when removing sensors indicating that an invariance to the input modality is learned by the VOT. Table C9 contains a more extensive ablation of p.

pretrain b_ϕ	rm obs o	p	S	SPL	SSPL	d_g
ImageNet-21k	–	0.0	**91.4**	**69.9**	**71.5**	**38.1**
DINO	–	0.0	85.0	65.4	71.1	43.9
MultiMAE	–	0.0	85.1	65.0	69.5	51.3
ImageNet-21k	`RGB`	0.5	46.0	35.2	63.7	98.7
DINO	`RGB`	0.5	44.0	33.7	63.2	109.9
MultiMAE	`RGB`	0.5	**82.9**	**64.3**	**70.5**	**55.2**
ImageNet-21k	`RGB`	1.0	24.2	18.1	57.0	159.2
DINO	`RGB`	1.0	22.8	17.1	56.2	160.8
MultiMAE	`RGB`	1.0	**75.9**	**58.5**	**69.9**	**59.5**
ImageNet-21k	`Depth`	0.5	**90.6**	**68.9**	**71.2**	**38.9**
DINO	`Depth`	0.5	83.8	64.5	70.8	47.9
MultiMAE	`Depth`	0.5	45.5	34.9	64.4	100.4
ImageNet-21k	`Depth`	1.0	**90.4**	**69.1**	**71.2**	**39.2**
DINO	`Depth`	1.0	81.2	62.7	70.5	50.1
MultiMAE	`Depth`	1.0	26.1	20.0	58.7	148.1

The flexibility of the presented Transformer architecture allows for an evaluation of the VOT's dependency on the input modalities. In case of sensor malfunctioning, *e.g.*, only a single modality might be available. In such a case, the ConvNet's failure is predetermined as it requires a fixed-size input to function. The system then has to construct an alternative input, *e.g.*, zeros, a non-trivial selection. Another option is to fall back on a model trained without the missing modality, requiring additional preparation. Finally, training the network with dummy inputs to simulate missing modalities is also a rather inelegant solution. However, the flexibility in the number of input tokens allows the VOT to pass only a single modality during test time while training on multiple modalities. A sensor might also malfunction for some percentage of an episode, *e.g.*, 10% of `Depth` measurements fail. The following Bernoulli distribution models this behavior as

$$P(x) = \begin{cases} 1-p & for \quad x = 0 \\ p & for \quad x = 1 \end{cases} \tag{4.2}$$

with $P(x_t)$ the probability of deactivating a modality input at time step t and the probability of sensor failure $p \in [0.0, 0.1, 0.25, 0.5, 0.75, 1.0]$. For simplicity, only a single modality (`RGB` or `Depth`) is turned off in each experiment. The sensor failure is simulated for a VOT-B, fine-tuned on the VO task and with various types of backbone pre-training. Table 4.6 gives results for a small selection of this setup, while Table C9 contains the full scope of experiments.

Observing the results, it becomes evident that supervised and DINO pre-training experience drastic drops in performance when access to RGB gets reduced gradually. Even though the attention mechanism learns to focus on the Depth input (*cf.* Figure 4.5), the models still rely heavily on RGB features to make accurate predictions. This assumption gets supported by the fact that removing Depth does not significantly impact the performance. Throughout the experiments, the SSPL is close to the oracle's performance indicating that the agent can still navigate towards the goal. However, the low S, and low SPL suggest that it fails to call the `stop` action at the correct position.

In contrast, the backbone initialized with the multi-modal MultiMAE experiences an almost opposite behavior. Its SSPL even increases when RGB gets removed from the input. However, S and SPL decrease slightly, suggesting that some RGB features might be crucial to determine when to `stop`. An assumption is that the noisy RGB input distracts the model, leading to a lower SSPL, while fine-tuning causes the model to depend on some RGB features to call `stop`. The former is unsurprising as this work does not take any active measures against the observation noise.

To conclude, backbone pre-training strongly influences the modality invariance of the final VOT. Pre-training the backbone using RGB data causes the VOT to rely heavily on the modality, while multi-modal pre-training clings to the more informative Depth input. The experiments show that all VOTs's learn an invariance to the input modality indicated by the high SSPL. However, the PointGoal navigation task reacts quite sensitive to subtle inconsistencies in the VO estimation intensified by the success-dependent metrics S and SPL. To explicitly deal with the sensitivity, input modalities could already get deactivated during training or an additional fine-tuning stage. Nonetheless, the findings are rather impressive as the VOT does not explicitly train to be robust to sensor failure or privileged modalities.

4.5. Discussion & Outlook

This work presents an important step toward multi-modal VO and its robustness to privileged modality access. It proposes the VOT, a ViT-based architecture capable of handling varying input modalities. The model takes as input RGB-D observations of two consecutive time steps and regresses the VO parameters that govern the underlying transformation of the coordinate system. By analyzing its behavior, the VO problem in indoor navigation emerges as a global problem that the VOT can exploit through its large receptive field. Furthermore, the model acts as a swap-in replacement for GPS+Compass sensors in realistic PointGoal navigation, where it achieves near-perfect localization results.

The action taken between two time steps turns out to be a strong prior on predicting the VO parameters. Injecting it into the architecture as a distinct token stabilizes training and boosts the model's performance. Investigating its impact on the attention mechanism shows that it primes the model to attend to image regions present at both time steps.

Additionally, supervised (classification), unsupervised (DINO), and multi-modal unsupervised (MultiMAE) methods get evaluated to reduce the ViT's data requirements. Surprisingly, methods trained on RGB input generalize well to Depth, while multi-modal ones adapt faster to multiple modalities. On the other hand, the limited dataset makes models pre-trained with RGB overfit the visual appearance of the training scene. Overall, Depth as only input emerges as the best performing modality, stressing the importance of the geometric structure for VO.

Studying the robustness reveals that pre-training methods strongly impact the VOT's dependence on different modalities. For instance, the resulting model focuses on the pre-training modality (RGB), while multi-modal

pre-training lets it focus on the most informative one (`Depth`). Simulating sensor failures of various degrees further supports this observation and shows the VOT's robustness to them. The model implicitly learns an invariance to the input modality that allows it to navigate close to the goal even if modality access gets removed. However, training on `RGB-D` input causes a drop in performance compared to training on `Depth` alone. If modality access gets removed, the VO estimation is no longer accurate enough to reach its initial performance on the PointGoal navigation task. Because the success-dependent task definition most likely punishes this inaccuracy, it could be eliminated by an additional fine-tuning step that lets the agent adapt to the inconsistencies in the VO model. Furthermore, masking out modalities during training could explicitly facilitate robustness to privileged information.

Finally, training a unified model for all actions instead of separate ones is shown to be beneficial. In contrast to the findings in [3], it better captures the overlapping information between actions. If the ConvNets capacity is chosen appropriately, it also does not overfit the training data. While the first assumption holds for the VOT, the second does not, showing signs of overfitting. Future work should focus on eliminating this issue by exploring regularization techniques.

Combining the presented advancements in the VOT architecture outperforms the state-of-the-art using only 25% of the data. The model also opens up the possibilities to deal with privileged modality access. Future work should further investigate the multi-modality aspect of the VOT, focusing on, *e.g.*, semantic segmentation [54, 58] or pseudo depth [114]. Those additional input modalities and the robustness to sensor failure become increasingly significant in developing autonomous systems and self-driving [143, 144], where decision-making has a higher stake than in indoor environments.

5. Conclusion

In conclusion, this thesis proposed a novel architecture, the Visual Odometry Transformer, that can deal with multiple modalities and privileged access to those. An interpretation of the model's prediction shows the global structure of the Visual Odometry problem and how the large receptive field of the model can exploit it. Ablations on pre-training methods, an action prior, and training a unified model offer solutions to the lack of training data. The improved architecture outperforms state-of-the-art approaches in the task of realistic indoor navigation. Furthermore, the presented method allows studying the dependence on input modalities and robustness to those. While the models implicitly learn invariance to those, performance is improvable. Additionally, performance on `RGB-D` is still worse than `Depth` only, suggesting that the architecture does not fully exploit the input modalities. Nonetheless, this work lays the foundation for future investigations on robust Visual Odometry models that can deal with privileged multi-modal data.

References

[1] M. Savva *et al.*, "Habitat: A platform for embodied ai research", in *Conference on Computer Vision and Pattern Recognition*, 2019.

[2] S. Datta, O. Maksymets, J. Hoffman, S. Lee, D. Batra, and D. Parikh, "Integrating egocentric localization for more realistic point-goal navigation agents", in *Conference on Robot Learning*, 2021.

[3] X. Zhao, H. Agrawal, D. Batra, and A. G. Schwing, "The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation", in *International Conference on Computer Vision*, 2021.

[4] P. Anderson *et al.*, "On evaluation of embodied navigation agents", in *arXiv preprint arXiv:1807.06757*, 2018.

[5] A. Szot *et al.*, "Habitat 2.0: Training home assistants to rearrange their habitat", in *Advances in Neural Information Processing Systems*, 2021.

[6] F. Xia *et al.*, "Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments", in *IEEE Robotics and Automation Letters*, 2020.

[7] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments", in *arXiv preprint arXiv:1712.03931*, 2017.

[8] E. Kolve *et al.*, "Ai2-thor: An interactive 3d environment for visual ai", in *arXiv preprint arXiv:1712.05474*, 2017.

[9] E. Wijmans *et al.*, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames", in *International Conference on Learning Representations*, 2020.

[10] A. Murali *et al.*, "Pyrobot: An open-source robotics framework for research and benchmarking", 2019.

[11] L. Biewald, *Habitat challenge 2020*, `https://eval.ai/web/challenges/challenge-page/580/leaderboard/1631`, 2020.

[12] M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang, and A. Dosovitskiy, "Do vision transformers see like convolutional neural networks?", in *Advances in Neural Information Processing Systems*, 2021.

[13] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", in *International Conference on Learning Representations*, 2021.

[14] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, "How to train your vit? data, augmentation, and regularization in vision transformers", in *arXiv preprint arXiv:2106.10270*, 2021.

[15] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization", in *International Conference on Computer Vision*, 2015.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *Conference on Computer Vision and Pattern Recognition*, 2009.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", in *arXiv preprint arXiv:1810.04805*, 2018.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, in *Conference on Computer Vision and Pattern Recognition*, 2016.

[19] T.-Y. Lin *et al.*, “Microsoft coco: Common objects in context”, in *European Conference on Computer Vision*, 2014.

[20] S. Antol *et al.*, “Vqa: Visual question answering”, in *International Conference on Computer Vision*, 2015.

[21] J. J. Gibson, *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

[22] J. Bohg *et al.*, “Interactive perception: Leveraging action in perception and perception in action”, in *IEEE Transactions on Robotics*, 2017.

[23] D. Mishkin, A. Dosovitskiy, and V. Koltun, “Benchmarking classic and learned navigation in complex 3d environments”, in *arXiv preprint arXiv:1901.10915*, 2019.

[24] D. Batra *et al.*, “Objectnav revisited: On evaluation of embodied agents navigating to objects”, in *CoRR*, 2020.

[25] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra, “Embodied question answering”, in *Conference on Computer Vision and Pattern Recognition*, 2018.

[26] E. Wijmans *et al.*, “Embodied question answering in photorealistic environments with point cloud perception”, in *Conference on Computer Vision and Pattern Recognition*, 2019.

[27] S. Thrun, “Probabilistic robotics”, in *Communications of the ACM*, 2002.

[28] N. Kojima and J. Deng, “To learn or not to learn: Analyzing the role of learning for navigation in virtual environments”, in *arXiv preprint arXiv:1907.11770*, 2019.

[29] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, “Learning to explore using active neural slam”, in *International Conference on Learning Representations*, 2020.

[30] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, “Neural topological slam for visual navigation”, in *Conference on Computer Vision and Pattern Recognition*, 2020.

[31] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Siddharth, and P. H. Torr, “Playing doom with slam-augmented deep reinforcement learning”, in *arXiv preprint arXiv:1612.00380*, 2016.

[32] J. Zhang, L. Tai, M. Liu, J. Boedecker, and W. Burgard, “Neural slam: Learning to explore with external memory”, in *arXiv preprint arXiv:1706.09520*, 2017.

[33] H. Zhan, C. S. Weerasekera, J.-W. Bian, and I. Reid, “Visual odometry revisited: What should be learnt?”, in *IEEE International Conference on Robotics and Automation*, 2020.

[34] Z. Teed and J. Deng, “Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras”, in *Advances in Neural Information Processing Systems*, 2021.

[35] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, “Deepfactors: Real-time probabilistic dense monocular slam”, in *IEEE Robotics and Automation Letters*, 2020.

[36] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, “Gibson env: Real-world perception for embodied agents”, in *Conference on Computer Vision and Pattern Recognition*, 2018.

[37] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, “A database and evaluation methodology for optical flow”, in *International Journal of Computer Vision*, 2011.

[38] Abhishek Kadian* *et al.*, “Are We Making Real Progress in Simulated Environments? Measuring the Sim2Real Gap in Embodied Visual Navigation”, in *arXiv preprint arXiv:1912.06321*, 2019.

[39] A. Chang *et al.*, “Matterport3d: Learning from rgb-d data in indoor environments”, in *International Conference on 3D Vision*, 2017.

[40] J. Straub *et al.*, "The replica dataset: A digital replica of indoor spaces", in *arXiv preprint arXiv:1906.05797*, 2019.

[41] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian, "Building generalizable agents with a realistic and rich 3d environment", in *International Conference on Learning Workshops*, 2018.

[42] A. Kadian *et al.*, "Sim2real predictivity: Does evaluation in simulation predict real-world performance?", in *IEEE Robotics and Automation Letters*, 2020.

[43] S. Choi, Q.-Y. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes", in *Conference on Computer Vision and Pattern Recognition*, 2015.

[44] T. Sattler, B. Leibe, and L. Kobbelt, "Efficient & effective prioritized matching for large-scale image-based localization", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[45] ——, "Improving image-based localization by active correspondence search", in *European Conference on Computer Vision*, 2012.

[46] D. G. Lowe, "Distinctive image features from scale-invariant keypoints", in *International Journal of Computer Vision*, 2004.

[47] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features", in *European Conference on Computer Vision*, 2006.

[48] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description", in *Conference on Computer Vision and Pattern Recognition Workshops*, 2018.

[49] Y. Shavit and R. Ferens, "Introduction to camera pose estimation with deep learning", in *arXiv preprint arXiv:1907.05272*, 2019.

[50] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]", in *IEEE Robotics & Automation Magazine*, 2011.

[51] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part ii: Matching, robustness, optimization, and applications", in *IEEE Robotics & Automation Magazine*, 2012.

[52] H. Taira *et al.*, "Inloc: Indoor visual localization with dense matching and view synthesis", in *Conference on Computer Vision and Pattern Recognition*, 2018.

[53] C. Szegedy *et al.*, "Going deeper with convolutions", in *Conference on Computer Vision and Pattern Recognition*, 2015.

[54] N. Radwan, A. Valada, and W. Burgard, "Vlocnet++: Deep multitask learning for semantic visual localization and odometry", in *IEEE Robotics and Automation Letters*, 2018.

[55] R. Zhu, M. Yang, W. Liu, R. Song, B. Yan, and Z. Xiao, "Deepavo: Efficient pose refining with feature distilling for deep visual odometry", in *Neurocomputing*, 2022.

[56] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese, "Generic 3d representation via pose estimation and matching", in *European Conference on Computer Vision*, 2016.

[57] Z. Laskar, I. Melekhov, S. Kalia, and J. Kannala, "Camera relocalization by computing pairwise relative poses using convolutional neural network", in *International Conference on Computer Vision Workshops*, 2017.

[58] A. Valada, N. Radwan, and W. Burgard, "Deep auxiliary learning for visual localization and odometry", in *IEEE International Conference on Robotics and Automation*, 2018.

[59] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks", in *IEEE International Conference on Robotics and Automation*, 2017.

[60] ——, “End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks”, in *The International Journal of Robotics Research*, 2018.

[61] A. Dosovitskiy *et al.*, “Flownet: Learning optical flow with convolutional networks”, in *International Conference on Computer Vision*, 2015.

[62] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset”, in *The International Journal of Robotics Research*, 2013.

[63] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume”, in *Conference on Computer Vision and Pattern Recognition*, 2018.

[64] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “Cbam: Convolutional block attention module”, in *European Conference on Computer Vision*, 2018.

[65] S. Niklaus, *A reimplementation of PWC-Net using PyTorch*, `https://github.com/sniklaus/pytorch-pwc`, 2018.

[66] X. Wang, D. Maturana, S. Yang, W. Wang, Q. Chen, and S. Scherer, “Improving learning-based ego-motion estimation with homomorphism-based losses and drift correction”, in *International Conference on Intelligent Robots and Systems*, 2019.

[67] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting”, in *The Journal of Machine Learning Research*, 2014.

[68] R. Partsey, “Robust visual odometry for realistic pointgoal navigation”, 2021.

[69] M. Caron *et al.*, “Emerging properties in self-supervised vision transformers”, in *International Conference on Computer Vision*, 2021.

[70] A. Vaswani *et al.*, “Attention is all you need”, in *Advances in Neural Information Processing Systems*, 2017.

[71] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning”, 2021.

[72] M. Corbetta and G. L. Shulman, “Control of goal-directed and stimulus-driven attention in the brain”, in *Nature Reviews Neuroscience*, 2002.

[73] J. J. Webster and C. Kit, “Tokenization as the initial phase in nlp”, in *International Conference on Computational Linguistics*, 1992.

[74] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks”, *Advances in neural information processing systems*, 2014.

[75] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, in *Neural Computation*, 1997.

[76] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers”, in *European Conference on Computer Vision*, 2020.

[77] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning”, in *International Conference on Machine Learning*, 2017.

[78] X. Chen, S. Xie, and K. He, “An empirical study of training self-supervised vision transformers”, in *International Conference on Computer Vision*, 2021.

[79] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization”, 2016.

[80] T. Xiao, P. Dollar, M. Singh, E. Mintun, T. Darrell, and R. Girshick, “Early convolutions help transformers see better”, in *Advances in Neural Information Processing Systems*, 2021.

[81] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention", in *International Conference on Machine Learning*, 2021.

[82] S. H. Lee, S. Lee, and B. C. Song, "Vision transformer for small-size datasets", in *arXiv preprint arXiv:2112.13492*, 2021.

[83] P. Zhang *et al.*, "Multi-scale vision longformer: A new vision transformer for high-resolution image encoding", in *International Conference on Computer Vision*, 2021.

[84] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era", in *International Conference on Computer Vision*, 2017.

[85] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for semantic segmentation", in *Conference on Computer Vision and Pattern Recognition*, 2021.

[86] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction", in *International Conference on Computer Vision*, 2021.

[87] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows", in *International Conference on Computer Vision*, 2021.

[88] A. Eftekhar, A. Sax, J. Malik, and A. Zamir, "Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans", in *International Conference on Computer Vision*, 2021.

[89] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s", in *Conference on Computer Vision and Pattern Recognition*, 2022.

[90] R. Wightman, H. Touvron, and H. Jégou, "Resnet strikes back: An improved training procedure in timm", in *Advances in Neural Information Processing Systems Workshop ImageNet PPF*, 2021.

[91] I. O. Tolstikhin *et al.*, "Mlp-mixer: An all-mlp architecture for vision", in *Advances in Neural Information Processing Systems*, 2021.

[92] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks", in *arXiv preprint arXiv:1806.01261*, 2018.

[93] J.-B. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers", in *International Conference on Learning Representations*, 2020.

[94] N. Park and S. Kim, "How do vision transformers work?", in *International Conference on Learning Representations*, 2022.

[95] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning", in *Conference on Computer Vision and Pattern Recognition*, 2018.

[96] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization", in *European Conference on Computer Vision*, 2016.

[97] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations", in *International Conference on Learning Representations*, 2018.

[98] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles", in *European Conference on Computer Vision*, 2016.

[99] I. Misra and L. v. d. Maaten, "Self-supervised learning of pretext-invariant representations", in *Conference on Computer Vision and Pattern Recognition*, 2020.

[100] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting", in *Conference on Computer Vision and Pattern Recognition*, 2016.

[101] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping", in *Conference on Computer Vision and Pattern Recognition*, 2006.

[102] A. Atanov, S. Xu, O. Beker, A. Filatov, and A. Zamir, "Measuring the effectiveness of self-supervised learning using calibrated learning curves", 2022.

[103] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations", in *International Conference on Machine Learning*, 2020.

[104] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton, "Big self-supervised models are strong semi-supervised learners", in *Advances in Neural Information Processing Systems*, 2020.

[105] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning", in *Conference on Computer Vision and Pattern Recognition*, 2020.

[106] O. Henaff, "Data-efficient image recognition with contrastive predictive coding", in *International Conference on Machine Learning*, 2020.

[107] A. Bardes, J. Ponce, and Y. LeCun, "Vicreg: Variance-invariance-covariance regularization for self-supervised learning", in *arXiv preprint arXiv:2105.04906*, 2021.

[108] J.-B. Grill *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning", in *Advances in Neural Information Processing Systems*, 2020.

[109] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction", in *International Conference on Machine Learning*, 2021.

[110] W. Stammer, M. Memmel, P. Schramowski, and K. Kersting, "Interactive disentanglement: Learning concepts by interacting with their prototype representations", in *Conference on Computer Vision and Pattern Recognition*, 2022.

[111] A. Jaegle *et al.*, "Perceiver io: A general architecture for structured inputs & outputs", in *arXiv preprint arXiv:2107.14795*, 2021.

[112] V. Likhosherstov *et al.*, "Polyvit: Co-training vision transformers on images, videos and audio", in *arXiv preprint arXiv:2111.12993*, 2021.

[113] R. Girdhar, M. Singh, N. Ravi, L. van der Maaten, A. Joulin, and I. Misra, "Omnivore: A single model for many visual modalities", in *arXiv preprint arXiv:2201.08377*, 2022.

[114] R. Bachmann, D. Mizrahi, A. Atanov, and A. Zamir, "Multimae: Multi-modal multi-task masked autoencoders", in *arXiv preprint arXiv:2204.01678*, 2022.

[115] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners", in *arXiv preprint arXiv:2111.06377*, 2021.

[116] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer", in *Pattern Analysis and Machine Intelligence*, 2020.

[117] M. Memmel, P. Liu, D. Tateo, and J. Peters, "Dimensionality reduction and prioritized exploration for policy search", in *Conference on Artificial Intelligence and Statistics*, 2022.

[118] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics", in *Foundations and Trends in Robotics*, 2013.

[119] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", in *The International Journal of Robotics Research*, 2013.

[120] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search", in *AAAI*, 2010.

[121] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *International Conference on Machine Learning*, 2015.

[122] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", in *arXiv preprint arXiv:1707.06347*, 2017.

[123] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[124] A. D. Laud, *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[125] A. N. Rao, "Learning-based visual odometry-a transformer approach", Ph.D. dissertation, University of Cincinnati, 2021.

[126] H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville, "Modulating early visual processing by language", in *Advances in Neural Information Processing Systems*, 2017.

[127] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", in *International Conference on Machine Learning*, 2015.

[128] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)", in *arXiv preprint arXiv:1606.08415*, 2016.

[129] A. F. Agarap, "Deep learning using rectified linear units (relu)", in *arXiv preprint arXiv:1803.08375*, 2018.

[130] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems*, 2019.

[131] A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto, "Robot learning in homes: Improving generalization and reducing dataset bias", in *Advances in Neural Information Processing Systems*, 2018.

[132] R. Wightman, *Pytorch image models*, `https://github.com/rwightman/pytorch-image-models`, 2019.

[133] R. Bachmann and D. Mizrahi, *Multimae: Multi-modal multi-task masked autoencoders*, `https://github.com/EPFL-VILAB/MultiMAE`, 2022.

[134] L. Biewald, *Experiment tracking with weights and biases*, `https://www.wandb.com`, Software available from wandb.com, 2020.

[135] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity", 2020.

[136] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *International Conference on Learning Representations*, 2015.

[137] X. Zhao, *PointNav-VO*, `https://github.com/Xiaoming-Zhao/PointNav-VO`, 2021.

[138] M. Dehghani, A. Arnab, L. Beyer, A. Vaswani, and Y. Tay, "The efficiency misnomer", in *International Conference on Learning Representations*, 2022.

[139] A. Krogh and J. Hertz, "A simple weight decay can improve generalization", 1991.

[140] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization", in *International Conference on Learning Representations*, 2019.

[141] J. Gildenblat, *Explainability for vision transformers (in pytorch)*, `https://github.com/jacobgil/vit-explain`, 2021.

[142] A. R. Gosthipaty and S. Paul, *Probing vits*, `https://github.com/sayakpaul/probing-vits`, 2022.

[143] C. Badue *et al.*, “Self-driving cars: A survey”, 2021.

[144] D. Omeiza, H. Webb, M. Jirotka, and L. Kunze, “Explanations in autonomous driving: A survey”, 2021.

[145] P. Mirowski *et al.*, “Learning to navigate in complex environments”, in *International Conference on Learning*, 2017.

Appendix

A. Dataset Statistics

Table A1.: Gibson4+ training and testing scenes as listed by [3].

Split	Scenes
Train	Adrian, Applewold, Bolton, Cooperstown, Goffs, Hominy, Mobridge, Nuevo, Quantico, Roxboro, Silas, Stanleyville, Albertville, Arkansaw, Bowlus, Crandon, Hainesburg, Kerrtown, Monson, Oyens, Rancocas, Sanctuary, Sodaville, Stilwell, Anaheim, Avonia, Brevort, Delton, Hambleton, Maryhill, Mosinee, Parole, Reyno, Sasakwa, Soldier, Stokes, Andover, Azusa, Capistrano, Dryville, Haxtun, Mesic, Nemacolin, Pettigrew, Roane, Sawpit, Spencerville, Sumas, Angiola, Ballou, Colebrook, Dunmor, Hillsdale, Micanopy, Nicut, Placida, Roeville, Seward, Spotswood, Superior, Annawan, Beach, Convoy, Eagerville, Hometown, Mifflintown, Nimmons, Pleasant, Rosser, Shelbiana, Springhill, Woonsocket
Validation	Cantwell, Denmark, Eastville, Edgemere, Elmira, Eudora, Greigsville, Mosquito, Pablo, Ribera, Sands, Scioto, Sisters, Swormville

Table A2.: Sample statistics of the dataset $\mathcal{D}$ extended by $15°$ turning actions.

dataset	`forward`	`left` $30°$	`left` $15°$	`right` $30°$	`right` $15°$	collisions	rollouts	total
$\mathcal{D}_{ext}$	141683	26743	27937	26883	26754	26957	4797	250000

Table A3.: Sample statistics of the training and validation sets $\mathcal{D}_{train}^{left,right}$ and $\mathcal{D}_{val}^{left,right}$ with data augmentation of `left` and `right`. Augmented datasets are denoted as $\widetilde{\mathcal{D}}$.

dataset	`fwd`	`left`	`right`	total
$\mathcal{D}_{train}^{left,right}$	0	49870	50130	100000
$\widetilde{\mathcal{D}}_{train}^{left,right}$	0	100000	100000	200000
$\mathcal{D}_{val}^{left,right}$	0	4999	5001	10000
$\widetilde{\mathcal{D}}_{val}^{left,right}$	0	10000	10000	20000

Table A4.: Comparison of the training and validation sets for the presented method and [3].

dataset	`fwd`	`left`	`right`	collisions	rollouts	total
$\mathcal{D}_{train}$	143697	53140	53163	28255	4891	250000
$\mathcal{D}_{train}^{forward}$	150000	0	0	18099	5192	150000
$\mathcal{D}_{train}^{left,right}$	0	49870	50130	9809	4590	100000
$\mathcal{D}_{val}$	13990	5542	5468	2809	599	25000
$\mathcal{D}_{val}^{forward}$	15000	0	0	1804	643	15000
$\mathcal{D}_{val}^{left,right}$	0	4999	5001	966	541	10000

B. Experiment Configurations

Table B5.: Experiment configurations. Maximum gradient norm clipping (clip GN). Action embedding ($[ACT]$). Flip left/right data augmentation ($\widetilde{\mathcal{D}}$). Invariance losses ($\mathcal{L}^{inv}$).

action	obs o	batch size	lr	epochs	warm up steps	clip GN	dropout	$[ACT]$	$\widetilde{\mathcal{D}}$	$\mathcal{L}^{inv}$	#gpus	#params
VOT-S												
unified	RGB-D	256	0.0002	150	10	1.0	0	✔	✔	✔	1	22-30M
unified	RGB/Depth	512	0.0002	150	10	1.0	0	✔	✔	✔	1	22-30M
VOT-B												
unified	RGB-D	128	0.0002	150	10	1.0	0	✔	✔	✔	2	86-103M
unified	RGB/Depth	384	0.0002	150	10	1.0	0	✔	✔	✔	1	86-103M
VOT-B (MultiMAE)												
unified	RGB-D	128	0.0002	150	10	1.0	0	✔	✔	✔	2	86M
unified	RGB/Depth	128	0.0002	150	10	1.0	0	✔	✔	✔	1	86M
ResNet-50 [3]												
unified	RGB-D, DD, s-proj	256	0.0002	150	10	0.0	0.2	✔	✔	✔	1	27M
fwd	RGB-D, DD, s-proj	256	0.0002	150	10	0.0	0.2	✘	✘	✔	1	27M
left/right	RGB-D, DD, s-proj	256	0.0002	75	10	0.0	0.2	✘	✔	✘	1	27M
left/right	RGB-D, DD, s-proj	224	0.0002	75	10	0.0	0.2	✘	✔	✔	1	27M
unified	RGB	256	0.0002	150	10	0.0	0.2	✔	✔	✔	1	27M
fwd	RGB	256	0.0002	150	10	0.0	0.2	✘	✘	✔	1	27M
left/right	RGB	256	0.0002	75	10	0.0	0.2	✘	✔	✔	1	27M
left/right	RGB	224	0.0002	75	10	0.0	0.2	✘	✔	✔	1	27M
unified	Depth, DD, s-proj	256	0.0002	150	10	0.0	0.2	✔	✔	✔	1	27M
fwd	Depth, DD, s-proj	256	0.0002	150	10	0.0	0.2	✘	✘	✔	1	27M
left/right	Depth, DD, s-proj	256	0.0002	75	10	0.0	0.2	✘	✔	✘	1	27M
left/right	Depth, DD, s-proj	224	0.0002	75	10	0.0	0.2	✘	✔	✔	1	27M

C. Extended Experiment Results

Table C6.: Full ablation study for a VOT-S. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category.

pre-train b_ϕ	train b_ϕ	obs o	$[ACT]$	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$	$\mathcal{L}_{val} \downarrow$	$\mathcal{L}_{train} \downarrow$	$\mathcal{L}_{val}^{forward} \downarrow$	$\mathcal{L}_{val}^{left} \downarrow$	$\mathcal{L}_{val}^{right} \downarrow$
None	✘	*oracle*	✘	**97.8**	**74.8**	**73.1**	**29.9**	–	–	0	0	0
None	✔	`Blind`	✘	0.0	0.0	5.4	398.7	47.258	48.770	5.880	30.014	30.647
None	✔	`RGB`	✘	36.8	28.2	61.5	125.2	0.620	0.485	0.674	0.174	0.154
None	✔	`Depth`	✘	**69.5**	**53.2**	**66.7**	75.8	0.226	0.107	0.106	0.084	0.102
None	✔	`RGB-D`	✘	51.5	39.5	62.9	115.0	0.208	0.132	0.110	0.098	0.092
None	✔	`Blind`	✔	13.3	10.0	46.3	251.8	1.641	1.637	1.564	0.528	0.537
None	✔	`RGB`	✔	12.9	10.0	46.5	243.6	1.645	1.738	1.564	0.529	0.537
None	✔	`Depth`	✔	**79.7**	**61.3**	**69.5**	64.0	0.107	0.006	0.050	0.050	0.051
None	✔	`RGB-D`	✔	73.9	56.8	68.1	66.6	0.146	0.014	0.071	0.066	0.069
supervised	✘	`RGB`	✔	11.2	8.3	44.8	257.6	1.849	1.699	1.663	0.572	0.532
supervised	✘	`Depth`	✔	**13.5**	**10.1**	**47.3**	242.2	1.674	1.585	1.558	0.528	0.534
supervised	✘	`RGB-D`	✔	12.5	9.3	45.3	249.8	1.874	1.658	1.662	0.550	0.577
supervised	✔	`RGB`	✔	53.6	40.9	66.0	76.3	0.383	0.009	0.356	0.125	0.129
supervised	✔	`Depth`	✔	**82.6**	**63.9**	**69.8**	59.1	0.087	0.004	0.030	0.048	0.048
supervised	✔	`RGB-D`	✔	81.8	63.2	69.2	62.6	0.085	0.006	0.015	0.052	0.052
DINO	✘	`RGB`	✔	**14.6**	**11.3**	**48.9**	**228.6**	1.596	1.599	1.540	0.499	0.507
DINO	✘	`Depth`	✔	13.6	9.8	46.6	232.6	1.533	1.506	1.446	0.496	0.497
DINO	✘	`RGB-D`	✔	10.8	7.9	46.3	246.1	1.576	1.510	1.503	0.500	0.510
DINO	✔	`RGB`	✔	41.3	31.7	61.5	112.4	0.550	0.417	0.593	0.141	0.144
DINO	✔	`Depth`	✔	**86.4**	**66.0**	**70.6**	**46.1**	0.083	0.016	0.037	0.044	0.041
DINO	✔	`RGB-D`	✔	80.6	62.5	69.9	53.5	0.069	0.008	0.028	0.036	0.036

Table C7.: Full ConvNet baselines. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. *Italic* results are taken from [3].

action	obs o	b_ϕ	#samples	#params	pre-train b_ϕ	$S\uparrow$	SPL↑	SSPL↑	$d_g\downarrow$	$\mathcal{L}_{val}\downarrow$	$\mathcal{L}_{train}\downarrow$
`fwd`	`RGB-D, DD, s-proj`	ResNet-50	150k	27M	None	–	–	–	–	0.088	0.026
`left`	`RGB-D, DD, s-proj`	ResNet-50	100k	27M	None	–	–	–	–	0.329	0.275
`right`	`RGB-D, DD, s-proj`	ResNet-50	100k	27M	None	–	–	–	–	0.337	0.273
`left/ right`	`RGB-D, DD, s-proj`	ResNet-50	100k	2×27M	None	–	–	–	–	0.650	0.621
separate		3×ResNet-50	250k	3×27M		22.4	13.8	31.5	305.3	–	–
`fwd`	`RGB-D, DD, s-proj`	ResNet-50	150k	27M	supervised	–	–	–	–	0.154	0.041
`left`	`RGB-D, DD, s-proj`	ResNet-50	100k	27M	supervised	–	–	–	–	0.662	0.098
`right`	`RGB-D, DD, s-proj`	ResNet-50	100k	27M	supervised	–	–	–	–	0.724	0.093
`left/ right`	`RGB-D, DD, s-proj`	ResNet-50	100k	2×27M	supervised	–	–	–	–	1.252	0.091
separate		3×ResNet-50	250k	3×27M		**47.7**	**36.0**	**62.6**	**103.5**	–	–
unified	`RGB`	ResNet-50	250k	27M	None	45.1	34.3	63.7	95.2	0.450	0.454
unified	`Depth, DD, s-proj`	ResNet-50	250k	27M	None	59.6	45.2	65.2	81.0	0.311	0.453
unified	`RGB-D, DD, s-proj`	ResNet-50	250k	27M	None	64.5	48.9	65.4	85.3	0.264	0.420
unified	`RGB`	ResNet-50	250k	27M	supervised	33.5	25.4	59.8	136.3	0.794	0.243
unified	`Depth, DD, s-proj`	ResNet-50	250k	27M	supervised	54.7	41.4	64.2	93.6	0.401	0.245
unified	`RGB-D, DD, s-proj`	ResNet-50	250k	27M	supervised	53.3	40.8	64.7	95.1	0.407	0.422
DeepVO [59]	*RGB*		*1000k*	*100M*	*None*	*50.0*	*39.0*	*65.0*	*93.0*	–	–
separate [3]	*RGB-D, DD, s-proj*	*3×ResNet-18*	*1000k*	*3×4M*	*None*	*81.0*	*62.0*	*70.0*	*51.0*	–	–
unified [3]	*RGB-D, DD, s-proj*	*wide ResNet-18*	*1000k*	*12M*	*None*	*72.0*	*53.0*	*65.0*	*83.0*	–	–

Table C8.: Full comparison of different pre-training methods on the VOT-B with and without frozen weights. All models use $[ACT]$. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category.

pre-train b_ϕ	train b_ϕ	obs o	$S\uparrow$	SPL↑	SSPL↑	$d_g\downarrow$	$\mathcal{L}_{val}\downarrow$	$\mathcal{L}_{train}\downarrow$	$\mathcal{L}_{val}^{forward}\downarrow$	$\mathcal{L}_{val}^{left}\downarrow$	$\mathcal{L}_{val}^{right}\downarrow$
supervised	✘	RGB	**12.6**	**9.3**	45.4	275.96	1.656	1.732	1.581	0.511	0.515
DINO	✘	RGB	11.2	8.4	**46.4**	240.87	1.512	**1.297**	1.432	**0.458**	**0.465**
MultiMAE	✘	RGB	11.7	8.4	45.8	**237.56**	**1.466**	1.429	**1.361**	0.461	0.489
supervised	✘	Depth	13.5	**10.1**	**47.3**	242.21	1.595	1.272	1.473	0.509	0.514
DINO	✘	Depth	**13.6**	9.8	46.6	232.69	**1.331**	**1.149**	**1.174**	**0.429**	**0.441**
MultiMAE	✘	Depth	13.5	9.6	46.6	**229.89**	1.450	1.315	1.333	0.469	0.478
supervised	✘	RGB-D	13.0	9.7	47.5	237.05	1.589	1.371	1.515	0.486	0.512
DINO	✘	RGB-D	**15.1**	**11.3**	**47.8**	**232.91**	**1.419**	1.475	1.304	0.449	0.468
MultiMAE	✘	RGB-D	12.7	9.4	45.8	249.52	1.424	**1.199**	**1.298**	**0.447**	**0.454**
supervised	✔	RGB	**60.2**	**46.0**	**67.2**	68.14	0.350	0.007	0.321	0.118	0.118
DINO	✔	RGB	57.8	43.9	66.3	78.30	0.361	0.004	0.328	0.121	0.123
MultiMAE	✔	RGB	59.3	45.4	66.7	**66.23**	**0.280**	**0.003**	**0.239**	**0.100**	**0.103**
supervised	✔	Depth	82.6	63.9	69.8	59.12	0.064	0.005	0.020	0.037	0.035
DINO	✔	Depth	86.4	66.0	70.6	46.16	0.074	0.013	0.026	0.041	0.040
MultiMAE	✔	Depth	**93.3**	**71.7**	**72.0**	**38.83**	**0.044**	**0.004**	**0.014**	**0.025**	**0.023**
supervised	✔	RGB-D	**91.4**	**69.9**	**71.5**	**38.13**	0.053	0.003	0.010	0.034	0.031
DINO	✔	RGB-D	85.0	65.4	71.1	43.91	0.069	**0.001**	0.013	0.043	0.042
MultiMAE	✔	RGB-D	88.2	67.9	71.3	42.13	**0.051**	0.004	**0.009**	**0.033**	**0.030**

Table C9.: Full modality ablation study for a VOT with ViT-B backbone. All models use $[ACT]$, are trained with RGB-D and during test time stripped of one of the modalities (rm obs o) with probability p. **Bold** indicates best results for each category.

pre-train b_ϕ	rm obs o	p	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$
supervised	–	0.00	**91.4**	**69.9**	**71.5**	**38.1**
DINO	–	0.00	85.0	65.4	71.1	43.9
MultiMAE	–	0.00	85.1	65.0	69.5	51.3
supervised	RGB	0.10	77.3	58.9	69.0	53.5
DINO	RGB	0.10	75.3	57.4	68.9	58.5
MultiMAE	RGB	0.10	**87.0**	**66.9**	**70.9**	**44.3**
supervised	RGB	0.25	61.2	47.0	66.3	79.9
DINO	RGB	0.25	60.0	46.1	66.8	76.6
MultiMAE	RGB	0.25	**85.8**	**65.9**	**70.7**	**44.7**
supervised	RGB	0.50	46.0	35.2	63.7	98.7
DINO	RGB	0.50	44.0	33.7	63.2	109.9
MultiMAE	RGB	0.50	**82.9**	**64.3**	**70.5**	**55.2**
supervised	RGB	0.75	30.7	23.5	59.0	135.5
DINO	RGB	0.75	33.1	25.2	60.5	126.2
MultiMAE	RGB	0.75	**79.6**	**61.4**	**70.1**	**58.9**
supervised	RGB	1.00	24.2	18.1	57.0	159.2
DINO	RGB	1.00	22.8	17.1	56.2	160.8
MultiMAE	RGB	1.00	**75.9**	**58.5**	**69.9**	**59.5**
supervised	Depth	0.10	**90.7**	**69.2**	**71.3**	**36.5**
DINO	Depth	0.10	84.4	64.7	70.6	47.6
MultiMAE	Depth	0.10	76.3	58.1	69.1	56.3
supervised	Depth	0.25	**91.0**	**69.2**	**71.0**	**37.0**
DINO	Depth	0.25	83.8	64.6	70.9	45.3
MultiMAE	Depth	0.25	64.0	49.3	68.1	68.7
supervised	Depth	0.50	**90.6**	**68.9**	**71.2**	**38.9**
DINO	Depth	0.50	83.8	64.5	70.8	47.9
MultiMAE	Depth	0.50	45.5	34.9	64.4	100.4
supervised	Depth	0.75	**89.1**	**68.1**	**70.6**	**47.0**
DINO	Depth	0.75	82.3	63.4	70.3	51.2
MultiMAE	Depth	0.75	33.0	25.2	61.6	123.8
supervised	Depth	1.00	**90.4**	**69.1**	**71.2**	**39.2**
DINO	Depth	1.00	81.2	62.7	70.5	50.1
MultiMAE	Depth	1.00	26.1	20.0	58.7	148.1

D. Additional Attention Maps

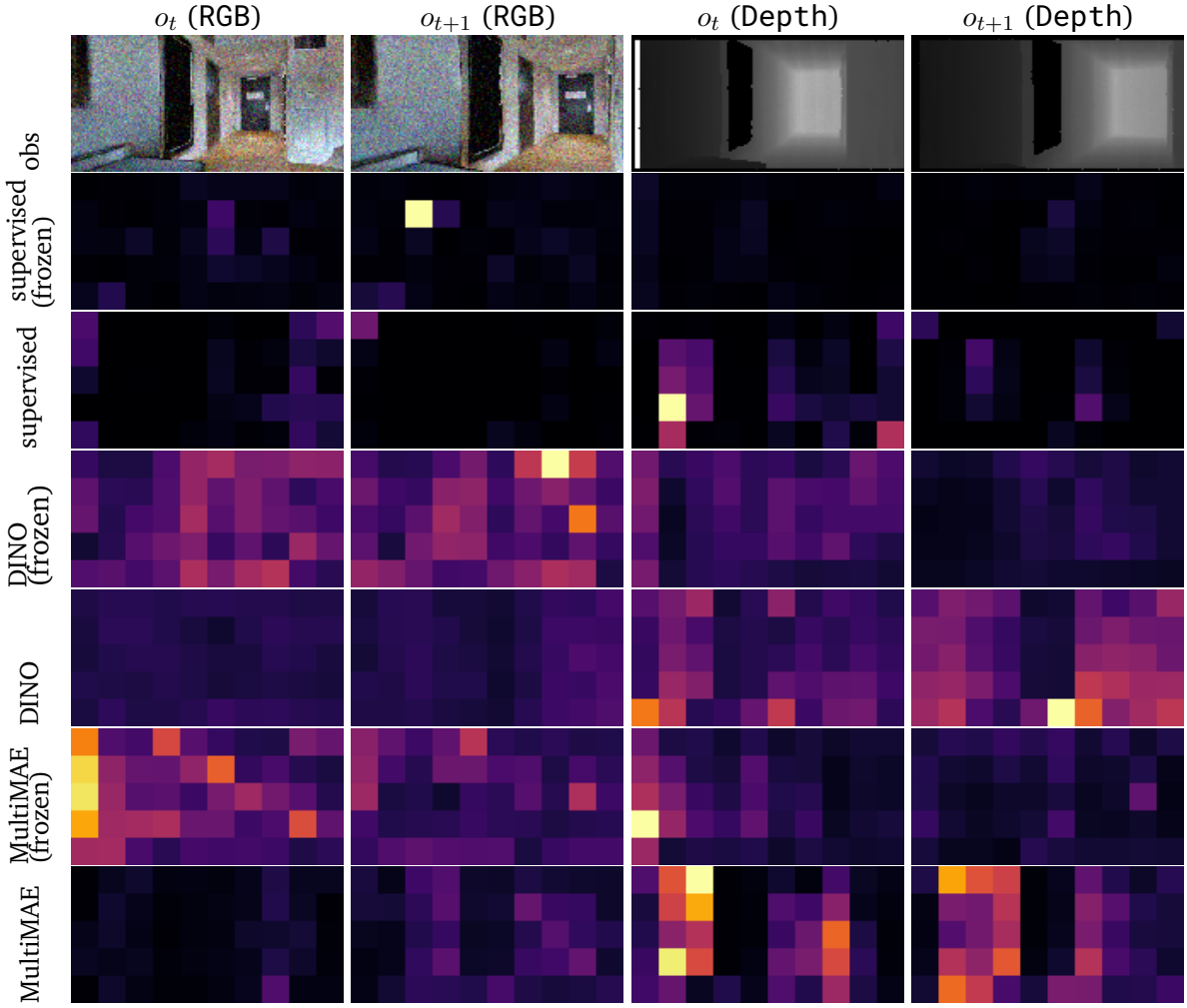


Figure D1.: Attention maps of the last MHA-layer of the VOT with ViT-B backbone. The figure shows the visualizations for different pre-training methods and a frozen or fine-tuned visual backbone. Note that the fine-tuned models attend tokens towards the center of the image as those are present at both time steps $t, t+1$. The artifact (blacked out windows) does not influence the prediction and is largely ignored by the fine-tuned models. Action taken by the agent is `fwd`.

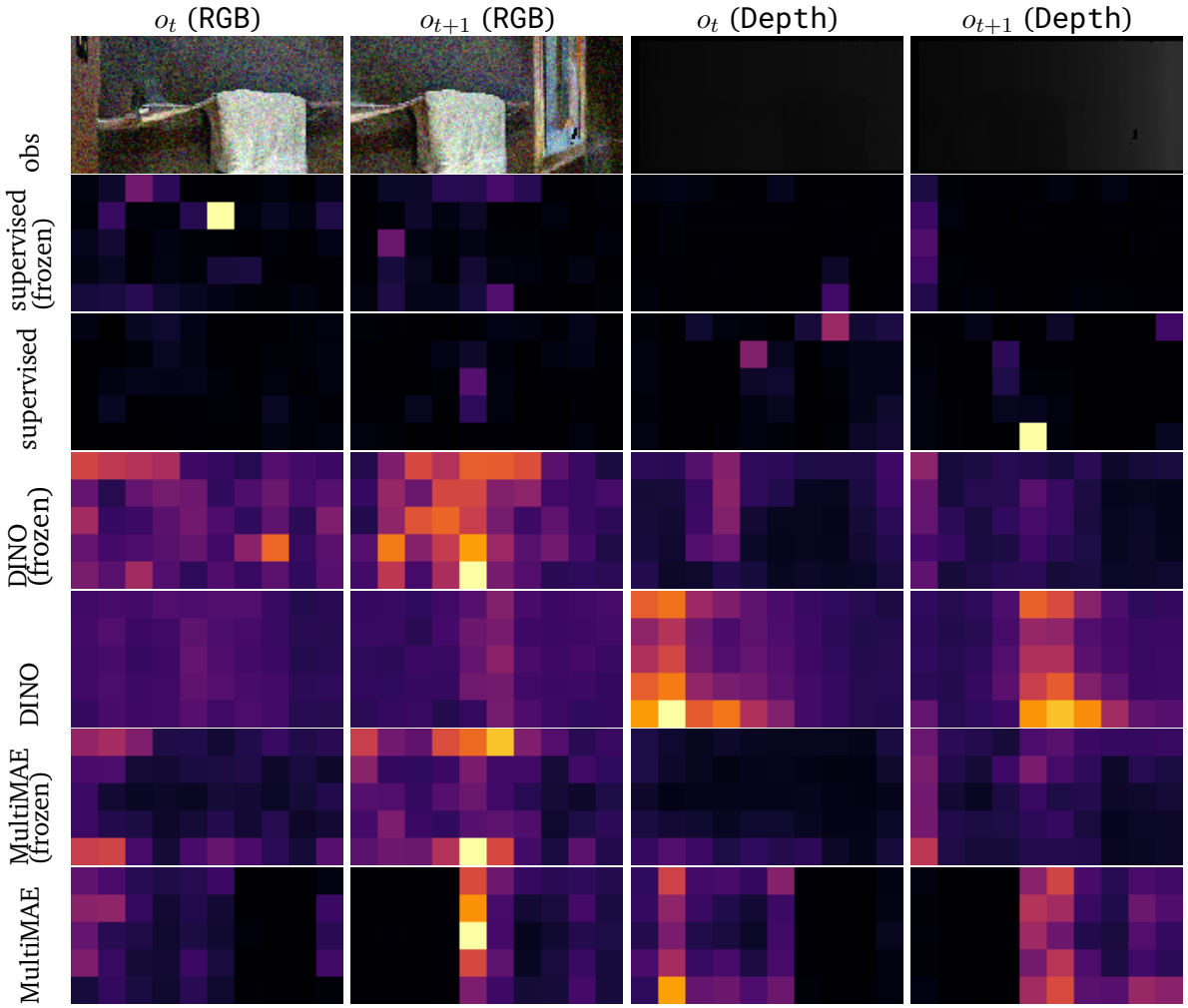


Figure D2.: Attention maps of the last MHA-layer of the VOT with ViT-B backbone. The figure shows the visualizations for different pre-training methods and a frozen or fine-tuned visual backbone. The attention maps are opposing to the ones shown in Figure 4.5 as the rotation is in the opposite direction. Action taken by the agent is `right`.

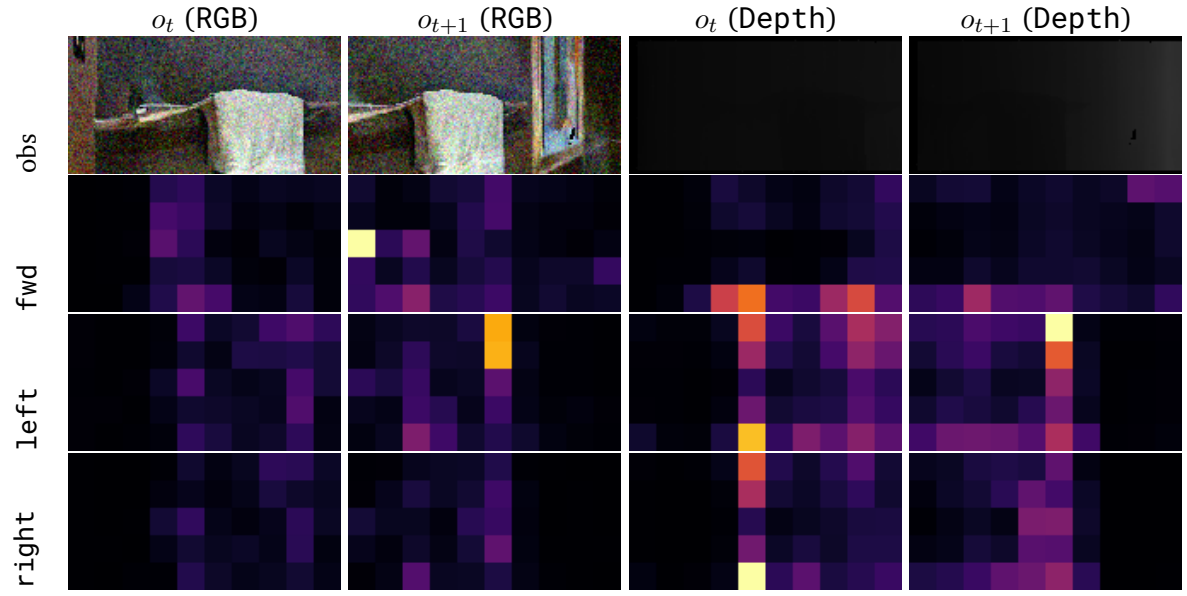


Figure D3.: Impact of the $[ACT]$ token on a trained VOT with ViT-B backbone and MultiMAE pre-training. Ground truth action: `right`. Injected actions: `fwd`, `left`, `right`. Direction of the rotation does not seem to matter. Behavior similar to Figure D3.

E. Auxiliary Depth Estimation Loss

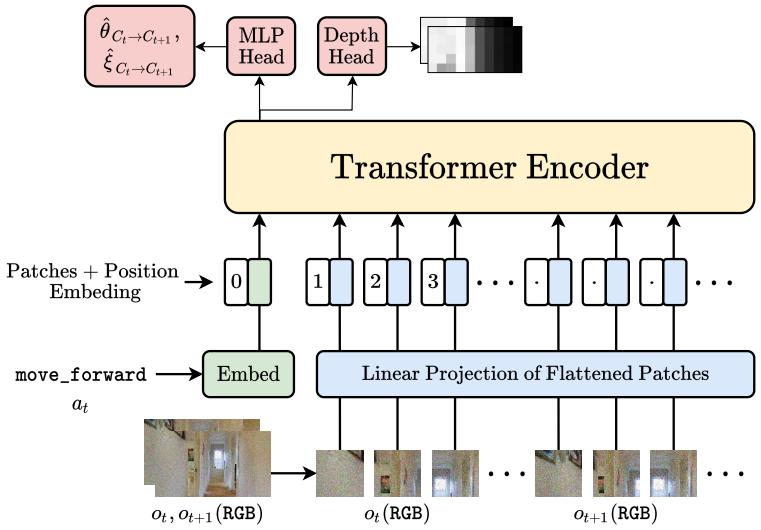


Figure E4.: Single modality VOT for RGB with additional depth estimation head required for the auxiliary depth loss. Visualization inspired by Dosovitskiy *et al.* [13].

Inspired by Mirowski *et al.* [145], who utilize an auxiliary `Depth` estimation to improve navigation from RGB, learning an auxiliary `Depth` estimation poses an alternative to using `Depth` observations as model input. Such an auxiliary task incentivizes the model's latent space to contain `Depth`-like information like the geometric structure of the scene. Assuming this structure is relevant to VO, the results should surpass RGB training. With the input consisting only of RGB tokens, the computational effort decreases due to the quadratic computational complexity of the Transformer *w.r.t.* to the input sequence length. Furthermore, predicting `Depth` at a lower resolution would eliminate the need for highly accurate labels.

Due to increased artifact occurrence in the form of black boxes around the edges (*cf.* Figure 2.3), the `Depth` observations $\langle \boldsymbol{o}_t^{depth}, \boldsymbol{o}_{t+1}^{depth} \rangle$ get pre-processed by cutting off 10 pixels from the top, bottom, left, and right, subsequently down-scaled to a resolution of 10×5 each, and stacked into a 10×10 square depth map to be predicted by the model. The final transformed depth observation $\bar{\boldsymbol{o}}_t^{depth}$ contains information from $\langle \boldsymbol{o}_t^{depth}, \boldsymbol{o}_{t+1}^{depth} \rangle$.

To regress $\bar{\boldsymbol{o}}_t^{depth}$ from visual features $\boldsymbol{v}_{t \to t+1}$, an additional head is added to the VOT architecture (*cf.* Equation (3.8)). This head shares similar components to the MLP head, consisting of a two-layer MLP parameterized by κ consisting of $W_0 \in \mathbb{R}^{dim \times dim_h}, \boldsymbol{b}_0 \in \mathbb{R}^{dim_h}$, and $W_1 \in \mathbb{R}^{dim_h \times 10 \times 10}, \boldsymbol{b}_1 \in \mathbb{R}^3$ with token dimensions $dim = 384$ (ViT-S) or $dim = 768$ (ViT-B), and hidden dimensions $dim_h = dim/2$. A GELU activation function is used between the two layers and a Sigmoid activation function over the output. The model then computes the auxiliary depth prediction $\hat{\boldsymbol{o}}_t^{depth}$ as

$$\begin{aligned} \boldsymbol{v}_{t \to t+1} &= b_\phi(\langle \boldsymbol{o}_t, \boldsymbol{o}_{t+1} \rangle) \\ \text{MLP}_\kappa(\boldsymbol{v}) &= \text{Sigmoid}(\text{GELU}(\boldsymbol{v}W_0 + \boldsymbol{b}_0)W_1 + \boldsymbol{b}_1) \\ \hat{\boldsymbol{o}}_t^{depth} &= \text{MLP}_\kappa(\boldsymbol{v}_{t \to t+1}) \end{aligned} \tag{E1}$$

and use the mean squared error to define the loss:

$$\mathcal{L}^{auxd}_{C_t \to C_{t+1}} = \text{MSE}(\bar{\boldsymbol{o}}^{depth}_t, \hat{\boldsymbol{o}}^{depth}_t) \tag{E2}$$

The resulting optimization objective is then

$$\min_{\phi,\psi,\kappa} \mathcal{L}_{C_t \to C_{t+1}} = \sum_{d_{C_t \to C_{t+1}} \in \mathcal{D}} \left[\mathcal{L}^{reg}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,rot}_{C_t \to C_{t+1}} + \mathcal{L}^{inv,trans}_{C_t \to C_{t+1}} + \lambda \mathcal{L}^{auxd}_{C_t \to C_{t+1}} \right] \tag{E3}$$

with λ, a user-defined hyperparameter to weigh the impact of the auxiliary depth estimation loss. For simplicity, this work assumes $\lambda = 1$. Figure E4 shows how the auxiliary depth estimation loss integrates into the architecture and gives an example of such depth estimates.

Table E10.: VOT-S with an auxiliary depth estimation loss. Losses $\mathcal{L}$, S, SPL, SSPL, and d_g reported as e^{-2}. **Bold** indicates best results for each category.

obs o	$S \uparrow$	SPL↑	SSPL↑	$d_g \downarrow$	$\mathcal{L}_{val} \downarrow$	$\mathcal{L}_{train} \downarrow$	$\mathcal{L}^{auxd}_{val} \downarrow$	$\mathcal{L}^{forward}_{val} \downarrow$	$\mathcal{L}^{left}_{val} \downarrow$	$\mathcal{L}^{right}_{val} \downarrow$
RGB	53.6	40.9	66.1	76.3	0.384	0.009	–	0.356	0.125	0.130
Depth	82.7	63.9	**69.8**	**59.1**	0.088	**0.005**	–	0.030	**0.049**	**0.049**
RGB-D	81.9	63.2	69.3	62.6	**0.085**	0.006	–	**0.015**	0.053	0.052
RGB-D	53.9	41.2	65.0	92.1	0.539	0.010	0.227	0.465	0.251	0.328

Inspecting the results in Table E10, it becomes clear that the auxiliary loss actually hurts performance. Training all methods until convergence might lead to better results but is not guaranteed. Furthermore, Mirowski *et al.* [145] find regression of the Depth map more beneficial. Due to the higher result and Depth from two consecutive time steps, the resulting estimation would be too computationally expensive.