Learning for Perception and Strategy: Adaptive Omnidirectional Stereo Vision and Tactical Reinforcement Learning

Conner Pulling

CMU-RI-TR-24-56
August 5, 2024

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Professor Sebastian Scherer, *chair*
Professor Jeff Schneider
Cherie Ho

Submitted in partial fulfillment of the requirements for the degree of Masters of Science in Robotics.

To my dad for introducing me to robotics, look how far we've come,

Abstract

Multi-view stereo omnidirectional distance estimation usually needs to build a cost volume with many hypothetical distance candidates. The cost volume building process is often computationally heavy considering the limited resources a mobile robot has. We propose a new geometry-informed way of distance candidates selection method which enables the use of a very small number of candidates and reduces the computational cost. We demonstrate the use of the geometry-informed candidates in a set of model variants. We find that by adjusting the candidates during robot deployment, our geometry-informed distance candidates also improve a pre-trained model's accuracy if the extrinsics or the number of cameras changes. Without any re-training or fine-tuning, our models outperform models trained with evenly distributed distance candidates. Models are also released as hardware-accelerated versions with a new dedicated large-scale dataset. The project page, code, and dataset can be found at https://theairlab.org/gicandidates/.

Additionally, the field of reinforcement learning (RL) has transformed strategic game play, enabling AI agents to achieve superhuman performance in games like chess, Go, and StarCraft II. These advancements underscore the potential of RL in handling complex, long-horizon planning tasks against intelligent adversaries with a large search space of potential winning strategies. This project introduces a new competitive multi-phasic strategy game with partial observability and specialized units, demonstrating the use of RL to achieve winning performance. Additionally, this project explores the dynamics of the new competitive strategy game, how certain mechanics lead to different dominant strategies, and how to properly incentivize RL agents to learn winning strategies in this environment.

Acknowledgments

To Basti, I want to thank you for your mentorship, tutelage in the art of research, and for making my Master's journey possible. From my junior year at my undergraduate institution to the end of my MSR journey, you have supported and advised my path. The resources and academic discussions made available by you and the AirLab have facilitated my learning and growth dramatically. I could not have completed this thesis or degree without your support.

To Jeff Schneider, thank you for your mentorship and for taking me on. You are a great mentor and I am thankful for all that I have learned from you so far. Additionally, thank you for being on my thesis committee and advising me during the production of this thesis.

To Yaoyu Hu, thank you for mentoring me throughout my undergraduate research career and into the end of my MSR degree. You were the first person I met from CMU, interviewing me for the Robotics Insitute Summer Scholar (RISS) program, and the one who brought me into this wonderful world of robotics research at CMU. Thank you, you have changed my life immeasurably.

To Professor John Dolan and Rachel Burcin, thank you for supporting my growth through RISS. RISS was the beginning, and through which I have had the opportunity to be surrounding by peers and mentors at the top of their fields and conduct high-level research. I am continually astounded by the level of mentorship you provide to everyone throughout and even after the RISS program, your compassion, kindness, your ability to uplift your community around you is unmatched. RISS has felt like a second family, even years later, and it would not be possible without your involvement.

To Cherie Ho, thank you for all of your support throughout the process of producing this thesis and for being a friend. You are an amazing researcher, a brilliant mentor, and a kind person. I wish you the best of luck in the future and to many more CostCo trips.

To Wenshan Wang and the TartanAir Project, thank you for providing the compiled environments, this saved us a lot of time and enabled our dataset to be varied and impactful to the omnidirectional depth community.

To Siheng Teng, Je Hon Ton, and the folks at DSTA, thank you for working with me on the omnidirectional depth estimation project. You both are brilliant researchers and I am thankful for the chance we had to work together. I wish you luck in all future endeavours

To my friends in MSR, thank you for keeping me sane. Your friendship made the long nights more bearable and the lunch breaks more enjoyable. Here's to more boba and frozen yogurt little treats in the future.

To my father, my mother, my grandmother, and my sister; thank you for letting me talk about fisheye images and omnidirectional depth estimation during dinner. Your support allowed me to grow into the researcher and person I am today.

To my late granderfather who was unable to see me graduate, we made it. I know you would be so proud of me and give me a big bear hug. Thank you for believing in me and for all of the Pokemon cards. And in the end, I too will seek you out amongst the stars.

Funding

I. Omndirectional Stereo Vision

This work was funded by the Defence Science and Technology Agency, Singapore. This work used Bridges-2 at PSC through allocation cis220039p from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #213296.

II. Tactical Reinforcement Learning

This material is based upon work supported by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W519TC-23-C-0030.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Thesis Introduction

When intelligent robotic agents operate in the world, the execution process of their actions can be broadly divided into two main processes: perception and planning. Perception deals with the gathering of information from an agent's environment. This includes tasks such as utilizing sensors, mitigating environmental factors that introduce noise, and preprocessing data into formats that are useful for computational processes. Perception systems straddle the line between the real world and the computational world, using observations of physical processes to deduce intermediate facts that cannot be directly observed. Conversely, planning involves the processing of this information to determine the actions that an agent should execute. Within the vast landscape of potential actions, planning systems evaluate and prioritize choices based on set objectives and experiences learned from past interactions.

Perception systems serve as the critical gateway for deploying robotic agents to perform complex tasks. These systems are indispensable when precise information is required to inform an agent about the next action to take. Adhering to the old adage "Garbage in, garbage out," the quality of input data significantly impacts the accuracy and effectiveness of the agent. While an agent can be designed to be robust against noise and changes in the domain, cleaner and more reliable data that align with known assumptions can greatly enhance system performance. Moreover, some tasks demand the extraction of intricate information from sources that are increasingly noisy, unorganized, and simplistic. On the other hand, planning systems are tasked with narrowing down these inputs to ultimately select the correct action based on the observed state.

In recent years, learning-based systems have dramatically transformed the domains of perception and planning. Advances in machine learning, particularly in deep learning, have enabled perception systems to achieve unprecedented accuracy in object detection, scene interpretation, and real-time response to dynamic environments. These improvements have allowed robotic agents to understand and interact with their surroundings with greater precision and reliability. Simultaneously, in the domain of planning, innovations in reinforcement learning and decision-making algorithms have revolutionized how agents decide and act. These technologies allow agents to learn optimal strategies through iterative processes and adapt their actions based on continuous feedback from their environment, enhancing both the efficiency and adaptability of planning systems.

This thesis represents a dual exploration and contribution to these crucial domains. The first project contributes to the field of learning-based omnidirectional depth estimation, enhancing how robotic agents perceive depth in various settings, thereby improving their interaction with complex environments. The second project investigates the application of reinforcement learning in strategic games involving large and variable action spaces, showcasing how planning systems can evolve to handle highly dynamic and unpredictable scenarios. The thesis is structured into parts, each dedicated to discussing these advancements in depth, their implications for future technologies, and the integration of learning systems into practical applications.

Through detailed exploration of these projects, this thesis aims to highlight the significant potential of integrating advanced learning algorithms in both perception and planning. The goal is to provide a comprehensive understanding of how these technologies can be further developed and effectively applied, paving the way for more sophisticated and autonomous robotic systems in the future.

Part I

Adaptive Omnidirectional Stereo Vision

Chapter 2

Introduction

Distance perception is a key requirement in mobile robots that need to navigate and avoid obstacles. A larger field-of-view (FoV) and faster distance perception enable a robot to more effectively gather information about its surroundings, with omnidirectional sensing being most desirable. Presently, LiDAR devices are the go-to sensors for distance perception due to their accuracy and high update speed. However, LiDARs are mechanically complex, and this complexity increases with an increased number of sampling points. It is technically difficult and prohibitively expensive to achieve both large FoV and high resolution with LiDARs.

Using multiple cameras as a multi-view stereo (MVS) camera set can provide high-resolution omni-directional distance perception with much lower mechanical complexity and cost. Recent research has demonstrated that using multiple cameras with large FoV lenses (e.g., fisheye lens) can achieve omni-directional distance estimation [15, 31, 34]. Compared to LiDAR devices, vision-based distance estimation typically provides larger FoV and denser measurements. However, two challenges prevent MVS-omni-directional solutions from being the go-to sensor choice: 1) they are computationally expensive and; 2) they are difficult to deploy.

The majority of the MVS-omni-directional models, both learning-based and non-learning, utilize a cost volume structure that aggregates visual features by using virtual distance candidates along a viewing direction. The model compares the features at all candidates that are present in the cost volume and picks the best weights for a linear combination of the given candidates. This cost volume approach

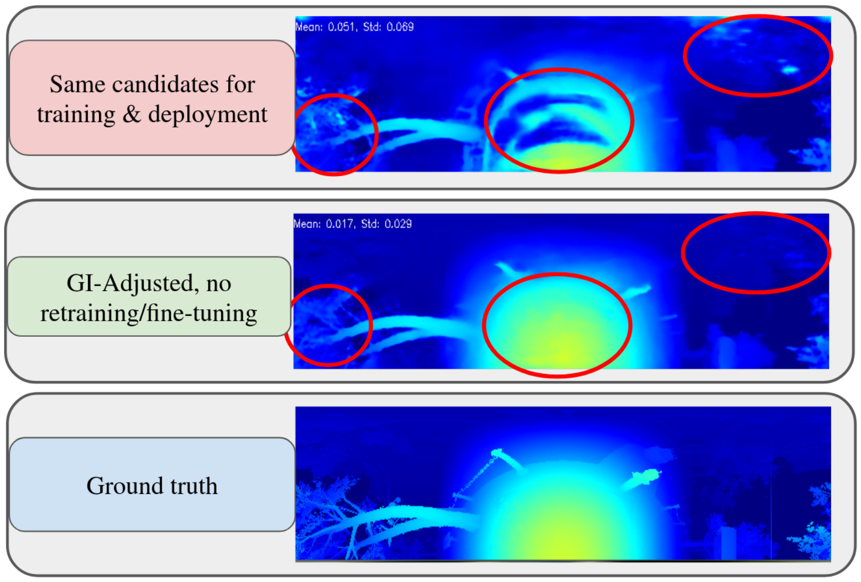

Figure 2.1: After training, using our geometry-informed (GI) distance candidate distribution, the baseline distance between cameras can be changed and the model's performance can be restored without fine-tuning.

consumes a significant amount of computing resources, which grows depending on the number of cameras and the number of distance candidates.

For deployment, cameras in an MVS-omni-directional system typically need to be placed such that maximum FoV can be achieved with minimum occlusions from the robot (self-occlusion). For learning-based methods, if the location or number of cameras is changed to mitigate occlusions, the method typically suffers significant performance degradation as the position of corresponding features in the camera images is changed, hence for the same distance candidates the patterns of accumulated features in the cost volume that differ greatly from training data.

To resolve the above issues related to learning-based visual omni-directional distance estimation, our insight is that we can train a model to utilize a small number of virtual distance candidates by picking distance candidates in a way that is informed by the geometry of the camera configuration. For a known set of camera extrinsics, we can select the candidates such that the positional displacement for the same feature sampled at two consecutive distance candidates are similar across all consecutive pairs of candidates. This ensures a similar pattern of feature accumulation in the cost volume, allowing the model to more effectively determine the best interpolation weights between a pair of consecutive candidates. This enables us to create models with a much lower number of candidates (16 or 8) compared to previous methods, significantly reducing computational cost. We are also able to compute such distance candidates for new camera configurations during deployment, allowing a trained model to be used and maintain its performance even if the camera extrinsics or the number of cameras is changed. In this work, our contributions are:

- A geometry-informed (GI) distance candidates selection method that enables the use of fewer candidates and change of extrinsics for deployment.
- Demonstration of a relaxed version of camera layout that can generate omni-directional distance estimation with self-occlusion explicitly handled and variable translations among the cameras.

After training on our dedicated new dataset, our model can efficiently generate omni-directional distance from multiple cameras with self-occlusion explicitly handled, even if the number and position of the cameras change during physical deployment. The code, pre-trained model, and the dedicated dataset are available through the project webpage.

Chapter 3

Background

Estimating distance from more than one camera is a common and fundamental capability of robot systems. There is a vast body of work that covers various topics within depth estimation as well as a myriad of methods to measure depth. This section will start with a high-level overview will cover current range-sensing methods. Next, an overview of projective geometry and camera models will provide an understanding of how images serve as two-dimensional projections of our three-dimensional world. Finally, an overview of pinhole cameras, plane-sweeping, and multi-view stereo vision will give a current understanding of how multiple camera images can be processed into a cost-volume to produce a depth image. Additionally, the exploration into current multi-view depth estimation methods will motivate the need for configurable and omnidirectional depth estimation With these background sections, the reader will be primed well-informed to appreciate the contributions that this work has made.

3.1 Overview of Range-Sensing Methods for UAVs

Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, employ various range-sensing technologies to navigate and interact with their environment effectively. Each sensing technology comes with its advantages and trade-offs, particularly concerning size, weight, power, and cost (SWAP) constraints, which are critical in UAV applications.

Light Detection and Ranging (LiDAR) systems are known for their precision in generating detailed 3D maps by emitting laser beams and measuring the time it takes for the reflection to return. While offering high accuracy, LiDAR systems tend to be heavy and power-intensive, making them less ideal for smaller drones where payload capacity is limited.

In contrast, photogrammetry, utilizing images from cameras to measure distances, presents a lighter and often more cost-effective option. Traditional setups involve stereo camera systems that infer depth through disparity calculations between images taken from slightly different viewpoints. However, these systems typically require multiple cameras positioned at various orientations to capture the environment fully, increasing the computational burden for image processing.

Fisheye cameras have recently gained popularity in UAV applications due to their wide field of view, which can cover much larger areas with fewer cameras. This reduction in the number of required cameras not only decreases the weight and power consumption but also simplifies the data processing pipeline. The wide-angle lenses of fisheye cameras capture a panoramic view, reducing blind spots and enhancing the UAV's ability to navigate complex environments.

The shift towards fisheye cameras aligns with the ongoing need to optimize SWAP in drone technology. Their ability to provide extensive coverage with minimal hardware makes them an attractive alternative to traditional photogrammetry and LiDAR systems, especially in scenarios where real-time response and agility are paramount.

3.2 Multi-view Stereo Vision

Multi-view stereo (MVS) has a longer history compared with the aforementioned multi-view distance estimation. MVS studies are more focused on reconstructing the 3D geometry of an object or a scene, other than providing distance estimations with respect to a robot. Similar to distance estimation, MVS studies use both non-learning [10][25][6][11][3][35] and learning-based approaches [36][9][37][38][18]. The result of an MVS method is usually a volumetric representation (e.g., voxel grid surface), point cloud, or surface mesh. Inside these learning-based models, a cost volume can be constructed following [36]. Most of the approaches use a reasonably large number of distance candidates. Some works, e.g. [8][12][7], explore multi-scale or adaptive candidates, which may use fewer candidates but need to do the computing in an iterative way, leading to additional computational overhead.

3.3 Omnidirectional Distance Estimation

The most relevant non-learning model is from Meuleman, [15] where they generate distance predictions for a reference fisheye image by selectively fusing information from other fisheye image views. A complete distance prediction is then made by stitching multiple estimations together. They also build a cost volume to aggregate information across different distance candidates. For efficiency, the number of candidates is kept at 32. Since the model is non-learning-based, there is no training and it can be deployed on various camera layouts. This model is one of our main baseline models.

For the learning-based models, SweepNet and OmniMVS, by Won, [31][32][33] are the standouts among the early approaches. Like the non-learning models, SweepNet and OmniMVS will build a cost volume for a fixed number of candidates. This number is configurable but in order to achieve desired accuracy the value is set at around 100 or 200. The cost volume is consumed by the downstream part of the model, typically layers of 3D Convolutional Neural Networks (CNN), and distance values are estimated. Later, Su, [26] implemented a hierarchical version that makes distance predictions on different scales, where at each scale, a cost volume is built in the same way. The above models are trained with a fixed number of cameras and placement. When the camera layout changes, new training and datasets may be required.

Two recent works are closely related to our approach. One conducted by Chen, [4] constructs multiple cost volumes for unsupervised learning. They use feature variance to compare the cost volumes [36]. Our approach is similar with the difference being that we handle self-occlusion explicitly. The other is OmniVidar [34], which turns distance estimation into multiple rounds of binocular stereo estimations. On a high level, the learning-based part of this approach is camera layout agnostic as long as we can cover the final by undistorting and rectifying the input fisheye images along different orientations. However, this process needs to be manually and carefully designed for every new camera layout. Our model can accommodate camera layout change through an easier process with fewer manual procedures.

3.4 Current Challenges for Omnidirectional Depth Estimation

Currently, omnidirectional depth estimation studies typically assumes a fixed camera confirmation. Their dataset, training procedures, and evaluation benchmarks all assume this same camera configuration. Without any mitigation, these learning-based methods fail when presented new camera configurations or are recalibrated. This rigidity in camera configuration makes current omnidirectional depth estimation datasets and pretrained models less useful for anything other than validating paper results.

Additionally, the computational efficiency of current omnidirectional methods are quite low. As stated, many current learning-based methods make use of a cost-volume approach that trades computational efficiency for a drastically reduced search space that relies on chosen guesses at the true depth. Recent studies make architectural strides to become faster, yet another way to make current methods faster would be to reduce the number of guesses that are made in the cost volume. There is currently little to no work on how to choose the hypothetical guesses nor how to reduce the number of guesses that are needed while maintaining an advantageous tradeoff between speed and accuracy.

Chapter 4

Methodology

4.1 Target and Evaluation Configurations

For real-world testing, we use an evaluation board with three fisheye cameras pointed in the same direction and arranged in a triangular formation as in Fig. 4.1 and the

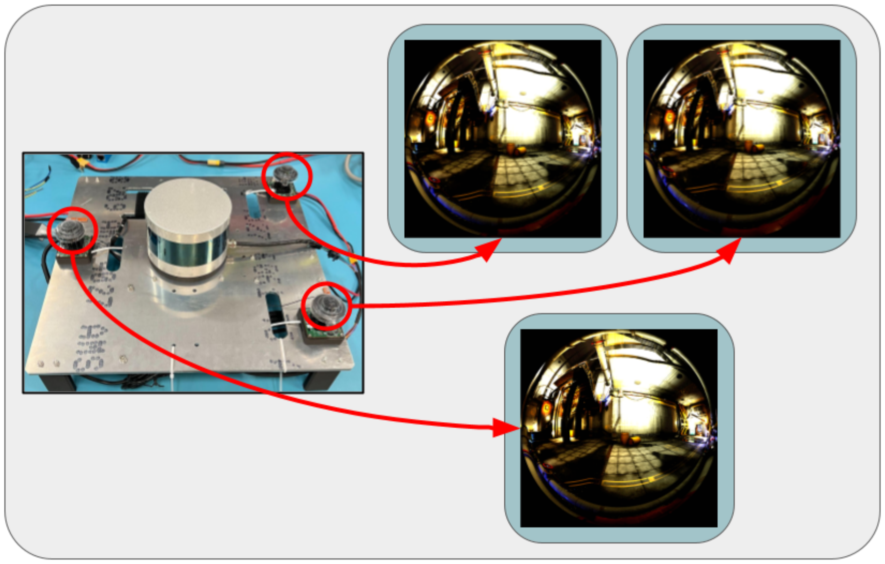

Figure 4.1: Camera Configuration for the evaluation board. Three fisheye cameras are mounted pointing upwards in a triangular formation. A LiDAR, unused for this study, introduces self-occlusions.

training layout in Fig. 5.5. This target configuration enables an aerial robot to have omnidirectional vision by placing cameras safely on top of its body, e.g. Skydio 2+ Drone. Additionally, this target configuration is especially challenging due to the fact that image boundary regions from fisheye lenses are extensively used where good calibration is hard to achieve. We utilize the TartanCalib toolbox to get better calibration results with the Double Sphere camera model[5][28].

4.2 Model Overview

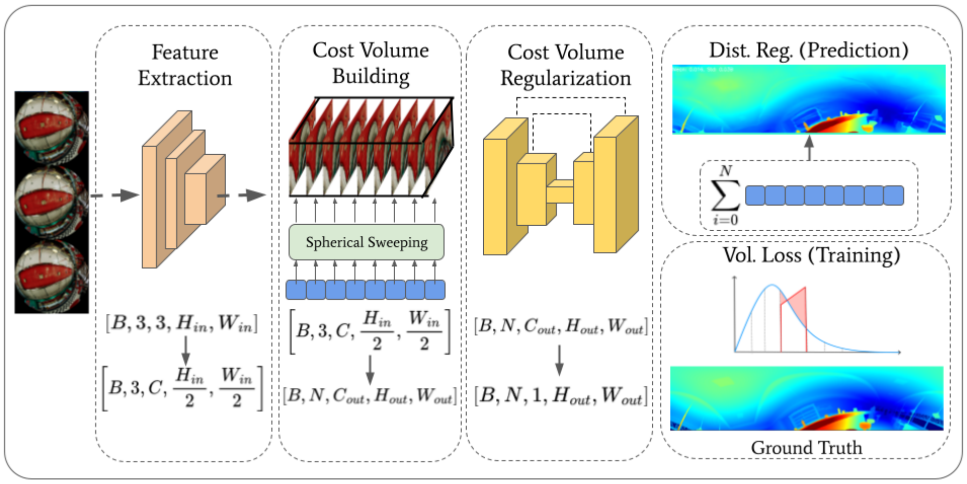
Feature Extraction
Cost Volume Building
Cost Volume Regularization
Dist. Reg. (Prediction)
Spherical Sweeping
$\sum_{i=0}^{N}$
Vol. Loss (Training)
Ground Truth
$[B, 3, 3, H_{in}, W_{in}]$
$\left[B, 3, C, \frac{H_{in}}{2}, \frac{W_{in}}{2}\right]$
$\left[B, 3, C, \frac{H_{in}}{2}, \frac{W_{in}}{2}\right]$
$[B, N, C_{out}, H_{out}, W_{out}]$
$[B, N, C_{out}, H_{out}, W_{out}]$
$[B, N, 1, H_{out}, W_{out}]$

Figure 4.2: Model Overview. The model takes three fisheye images as input during training and performs learned feature extraction with a shared feature extractor, builds a cost volume with spherical sweeping, and regularizes the distance with a 3D U-Net [31].

Similar to [31], our model builds a cost volume from spherically-sweeping learned features and then regularizes this cost volume to achieve a probability distribution of the true distance for each pixel. First, the model takes in three fisheye images during training. Feature maps are extracted from the images with a shared 2D-convolution feature extractor. Next, spherical sweeping is employed using a set of distance candidates to warp the other fisheye images into the reference image frame at the candidate distance. To aggregate all of the views into C channels, differing from prior works in omnidirectional vision with fisheye images, one of our model variants (introduced in Section 5.2) uses feature variance to build the cost volume, similar to [36]. By using feature variance as opposed to concatenating the feature vectors together for each pixel for each warped image, the channel dimension is reduced by a factor of N (number of images). Additionally, because the variance between a set of vectors results in a same-length vector no matter how many vectors there are from the input images, the model can explicitly exclude self-occluded pixels while maintaining the required length of the C dimension. Since the cost volume has one dimension more than the shape of the extracted 2D features, operations like 3D convolutions need to be applied. We utilize a 3D U-Net typed regularizer to process the cost volume into a probability distribution. The probability for each candidate is used in a weighted sum to regress the distance for each pixel.

4.3 Spherical Sweeping

The process of spherical sweeping is crucial for aligning features between cameras, thereby building a cost volume that reduces the search space of possible depth candidates effectively through geometric constraints. If a depth candidate is more accurate, the projections between cameras will be more geometrically consistent. Therefore, the spherical sweeping step is essential for unprojecting query images, rotating or transforming them into the reference image frame, and then projecting them back onto the reference image plane to check correspondences.

Each depth hypothesis transforms the unit camera array from the unprojection function into an actual 3D point that is hypothesized to have that depth. Spherical sweeping poses the question: What if this pixel had this depth? What would the resulting image look like in the reference image plane? The correct depth allows a query pixel to overlay onto its corresponding reference pixel almost exactly.

Rather than regressing out a depth, this method projects the query image onto the reference image plane to test the depth hypothesis. Analogous to plane sweeping used to test disparity hypotheses in general stereo vision, spherical sweeping must accommodate the unique characteristics of fisheye images. Unlike traditional images where correspondences lie along straight epipolar lines, in fisheye images, correspondences lie along curved epipolar paths.

Therefore, a generalized warping operation is required for fisheye images. This sweeping step sweeps a query image across a series of depth images, warping the image into multiple possible configurations where the entire image is assumed to be at a certain depth candidate. This method builds a 3D cost volume, which, when passed to the cost volume regulator, represents a set of possible resultant images. The regulator's task is to propagate information and learn to detect correspondences across these images, increasing the likelihood of finding correct correspondences at certain pixels while decreasing it at others if the depth hypothesis is less likely.

The effectiveness of this step in ensuring accurate depth estimations highlights the necessity of integrating spherical sweeping with cost volume building and regulation techniques, which may be further detailed in subsections dedicated to cost volume building and regulation strategies.

4.4 Standard Deviation Cost Volume Aggregation

The use of equirectangular projection results in a region of high distortion at the top of the projected image. While conventional planar convolution kernels may effectively extract low-level features (e.g. edges, corners), they will face difficulty with high-level features (e.g. textures, object features) which appear distorted differently in different regions of the image. To account for the projection distortion we use a spherical convolution similar to that described in [13], implemented using the deformable convolution with fixed precomputed kernel offsets. Our spherical feature extraction module comprises two ResNet stages of 5 and 10 layers respectively, followed by a spherical convolution to aggregate high-level features. We show that the use of spherical convolution improves distance prediction quality in the high distortion region of the image.

4.5 Geometry-Informed Distance Candidates

Previous work on distance perception commonly used distance candidates spaced evenly in the inverse distance space (hereinafter named EV). In the case of plane-sweeping[36], EV candidates have the property that moving an object between consecutive candidates results in a constant pixel displacement of the corresponding features in feature space.

In the case of sphere-sweeping[15][33], EV candidates generally do not result in constant feature displacement due to the non-linearity of spherical sweeping. However, for small camera baselines, they provide a close approximation, as shown in Fig. 4.3. As previous work on sphere-sweeping has focused on small baseline configurations and large numbers of candidates[31][32][33], the use of EV candidates caused negligible impact on performance.

For better efficiency, we propose to use a small number of geometry-informed (GI) candidates computed for specific camera extrinsics and ensure similar displacement for each step between distance candidates. As feature position in the projected image is proportional to the feature ray angle, GI candidates are obtained by developing distance as a function of ray angle and sampling it with evenly spaced ray angle steps (see Fig. 4.4). Later in the experiment section, we show that the use of GI candidates

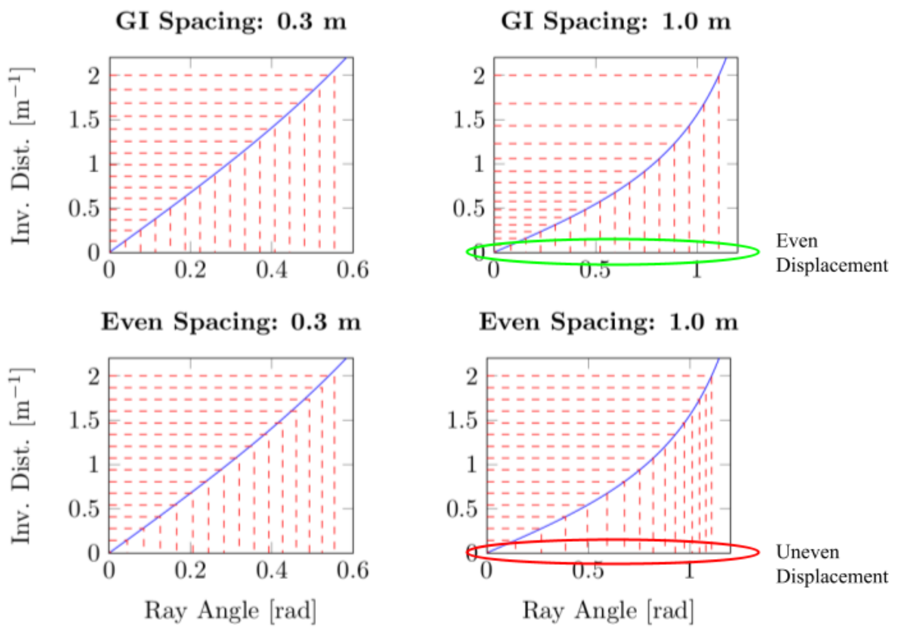

Figure 4.3: GI and EV candidates for different camera spacings. EV candidates approximate constant feature displacement steps for small spacings (baselines), but result in highly uneven steps in large spacings. GI candidates generate constant displacement steps as a function of camera spacing.

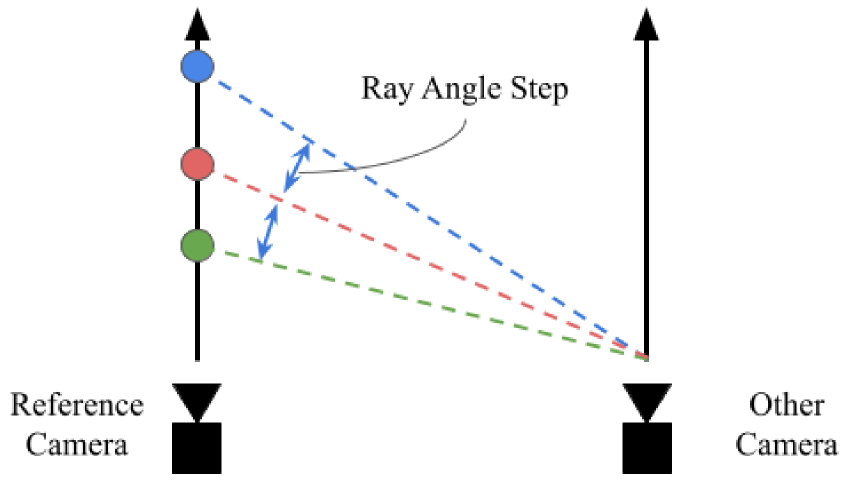

Figure 4.4: Sphere-sweeping geometry. We pick distance candidates that result in constant steps in ray angle corresponding to constant displacements in the projected feature.

improves distance prediction accuracy in the cases of large camera spacing and low candidate count.

4.6 Volume Loss

As seen in previous work [31][34][26], the main loss function of choice for the omnidirectional stereo vision supervised learning problem has been L1 loss on the final distance map. However, there is a rich amount of information in the cost volume itself before aggregation. Before linear combination but after softmaxing, the cost volume represents a probability distribution of which distance candidate is the most likely to be the true distance. In actuality, this probability distribution should look like the interpolation between the two closest distance candidates to the true distance value. Therefore, because the ground truth probability distribution is known and the softmax'd cost volume represents a predicted probability distribution, a soft cross-entropy loss function can be used as a more informative loss function [17]. Combined with the GI distribution described in the previous section, using the volumetric soft cross-entropy loss leads to accuracy gains.

4.7 Synthetic Data Pipeline

A synthetic data collection pipeline was developed to produce a state-of-the-art dataset for training models in omnidirectional stereo vision. The goal was to create a dataset sufficiently photorealistic to bridge the simulation-to-reality gap effectively during model inference. Due to the scarcity of fisheye data in existing datasets, where previous studies often resorted to creating their datasets from lower fidelity simulations, our approach emphasizes higher scene and condition variety.

The data pipeline was constructed using AirSim and simulated drones within the Unreal Engine environment, noted for its exceptional fidelity, including realistic lighting, exposure settings, reflections, and weather effects. These elements contribute to a dataset that closely mimics real-world conditions, making it valuable for practical applications.

Camera configurations critical to our studies were not available in existing datasets. Our pipeline ensures that the camera extrinsics are fixed while allowing flexibility in

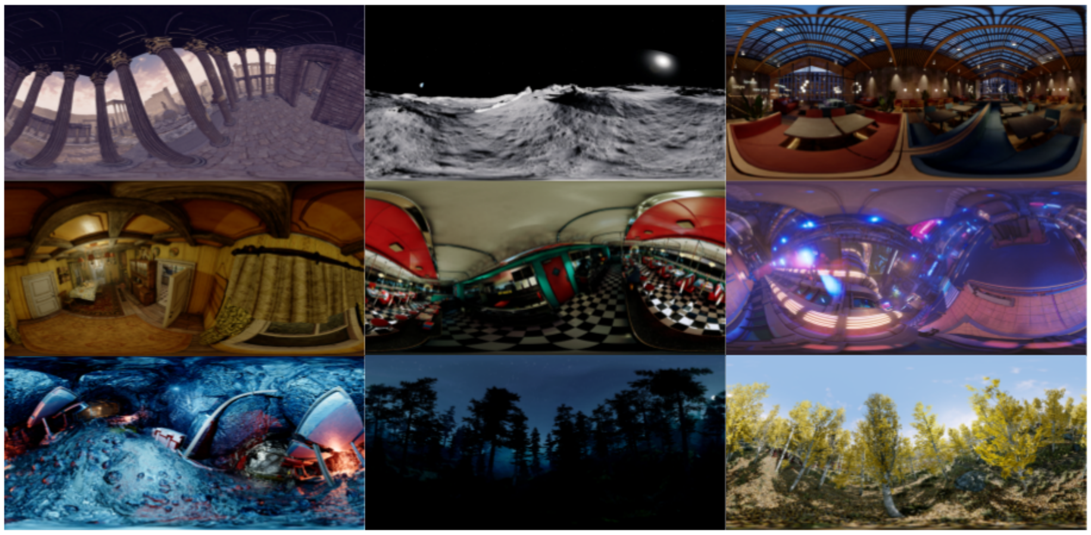

Figure 4.5: Our omni-directional stereo vision with fisheye images dataset consists of about 95K samples from over 60 Unreal Engine 4 high-fidelity simulation environments, manifested in various scene styles.

camera intrinsics through the use of cube maps. These cube maps enable users to configure their fisheye lenses as needed, facilitating future studies involving binocular and trinocular setups with varied camera configurations.

During data collection, the roll, pitch, and yaw of the simulated drone were randomized while maintaining the rigid body assumption across all three cameras. The raw, unprocessed cube maps from data collection allow other researchers to easily apply different fisheye configurations, enhancing the utility of the dataset for other studies.

The cube maps were processed into fisheye images using a custom sampler based on the double-sphere model. This model samples from pinhole cube map images to generate fisheye images, with intrinsics gathered from real-world calibration results using our calibration apparatus. Although this method introduced a visible seam in the fisheye images, it was mitigated by disabling auto-exposure, and did not significantly impact downstream results, although no comprehensive study on this effect has been conducted yet. The environments used for data generation were adapted from the Tartan Air dataset, modified to accommodate our multi-camera setup.

4.8 Synthetic Dataset Characteristics

One of our model development goals is to deploy models on a camera layout similar to that shown in Fig. 4.1. This three-camera plenary setup is the minimum to cover the semi-sphere FOV on top of the plane. This setup also ensures that the robot body in the middle of the cameras will not block the view of more than two cameras, making stereo distance estimation possible for all FOV directions. Currently, no such dataset exists and it motivates us to create a new dataset. In total, 100K samples were collected from over 65 Unreal Engine 4 simulation environments used in the collection efforts of TartanAir [30]. This dataset is over 10x larger than any currently available dataset for omnidirectional stereo vision with fisheye images [31] and is released for download on our project page. The camera layout is the *training layout* in Fig. 5.5. Each sample consists of three RGB-dense distance pairs in fisheye format. There are a large variety of outside, urban, indoor, and natural environments as shown in Fig. 4.5.

Chapter 5

Experimental Procedures & Results

5.1 Model Training

5.1.1 Procedures & Data Augmentation

To enhance the robustness of our model and ensure it generalizes well to various conditions, we employ several preprocessing and data augmentation techniques. These steps are crucial for improving the model's performance and reducing overfitting. The following subsections detail the preprocessing and augmentation methods used in this study.

5.1.2 Preprocessing

The initial step in our pipeline involves preprocessing the fisheye images. Each image undergoes several preprocessing steps to ensure consistency and quality across the dataset. These steps include:

- **Normalization**: All images are normalized to a standard scale to ensure uniformity in pixel intensity values.
- **Resizing**: Images are resized to a fixed resolution to match the input dimensions required by the model.
- **Calibration**: Using the TartanCalib toolbox, we calibrate the images with the Double Sphere camera model to correct any lens distortions and ensure

accurate geometric representations [5][28].

5.1.3 Data Augmentation

To further improve the generalization capability of our model, we apply several data augmentation techniques. Data augmentation helps in artificially expanding the dataset by creating modified versions of the original images. The following augmentations were applied:

Color Jittering

Color jittering is used to introduce variations in the color properties of the images, making the model more robust to changes in lighting and color conditions. The parameters adjusted include brightness, contrast, saturation, and hue. Figure 5.1 Part (b) illustrates the result of applying color jittering to an unaugmented fisheye image.

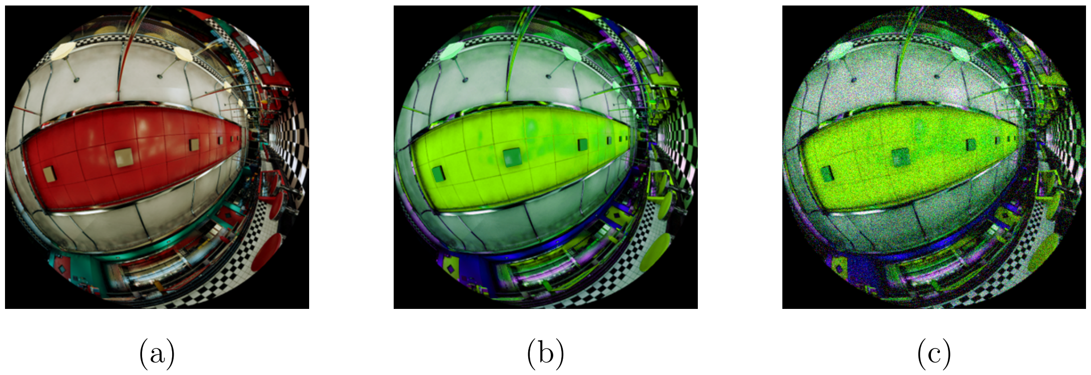

Figure 5.1: (a) Original fisheye image without augmentation. (b) Fisheye image after color jittering. (c) Fisheye image after color jittering and Gaussian noise.

Gaussian Noise

To simulate sensor noise and make the model more robust to noisy input data, we add Gaussian noise to the images. This technique helps the model to learn to ignore noise and focus on the relevant features for distance estimation. The augmented image after applying Gaussian noise to the color-jittered image is shown in Figure 5.1, Part (c).

By combining these preprocessing and data augmentation techniques, we ensure that our model is well-prepared to handle a wide range of real-world conditions, thereby improving its accuracy and robustness in omnidirectional distance estimation tasks.

5.1.4 Convergence Results

In this subsection, we analyze the convergence results of different model configurations through various performance metrics, including L1 Loss, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Structural Similarity Index Metric (SSIM), and Volumetric Loss. The models compared include G16VV, G16V, G16, and E16, with G referring to Geometry-Informed distance candidates, E to the standard Even distribution used in prior works, and the number indicating the number of distance candidates used. Additionally, G16V employs volumetric loss, and G16VV utilizes both volumetric loss and masking augmentation.

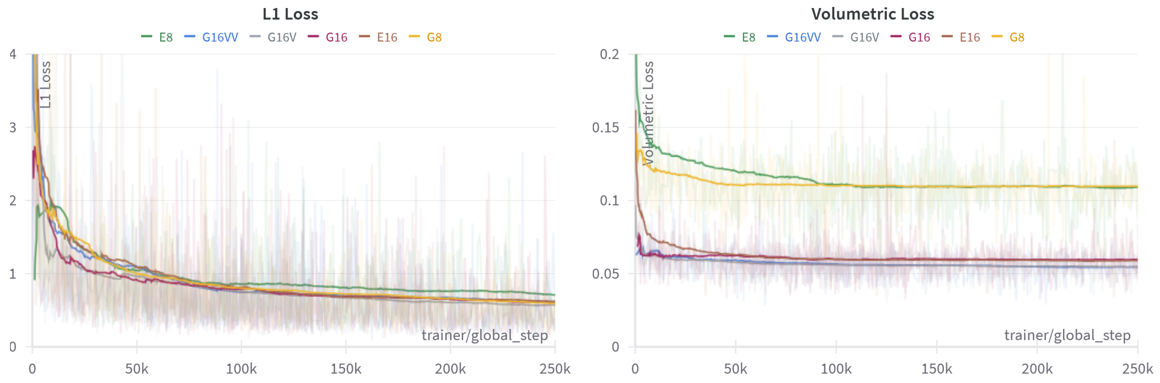

Figure 5.2: Convergence Results for L1 Loss and Volumetric Loss.

L1 Loss

As shown in Figure 5.2, models G16VV and G16V exhibit the lowest L1 loss values throughout the training process, indicating superior convergence. G16 also performs well but slightly lags behind G16VV and G16V. The E16 model has a higher L1 loss compared to the G16 variants, demonstrating the effectiveness of Geometry-Informed distance candidates and volumetric loss.

Volumetric Loss

The volumetric loss, crucial for models incorporating volumetric information, shows that G16VV and G16V significantly outperform the other models, maintaining lower volumetric loss values (Figure 5.2). This emphasizes the benefit of volumetric loss and masking augmentation in these configurations.

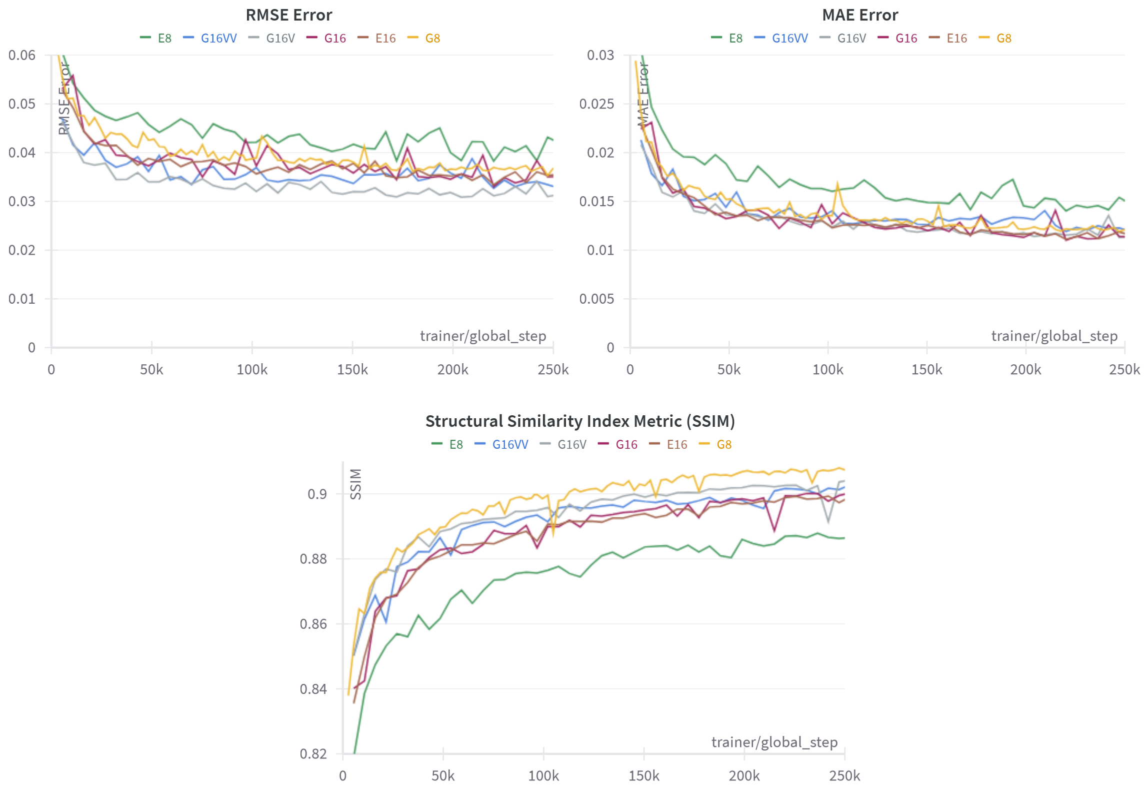

Figure 5.3: Performance Metrics for RMSE Error, MAE Error, and SSIM.

Mean Absolute Error (MAE)

In terms of MAE, G16VV and G16V consistently achieve the lowest errors, closely followed by G16 (Figure 5.3). E16, while performing adequately, shows higher MAE values, reinforcing the advantages of the Geometry-Informed approach and volumetric considerations.

Root Mean Squared Error (RMSE)

Similarly, RMSE results (Figure 5.3) highlight the superior performance of G16VV and G16V, with G16 being competitive but slightly behind. E16 has higher RMSE errors, indicating less robust performance in this metric.

Structural Similarity Index Metric (SSIM)

The SSIM values (Figure 5.3) demonstrate that G16VV and G16V achieve the highest structural similarity with the ground truth, suggesting better quality in the reconstructed images. G16 also performs well, whereas E16 shows the lowest SSIM values among the four models.

Overall Comparison

Overall, models G16VV and G16V consistently outperform others across all metrics, making them the best-performing configurations. The inclusion of volumetric loss and masking augmentation in G16VV provides an additional edge. G16 remains a strong contender but falls slightly behind G16VV and G16V. E16, while performing better than models with fewer candidates or no special augmentations, does not match the performance of the Geometry-Informed variants.

These results demonstrate the effectiveness of Geometry-Informed distance candidates and the enhancements provided by volumetric loss and masking augmentation, establishing G16VV and G16V as the top models in our experiments.

5.2 Model Variants and Camera Layouts

We propose that geometry-informed (GI) distance candidates can directly improve distance estimation. GI candidates can be adapted to most of the MVS-omnidirectional vision models where a fixed number of candidates are applied. In this work, we use a set of model variants to show that GI candidates can work well with small candidate numbers and changes in camera layout.

For a baseline comparison, we build a model for the 3-camera layout shown in Fig. 4.2 using a similar structure as the OmniMVS model[31], the state-of-the-art

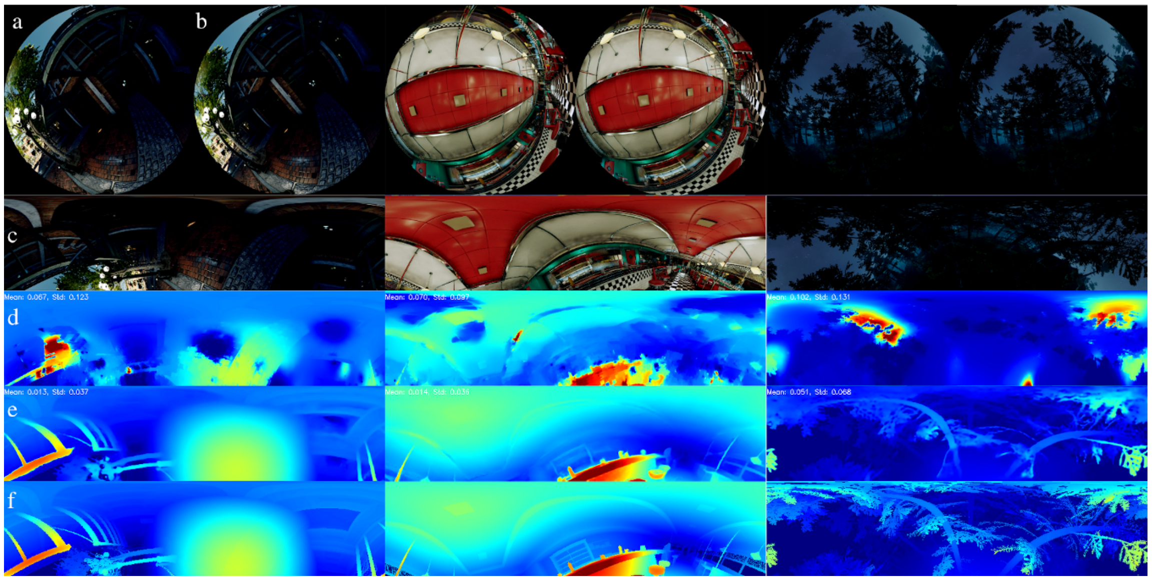

Figure 5.4: Comparison with synthetically-generated images from the unseen environments. a, b: second and third views. c: equirectangular projected reference (first) view. d, e: outputs of RTSS [15] and G16VV (ours). f: ground truth distance aligned with the reference view. In scenes with low light, high-frequency features such as patterns and trees, and thin objects, G16VV is more accurate and it can resolve fine details.

omnidirectional distance estimator. We then have two simple variants from OmniMVS, based on EV and GI candidates. We are targeting models with fewer candidates to have better efficiency. We designate model names E16 and E8 for baseline models with only 16 and 8 candidates, while G16 and G8 for the GI ones. Using the same naming, let G16V be the model trained with the volume loss function. Finally, we also apply the variance cost volume [36] to G16V and get G16VV. One detail about G16VV is

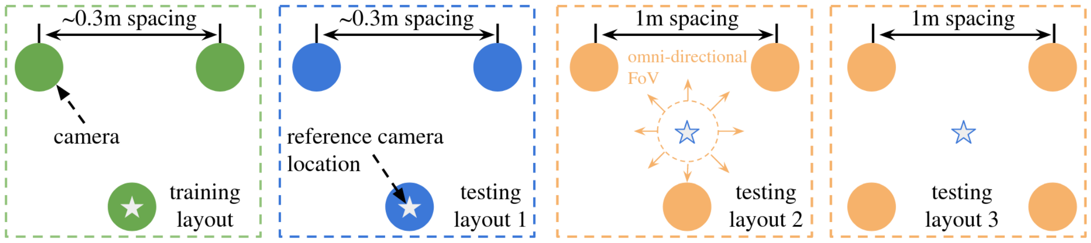

Figure 5.5: Camera layouts in experiments. From left to right: training, testing 1 (same as training), testing 2 (larger spacing, new reference location), testing 3 (larger spacing, new reference location, and more cameras).

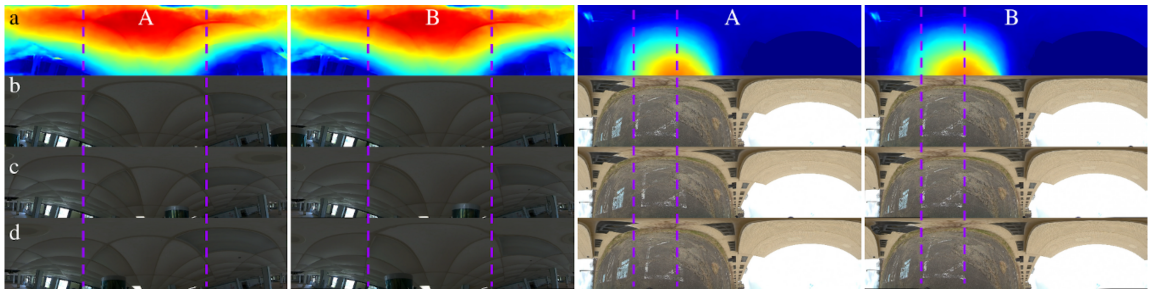

Figure 5.6: Sample results on real-world data. Model G16VV. Row a-d: distance estimation, three camera views. Columns: A - Original input image, B - Input image warped using the predicted distance (Raw a). Purple lines: vertical guidelines. If the distance prediction is good, then the pixels on the purple line across Column B should align among Row b-d. Our model trained only on synthetic data evaluated on real images. The model can be optimized with NVIDIA TensorRT for better inference speed on real robots.

that when calculating the cost volume, we explicitly handle the self-occlusion from the robot. This is done by additionally showing the model a binary mask for every input fisheye image. Such a mask marks non-occluded pixels as valid pixels. When building the cost volume, a variance value is calculated by only considering visual features from the non-masked regions. G16VV is smaller than other variants as a result of using feature variance for building the cost volume. Besides E16 and E8, we also make compare with the RTSS model [15]. In addition to the original RTSS model with 32 EV candidates, we also tested RTSS with GI candidates and 16-candidate variants.

All models are trained on the dataset in Section. 4.8. The distance range is fixed at 0.5-100m during training. For comparison purposes, all models are trained with the same fixed learning rate (0.0001) and batch size (16). We reserve some simulation environments from training and collect ground truth data for evaluation.

Several camera layouts are used in the following experiments. As shown in Fig. 5.5, all models are trained using the *training layout*. We test the models on different layouts representing the change of spacing, number of cameras, and reference location. A location on the plane is picked as the reference and the true omnidirectional distance image is generated w.r.t this reference location in the simulator.

Table 5.1: Comparison using the same camera layout

model	candidates		metrics			time	GPU (MB)	
	type	num	MAE	RMSE	SSIM	(ms)	start	peak
RS-E16	EV	16	0.075	0.129	0.699	146	820	2780
RS-G16	GI	16	0.076	0.129	0.713	140		
RS-E32	EV	32	0.053	0.101	0.776	144	1250	5130
RS-G32	GI	32	0.059	0.105	0.777	146		
E8	EV	8	0.013	0.032	0.862	65	790	1030
G8	GI	8	0.012	0.029	0.867			
E16	EV	16	0.011	0.028	0.876	111	790	1230
G16	GI	16	0.010	0.028	0.877			
G16V	GI	16	0.013	0.028	0.875			
G16VV	GI	16	0.012	0.028	0.872	114	800	1090

EV: evenly distributed candidates. *GI*: geometry-informed. *RS*: the RTSS[15] model.

5.2.1 Evaluation with the Same Camera Layout

We first collected over 1000 samples using *testing layout 1* in Fig. 5.5. Model predictions are compared with ground truth omnidirectional distance images. We use simple metrics including mean absolute error (MAE), mean root square error (RMSE), and the Structural Similarity Index (SSIM) as in [15]. All metrics are computed using the inverse distance (ranging from 0.01 to 2). We use a single NVIDIA V100 GPU for measuring the execution time and GPU memory usage. Several observations can be made from Table 5.1: 1) with 16 or 8 candidates, a model can have very competitive efficiency and GPU consumption compared to the real-time baseline model (RTSS[15], model RS-E16 to RS-G32). 2) GI candidates do not improve the non-learning baseline model[15] and this is the expected behavior. The baseline's performance increases with more candidates. 3) When using very few candidates, such as E8 and G8, the one that uses GI candidates tends to be better. 4) Upon proper training, all learning-based models have similar performance with and without GI candidates, if tested using the same camera layout. Until now, GI candidates have shown marginal performance gain. However, significant improvement is shown in the next section where the testing camera layout is different from the training configuration.

5.2.2 Evaluation with Different Camera Layout

To show the GI candidates' ability to handle a camera layout that is different from the training setup, we collect over 100 samples from the evaluation environments with larger distances among the cameras, as illustrated in *testing layout 2* Fig. 5.5. For this test, we only use the variants that have 16 candidates. In the tests, we apply a trained model twice, one with the candidates it was trained on, and the other with the dedicated new candidates that are calculated concerning the deployed camera layout (denoted as *new* in the following table). Table 5.2 shows that the GI candidates can boost the performance of a trained model when deployed on a camera layout that has longer displacement than the training data. We also observe from Table 5.2 that model G16V, which is trained with our volume loss, tends to have better SSIM values.

Using the G16VV model, since it builds the cost volume with feature variance across all views, we can demonstrate that using the GI candidates, our model can also handle the change of camera number. A separate set of over 100 samples is collected from evaluation environments with four cameras laid out as *testing layout 3* in Fig. 5.5. We show a sample result in Fig. 5.7. The quantitative results are also listed in Table 5.2. G16VV gains better performance from only changing the

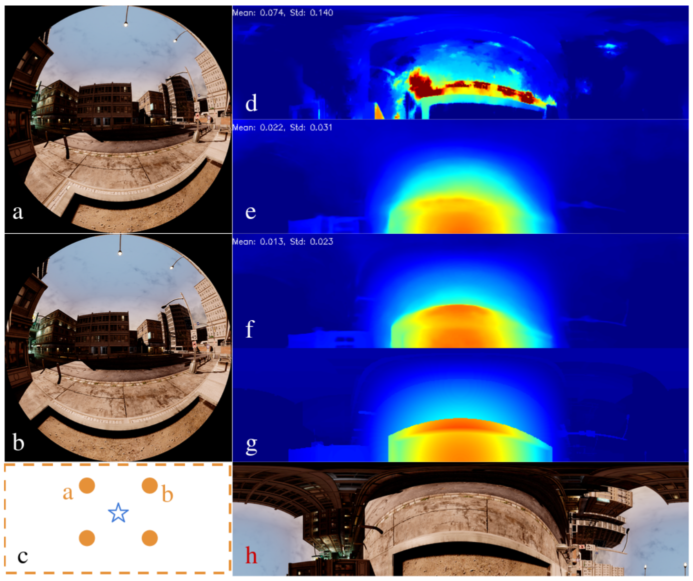

Figure 5.7: Sample results of four-camera layout. a: first view. b: second view. c: camera layout. d: RTSS, e: G16VV w/ training candidates. f: G16VV w/ adjusted GI candidates calculated w.r.t. camera layout. g: ground truth distance map. h: ground truth equirectangular image view. When adjusted GI candidates are used, G16VV is more accurate and resolves more details.

candidate values without any new training. On the speed side, from 3 cameras to 4 cameras, the processing time of G16VV adds only 6ms while the RTSS model experiences a 35ms time increase. On the GPU memory side, since the RTSS model precomputes the best view pairs for every output pixel, its memory does not change. For G16VV, we observe an increase of about 60M Bytes.

Table 5.2: Comparison using new camera layouts

new layout	model	candidates		metrics		
		train	eval	MAE	RMSE	SSIM
3 cam 1m apart	RS-E32	-	EV	0.124	0.217	0.697
	E16	EV	EV	0.033	0.053	0.768
		EV	new	0.018	0.037	0.829
	G16	GI	GI	0.030	0.050	0.786
		GI	new	0.020	0.039	0.823
	G16V	GI	GI	0.030	0.048	0.783
		GI	new	0.020	0.038	0.837
4 cam 1m apart	RS-E32	-	EV	0.090	0.147	0.637
	G16VV	GI	GI	0.024	0.041	0.817
		GI	new	0.016	0.033	0.860

Candidates type: *EV* - evenly distributed, *GI* - geometry informed, *new* - GI for the 1m spacing. All models are trained with a camera spacing of about 0.3m and tested with 1m. *RS*: the RTSS[15] model.

5.3 Deployment & Hardware Acceleration

We deploy the G16VV model on several Nvidia Jetson devices. To leverage the hardware acceleration capability, we convert the entire model using TensorRT. We demonstrate that our optimized model is capable of achieving more than 10 Hz on AGX Orin as shown in Table 5.3. All the deployed models (and intermediate, hardware-independent models) are available on the project website.

Table 5.3: Inference memory and time of accelerated models.

architecture	GPU	inference	
		time (ms)	mem. (MB)
x86-64	RTX3080Ti	11	710
	GTX1070MQ	210	800
NVIDIA Jetson	AGX Xavier	200	600
	Xavier NX	270	1800
	AGX Orin	65	1900

Original model: G16VV. The inference time and memory consumption are measured across 100 consecutive runs. The entire model is accelerated by TensorRT and all the models are available from the project website.

Chapter 6

Conclusions

6.1 Released Materials

This work introduces Geometry-Informed (GI) distance candidate selection for omnidirectional vision models. GI candidate approximate constant feature displacement between distance candidates. Additionally, GI candidates give the model extra flexibility after training: camera spacings (stereo baselines) can be adjusted after training without fine-tuning while maintaining good performance. We develop a set of models with our improvements and compare them against available state-of-the-art baseline models and show accuracy, speed, and memory consumption improvements. We also show that our model, only trained on synthetic data, produces reasonable distance estimations in the real world. Lastly, we release several model variants and our dataset for the use by the community.

6.2 Limitations

While our model demonstrates significant advancements in omnidirectional stereo vision, several limitations have been identified that warrant further investigation and improvement:

Need for Rotation Invariant Features

One of the primary limitations of our current model is its sensitivity to the orientation of objects in the scene. The model's performance degrades significantly when the orientation of the cameras change. This issue arises because the feature extraction process is not rotation invariant. We have not yet attempted data augmentation techniques or more advanced feature extraction algorithms to address this problem. Potential solutions could include pre-warping the RGB images so that everything is already aligned before being fed into the cost volume, or using advanced architectures like vision transformers that can learn rotation-independent features.

Configuration-Agnostic Evaluation Techniques

Our model was specifically trained and evaluated using a three-camera front-facing configuration. Prior studies have used outward-facing cameras. We did not test other configurations due to the high cost of data collection for new physical camera setups. However, it would be beneficial to develop evaluation methods and model architectures that can accept various camera configurations. This would allow a model trained on one dataset with a specific configuration to be evaluated on different configurations without significant performance loss.

Ghost Points

Ghost points occur most frequently at the boundaries of objects, especially sharp and straight ones. While the depth image may look fine from a 2D perspective, projecting it outwards reveals issues. Slight fading at the edges causes ghost points, which significantly impact mapping efforts. These erroneous points are due to the smearing of edges into 3D space caused by slight changes in occlusions from different viewing angles. Addressing this limitation will require refining the model's ability to handle edge information more robustly.

Thin Objects

Our model struggles with accurately detecting and estimating distances to thin objects such as wires, thin beams, guardrails, and long poles. These objects pose a challenge due to their minimal pixel area, which makes it difficult to find correspondences necessary for depth prediction. The cost volume approach relies on significant overlap of objects for accurate predictions, and thin objects inherently lack sufficient overlap. Enhancing the model's capability to handle thin objects will require improvements in both feature extraction and depth estimation algorithms.

6.3 Future Work

Addressing these limitations will be crucial for the continued development and refinement of our omnidirectional stereo vision model. Future work should focus on:

- Developing rotation-invariant feature extraction techniques to improve robustness to object orientation.
- Creating configuration-agnostic evaluation methods to ensure consistent performance assessment across various camera setups.
- Refining prediction algorithms to minimize the occurrence of ghost points and enhance overall accuracy.
- Enhancing the model's ability to detect and estimate distances to thin objects through improved feature extraction and depth estimation methods.

These efforts will help advance the field of omnidirectional stereo vision and contribute to the development of more reliable and accurate models for real-world applications.

Part II

Tactical Reinforcement Learning

Chapter 7

Introduction

With the advent of deep learning, artificial intelligence agents have been able to achieve superhuman-level performance at playing classical strategic games such as chess, Go, and Starcraft [22, 29]. This development marks a step-change in an agent's ability to plan longer-horizon tasks to achieve a competitive goal against an intelligent opponent with potentially unseen strategies.

The overarching framework and family of learning algorithms used to train agents to this level of performance is called reinforcement learning (RL) [2]. Rather than using a dataset of inputs and ground truth outputs, RL relies on an environment, the resulting state changes that an environment goes through based on an agent's actions, and a reward function that rewards or penalizes agents based on the current state of the environment. Through this framework, an agent can learn based off of experience. Agents learn to maximize the reward based on a learned policy. With deep learning, neural networks became the approximator of this learned policy with great success [16, 22].

However, one of the overarching problems in RL is the credit assignment problem for long-horizon tasks [19]. When an agent is asked to learn a task, the learning goal of the agent is to find a sequence of actions that results in the highest reward. However, if an agent cannot determine which action(s) led to the high reward, it becomes hard to learn to perform the task well. The credit assignment problem can become a problem when an environment's reward function is sparse, or when a useful reward signal is very infrequently given to the agent. For example, take an agent that needs to move a block onto a dot on the floor. Reward sparcity would happen if an agent only receives reward when a block is placed in a specific location. When the block is not in that location, no useful feedback is given to the agent. However, if reward is given proportionally to the negative distance from the goal, the agent will gain useful feedback every episode and learn that the maximum reward can be achieved by placing the block on the dot.

In competitive strategic games, where the whole game is centered around achieving a single long-term objective, sparse rewards become a large problem. Agents have to learn to achieve a small subset of victory objectives, even at the cost of short-term victories or sacrifices. Teaching delayed gratification with no feedback on if that action is useful in the moment leads RL agents in strategic games to be strongly affected by the credit assignment problem. There might even be misleading secondary reward objectives that are still important such as victory while maintaining the highest total health. These secondary objectives may lead agents to becoming stuck in local minima, unwilling to sacrifice health for a high-reward objective that is never achieved because an agent is unwilling and unable to organize it's behavior long enough to reach it.

In this project, a complex competitive strategy game is developed with rule-sets of varying complexity. In all of the rule-sets, there are two teams with a set of units that can move and attack over units. Players win the game by eliminating all of the enemy's units. There are different types of units that have various movement budgets, health amounts, and attack values that naturally specialize the units to be better at different strategic roles. The goal of the project is to show that RL agents can win at this game and explore the capability of an agent to use tactics against a heuristic agent to win. Tactics are defined by this project as the agent having an obvious intention behind the decision to move units for the purpose of obtaining an advantage in the game. Lastly, this project also performs a hyperparameter study to show the effect of various hyperparameters on the performance of the Proximal Policy Optimization (PPO) algorithm and the policy network.

Chapter 8

Background

8.1 Reinforcement Learning

8.1.1 The Framework

Reinforcement Learning (RL) represents a unique branch of machine learning where an agent learns to make decisions through interactions with a dynamic environment. Unlike in supervised or unsupervised learning paradigms, reinforcement learning does not involve direct feedback on the agent's actions. Instead, learning is driven by rewards that provide signals about the effectiveness of the actions taken [2].

Defining the Components of RL

The reinforcement learning process involves several key components: an agent has a policy $\pi(a|s)$ that produces actions a based on the current state s; the environment is the world through which the agent interacts and from which it receives feedback; a reward r is the immediate feedback given by the environment in response to an action.

At each discrete time step t, the agent receives the current state s_t from the environment, selects an action a_t, and receives a reward r_t. The environment then transitions to a new state s_{t+1}, according to a transition function $T(s, a)$ that models the dynamics of the environment.

The Objective of Reinforcement Learning

The primary objective in reinforcement learning is for the agent to discover a policy that maximizes the cumulative reward it receives over time [2]. This goal is often formulated as maximizing the expected return, where the return, G_t, is defined as the sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

The discount factor γ, a value between 0 and 1, moderates the importance placed on future rewards relative to immediate rewards. A lower γ results in a myopic agent that prioritizes immediate rewards, whereas a higher γ encourages consideration of long-term benefits.

Markov Decision Processes and Partially Observable MDPs

Reinforcement learning problems are often modeled as Markov Decision Processes (MDPs), providing a mathematical framework for decision-making where outcomes are partly random and partly under the control of a decision-maker. The formal definition of an MDP is given by:

$$(S, A, T, R, \gamma)$$

where S is the set of all states, A is the set of all actions, T is the transition function present in the environment, R is the reward function, and γ is the discount factor.

In this framework, $T(s'|s, a)$ describes the probability of transitioning to state s' from state s given action a, and $R(s, a)$ assigns a scalar feedback signal to each action taken in a specific state. These components define the dynamics and the rewards of the environment, guiding the learning process of the agent [2].

However, in many real-world scenarios, the agent cannot fully observe the current state of the environment. These situations are modeled as Partially Observable Markov Decision Processes (POMDPs), which extend MDPs by incorporating observations that provide partial information about the state. In a POMDP, the decision-making process must account for uncertainty in state information, making it significantly more complex but also more applicable to realistic environments.

Given the framework, the next step is is to define an algorithm that updates a policy in response to the reward signal such that the policy learns the maximize the expected reward. The following sections introduce a specific family of policy update methods, called Policy Gradient Optimization, setting the stage for more detailed discussions on specific algorithms and their applications in complex environments.

8.1.2 Policy Gradient Methods

Policy Gradient Methods constitute a class of algorithms in reinforcement learning that optimize the policy directly [14]. This approach contrasts with value-based methods, which estimate a value function and derive a policy indirectly. Policy gradient methods adjust the policy parameters θ through gradient ascent, aiming to maximize the expected return, thus providing a direct method of policy optimization.

Introduction to Policy Gradients

The foundation of policy gradient methods lies in the concept of gradient ascent on the expected return. The policy, denoted by π_θ, where θ represents the parameters of the policy, is optimized to maximize the expected return from the start distribution. This expected return, denoted as $J(\theta)$, is defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

Here, $R(\tau)$ represents the total reward received over a trajectory τ, and the expectation is taken over the distribution of trajectories generated by the policy π_θ.

Deriving the Policy Gradient

To optimize the policy, we need to compute the gradient of $J(\theta)$ with respect to the policy parameters θ. This gradient, known as the policy gradient, is derived using the likelihood ratio trick combined with the law of the unconscious statistician. The resulting gradient expression is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)Q_t\right]$$

where Q_t is the *quality function*, representing the return from time t onward. This formula implies that the updates to the policy parameters are proportional to the product of the gradient of the policy's log-probability and the return. This relationship encourages the increase in the probability of actions that lead to higher returns.

Advantages of Policy Gradient Methods

Policy gradient methods are advantageous due to their direct approach to optimizing the policy, which can be particularly effective in environments with high-dimensional or continuous action spaces. They naturally integrate the exploration through stochastic policies, as the policy can suggest a range of actions with varying probabilities. Additionally, these methods are theoretically guaranteed to converge to a local maximum or a saddle point of the policy performance function [14]

Challenges in Policy Gradient Methods

Despite their advantages, policy gradient methods face several challenges. The gradient estimates can have high variance, which can lead to inefficient learning and high data requirements. These methods are typically less sample efficient compared to value-based methods and are often sensitive to the initial conditions and the choice of hyperparameters.

Variance Reduction Techniques

To address the issue of high variance in policy gradient estimates, several techniques can be employed. One common approach is to subtract a baseline from the returns, which reduces the variance of the gradient estimates without introducing bias. This technique involves modifying the gradient formula to:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}(\nabla_\theta \log \pi_\theta(a_t|s_t)(Q_t - b(s_t)))\right]$$

where $b(s_t)$ is a baseline value, typically chosen as the state value $V(s_t)$. When $b(s_t) = V(s_t)$, note that $(Q_t - V_t)$ is also called the *advantage function*, A_t. The advantage function represents the ”relative goodness” of an action as compared to other actions. This is often easier to learn or estimate as the absolute reward is not needed and only a relative value needs to be learned.

Another approach is the use of actor-critic methods, where the policy gradient is combined with value function approximation to stabilize the training process.

This section has elaborated on the theoretical underpinnings and practical aspects of policy gradient methods, setting the stage for the specific enhancements introduced by Proximal Policy Optimization in the subsequent sections.

8.1.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) represents a significant advancement in the family of policy gradient methods, developed to address specific inefficiencies and performance issues inherent in earlier methods [24]. This section traces the development from early policy gradient methods to PPO, highlighting its operational and strategic improvements over its predecessors.

Limitations of Early Policy Gradient Methods

Early policy gradient methods, while foundational in developing RL strategies that optimize policy directly, often suffered from high variance in gradient estimates and were sensitive to the step size used during learning. The performance of these algorithms could drastically vary with slight changes in policy, leading to potentially unstable training processes.

Introduction of Trust Region Policy Optimization

To mitigate the instability issues observed in early policy gradient methods, Trust Region Policy Optimization (TRPO) was introduced [23]. TRPO aims to make robust updates by limiting the changes to the policy at each step, thereby ensuring that the new policy is not too far from the old policy. It achieves this by optimizing the policy within a trust region, preventing large updates that could lead to performance collapse. The TRPO optimization problem is formulated as:

$$\max_\theta \mathbb{E}_{\tau\sim\pi_{\theta_{old}}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A^{\pi_{\theta_{old}}}(s_t,a_t)\right]$$

subject to

$$\mathbb{E}_{s\sim\pi_{\theta_{old}}}\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s)\parallel\pi_\theta(\cdot|s))\right]\leq\delta$$

However, TRPO's reliance on solving complex constrained optimization problems made it computationally intensive and difficult to scale.

Proximal Policy Optimization

PPO was developed as a response to the computational challenges of TRPO, simplifying the approach to policy optimization while retaining the core idea of trust regions [24]. PPO modifies the optimization strategy by using a clipped surrogate objective function, which avoids the need for solving the constrained optimization problem. The PPO objective is defined as:

$$L^{CLIP}(\theta)=\mathbb{E}_t\left[\min\left(r_t(\theta)A_t,\text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)A_t\right)\right]$$

Here, $r_t(\theta)=\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability ratio, and ϵ is a small hyperparameter that controls the extent of clipping, thus indirectly controlling the size of the policy update.

PPO offers several advantages over its predecessors, including TRPO. It simplifies the implementation by eliminating the need for complex second-order optimization methods, making it more computationally efficient and easier to tune. This efficiency does not come at the cost of performance, as PPO has been shown to perform competitively on a wide range of tasks, from playing games to controlling robots. Its ability to balance exploration and exploitation effectively, coupled with its robustness to hyperparameter settings, makes PPO a popular choice among researchers and practitioners in the field of reinforcement learning.

This section delineates the evolutionary path from earlier policy gradient methods to PPO, emphasizing the methodological enhancements that PPO brings to the field of reinforcement learning, particularly in terms of computational efficiency and training stability.

8.2 Strategic Games

Strategic games in the context of this thesis refer to those that require long-horizon planning and involve tactics aimed at achieving a broader, theatre-level objective. Such games often incorporate complex, multi-step processes where each action taken by a player can significantly influence future states and outcomes. The strategic nature of these games arises from the need for fine-grained control over the game state, where players must devise and execute long-term strategies rather than focusing solely on immediate rewards.

8.2.1 Characteristics of Strategic Games

In strategic games, players are often required to manage resources, plan their moves in phases, and anticipate the actions of their opponents. These games typically feature:

- Long-term planning and reward structures.
- Multiple phases within a single turn, allowing for a series of actions before the opponent can respond.
- A dynamic action space where available actions may change depending on the state of the game.

Such characteristics demand that players not only react to immediate opportunities but also plan for future advantages, making the gameplay both challenging and rich in strategic depth.

8.2.2 Multiphasic Games

Multiphasic games are characterized by having multiple phases within a single turn. During each phase, a player can execute a subset of actions, after which the opponent has a chance to respond with their own set of actions. This structure introduces a complex layer of strategic depth as decisions made in one phase can significantly impact the outcomes of subsequent phases.

Challenges in Multiphasic Games

One of the primary challenges in multiphasic games is the credit assignment problem. Since certain actions do not directly result in rewards but influence the outcomes of later phases, it becomes difficult to accurately assess the value of each action. This indirect reward structure requires sophisticated strategies and planning, as players must evaluate actions not only based on immediate outcomes but also on their potential future benefits.

Example and Analysis

A typical example of a multiphasic game could involve a strategy game where players first deploy units, then move them, and finally engage in combat. Actions taken during the deployment and movement phases might not yield immediate rewards but are crucial for positioning and gaining tactical advantages in the combat phase.

8.2.3 Variable Action Spaces

In many strategic games, the set of actions available to a player can change based on the state of the game. This variable action space introduces additional complexity, as the strategy must adapt to the evolving set of possibilities.

Challenges with Variable Action Spaces

Handling variable action spaces effectively is a significant challenge. It requires the game AI to be flexible and context-aware, capable of dynamically adjusting its strategy based on the available actions. This is particularly difficult when the action space not only changes size but also varies significantly in terms of the types of actions available.

Pointer Networks in Game AI

To address the challenges of variable action spaces, techniques like pointer networks are employed. Pointer networks are a type of neural network that can learn the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. In the context of games like AlphaStar, pointer networks allow the AI to select actions from a dynamically changing set based on the state of the game.

These sections outline the critical elements and challenges of strategic games that feature multiphasic turns and variable action spaces. Understanding these aspects is crucial for appreciating the advanced AI strategies discussed in the subsequent chapters of this thesis.

8.3 Prior Works

8.3.1 AlphaStar

Background, Overview, and Key Features

AlphaStar, developed by DeepMind, achieved Grandmaster status in StarCraft II, showcasing advanced reinforcement learning capabilities in dynamic, strategic environments [29]. The AI uses a multi-agent reinforcement learning framework, employing diverse strategies and managing complex state and action spaces. A notable feature is the use of pointer networks, which help in managing the variable action spaces by effectively mapping game state inputs to action outputs.

Reinforcement Learning Techniques and Strategic Implications

AlphaStar extensively uses policy gradient methods, particularly Proximal Policy Optimization (PPO), to optimize strategies within the game's intricate dynamics. It combines supervised learning from human game replays and reinforcement learning from agent-versus-agent games, mastering strategic interactions. The AI's gameplay involves critical decision-making in resource management and tactical combat under partial information, illustrating the potential of reinforcement learning in strategic applications.

Strategic Implications and Mechanics

AlphaStar's strategic gameplay involves managing economies, expanding territories, and conducting warfare with meticulously planned strategies that evolve in response to the opponent's tactics. The use of pointer networks enables the AI to select from a diverse set of possible actions dynamically, adapting to changes in the game state seamlessly. This strategic depth requires the AI to balance short-term tactical decisions with long-term strategic goals, a challenge that AlphaStar manages with notable proficiency.

8.3.2 CivRealms

Concept, Significance, and Features

CivRealms serves as a simulated platform to study AI behavior in multi-agent environments that simulate complex, real-world strategic interactions [20]. It challenges AIs with features like partial observability and non-deterministic outcomes, providing a rich environment for testing diverse strategic models. CivRealms supports experiments in cooperative and competitive strategies, offering a flexible rule system that mimics various strategic game dynamics.

Reinforcement Learning Applications and Strategic Dynamics

In CivRealms, AIs are tested against evolving game dynamics and player strategies, where pointer networks are employed to handle dynamically changing action spaces. The environment facilitates deep strategic planning, including alliance formation and conflict resolution, requiring AIs to adapt their strategies continuously, mirroring the strategic thinking needed in human interactions.

Strategic Implications and Mechanics

The gameplay in CivRealms requires a deep understanding of multi-agent dynamics and the ability to predict and counteract the strategies of other players. Strategic implications in CivRealms involve navigating complex scenarios such as resource management, diplomatic engagements, and tactical conflicts, all managed through the AI's capability to process and react to partial information about the game state and other players' actions. This complex interaction model tests the limits of current AI technology, particularly in understanding and manipulating open-ended strategic environments.

Chapter 9

Methodology

9.1 Game Rules and Environment

This section discusses the mechanics of the game environment and how these give rise to various tactics that reinforcement learning agents adopt. Certain mechanics within the game lead to more complex behaviors in reinforcement learning agents. For example, partial observability necessitates that agents learn to infer the positions of enemy agents. Mechanics such as flanking bonuses for adjacent units encourage reinforcement learning agents to organize units into formations, enhancing the effectiveness of encounters.

Another aspect considered is the impact of environmental complexity on learning. Simpler environments allow for the study of basic agent behaviors, while more complex rules and larger environments explore the effects on an agent's learning rate and capability [20].

A distinction is made between two types of worlds: a grid world and a hexagonal world. Each world have similarities and differences in their rulesets and served different purposes in the development of this project. The similarities of each environment are discussed below and the differences are discussed in each worlds' subsections.

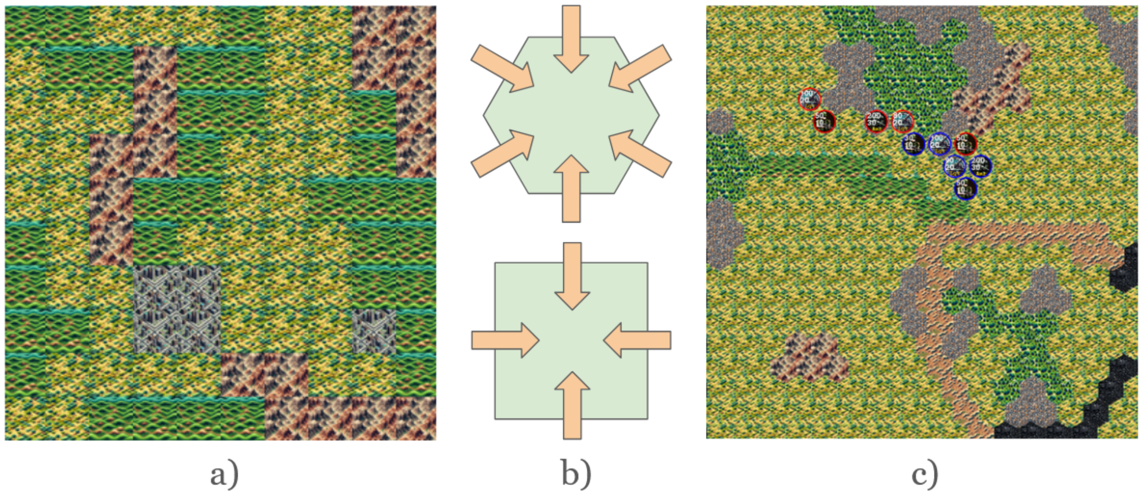

Figure 9.1: a) Example of Grid World Map, b) Diagram of Avenues-of-attack (AOA) between hex and grid map types, c) Example of Hex World Map

Units

In terms of unit types, there are knights, cavalry, and spearmen. Spearmen, although weaker with lower health and combat values, can still play a strategic role. Each team also has two cavalry units with greater movement, attack, and health capabilities but lesser than the single knight which boasts the highest combat effectiveness but limited movement.

Phases

The game phases include a 'reveal phase' where players can expose enemy units across the map, critical for initiating attacks during the subsequent 'combat phase'. The effectiveness of these phases in terms of strategic gameplay and learning outcomes forms a crucial part of the study.

9.1.1 Grid World and Simpler Rules

Both worlds differ in rules: the grid world operates under simpler rules without terrain-based bonuses or penalties, while the hexagonal world incorporates varied movement costs and terrain effects for three types of units: light, heavy, and motorized. These distinctions further complicate the learning environment and strategic possibilities.

In the grid world, agents can attack from four directions, which allows a maximum of four units to coordinate an attack on a single unit. Conversely, in the hexagonal world, the potential for flanking is increased, with six directions available for attack, allowing up to six units to simultaneously target a single unit.

The study also contrasts a small 5x5 grid world where enemy units are positioned on opposite sides, with a larger 20x20 grid world where enemies are clustered. This setup is used to investigate the effects of environmental size on strategic options and agent learning.

In the small grid world, the setup serves as a proof of concept to demonstrate that a reinforcement learning (RL) agent can consistently win against heuristic policy agents. The small, simplified environment facilitates easier learning and strategic execution by the RL agent.

In the larger grid world, agents are placed in clusters significantly far from each other, enhancing the complexity of strategic maneuvers and learning. This setup is intended to test the RL agents' ability to adapt to and navigate larger, more complex environments effectively.

9.2 Training Loop Overview

The policy was trained using Proximal Policy Optimization (PPO) and managed through StableBaselines3 and the PettingZoo multi-agent reinforcement learning environment library [21, 24, 27]]. The training loop was designed to end either when 1024 in-game turns, or 512 full turns, had been completed by the agent, or when victory conditions were met—these were defined as one team completely routing the enemy's units to zero health and thereby removing them from the game board.

The training process was executed over increments of 10 million steps to verify convergence, extending up to between 100 million and 500 million steps to explore the effects of long-term convergence. During training, the agents' units were vectorized across 32 CPUs to manage a rollout buffer size that varied across a range of parameters. This intensive setup facilitated the detailed study of training procedures and hyperparameter impacts on performance, which could potentially be discussed further in sections dedicated to experimental setups or hyperparameter studies.

Chapter 10

Experimental Procedure & Results

10.1 Hyperparameter Study

The hyperparameter study was conducted to systematically explore the impact of various hyperparameters on the performance of reinforcement learning agents within the grid world environment using Optuna, Stablebaselines3, and PettingZoo [1, 21, 27]. Over the course of 286 trials, different configurations were tested to optimize the model's ability to navigate and solve tasks effectively.

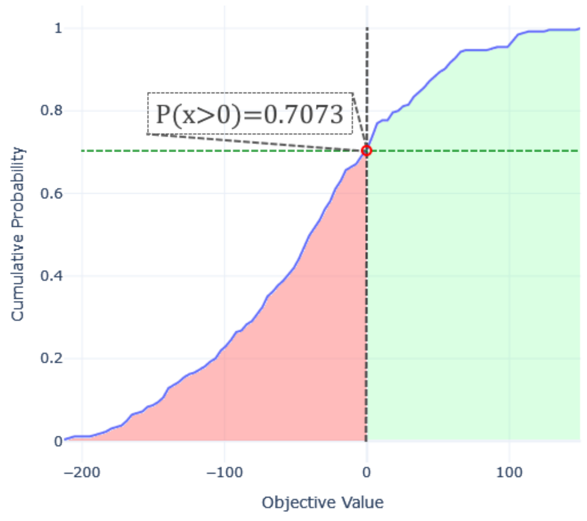

Figure 10.1: Cumulative Success Probability CDF. The probability of an agent winning across the entire hyperparameter study was 70.73%.

The primary objective of this study was to identify the optimal settings for key parameters, including the number of layers, layer sizes, learning rates, gamma values, batch sizes, and the number of steps per update. By varying these parameters, we aimed to understand their individual and collective effects on the model's training efficiency and overall performance.

The cumulative success probability CDF indicated that the agents had a 70.73% chance of achieving a positive outcome, demonstrating the robustness of the model under optimal conditions. This high success rate underscores the efficacy of the selected hyperparameter ranges and their tuning.

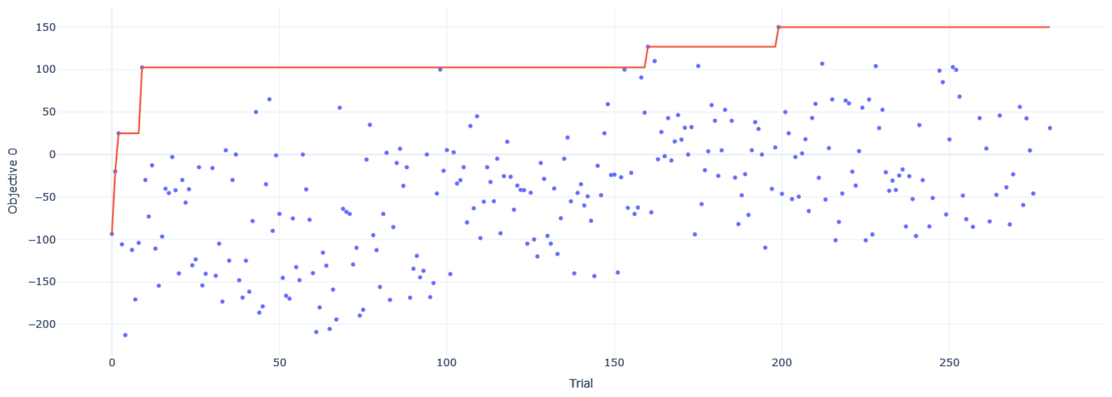

Figure 10.2: Hyperparameter Optimization Over Trials. 286 Trials were conducted.

Throughout the trials, it became evident that certain hyperparameters had a more pronounced impact on the learning outcomes. For instance, variations in the learning rate and batch size significantly influenced the stability and speed of convergence. Higher learning rates facilitated faster learning but sometimes led to instability, while lower rates provided more stable but slower improvements.

Table 10.1: Hyperparameter Choices for Tuning

Parameter	Choices
Layers	2, 3, 4
Layer Size	64, 128, 256
Learning Rate	1×10^{-4}, 3×10^{-4}, 1×10^{-3}
Gamma	0.9, 0.99, 0.999
Batch Size	64, 512, 2048
N Steps	64, 128

The table above summarizes the range of hyperparameters explored. Each parameter was selected based on its potential to influence the agent's learning process and performance. This systematic approach allowed for a detailed analysis of how different settings affected the agent's ability to learn and adapt within the grid world environment.

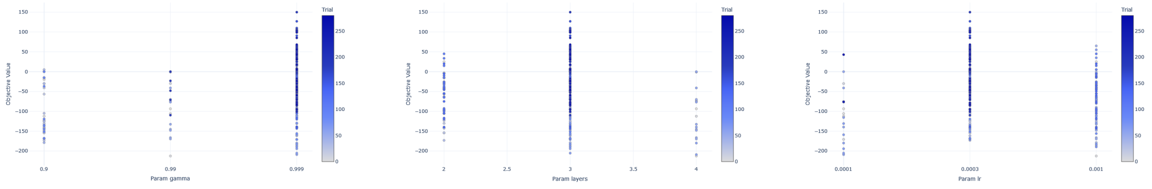

Figure 10.3: Left: Objective values for different batch sizes. Middle: Impact of layer sizes on performance. Right: Effectiveness of different numbers of steps.

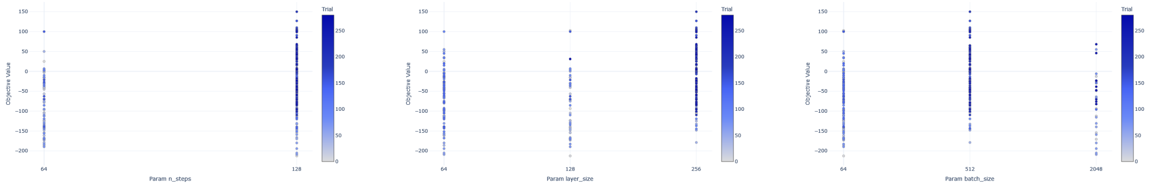

Figure 10.4: Left: Learning rate optimization results. Middle: Performance across different layer counts. Right: Gamma parameter adjustments and outcomes.

The results of the hyperparameter tuning provided valuable insights into how different configurations can enhance the agent's performance. For instance, larger batch sizes generally resulted in more stable learning, while smaller batch sizes sometimes led to faster but more volatile improvements. Similarly, the number of layers and layer sizes had a significant impact on the model's capacity to generalize and learn complex strategies.

The variation in gamma values highlighted the importance of the discount factor in shaping the agent's long-term strategy. Higher gamma values generally promoted long-term planning and more cautious strategies, while lower values encouraged more immediate, aggressive actions [19].

In summary, the hyperparameter study was crucial in refining the model's architecture and training process. By systematically exploring a wide range of parameter settings, we were able to identify configurations that significantly improved the agent's performance in the grid world environment. These findings not only enhanced our understanding of the model's learning dynamics but also provided a robust foundation for future research and application in more complex environments.

10.2 Training Results

The training process involved extensive hyperparameter tuning and performance evaluation to optimize the reinforcement learning agents within the grid world environment. This section details the key findings and insights drawn from the training results, highlighting the impact of different hyperparameter settings on model performance.

Overall Performance

The cumulative success probability, as illustrated in Figure 10.1, shows that the agents had a 70.73% chance of achieving a positive outcome across the entire hyperparameter study. This high success rate underscores the efficacy of the chosen hyperparameter ranges and their tuning, indicating that the agents were generally robust and adaptable to the grid world environment.

The hyperparameter optimization over 286 trials, shown in Figure 10.2, revealed significant variability in the performance outcomes based on different hyperparameter configurations. Notably, the models' performance varied greatly with changes in batch size, gamma, layer size, number of layers, learning rate, and number of steps.

Analysis of the hyperparameters revealed clear distinctions between well-performing and poorly-performing models. Consistency and balanced hyperparameter settings were crucial for optimal model performance, as demonstrated by the well-performing

Table 10.2: Common Hyperparameter Settings for Different Model Performances

Hyperparameter	**Best Models**	**Average Models**	**Poor Models**
Batch Size	512	64, 512	64, 2048
Gamma	0.999	0.999	0.990, 0.999
Layer Size	256	64, 256	64, 128
Number of Layers	3	2, 3	3, 4
Learning Rate	0.0003	0.0003, 0.0010	0.0001, 0.0010
Number of Steps	128	64, 128	128

models which maintained consistent settings across trials.

Insights from Training Dynamics

The training dynamics illustrated by the mean reward, health metrics, and win rates over time provide additional insights into the agents' learning process.

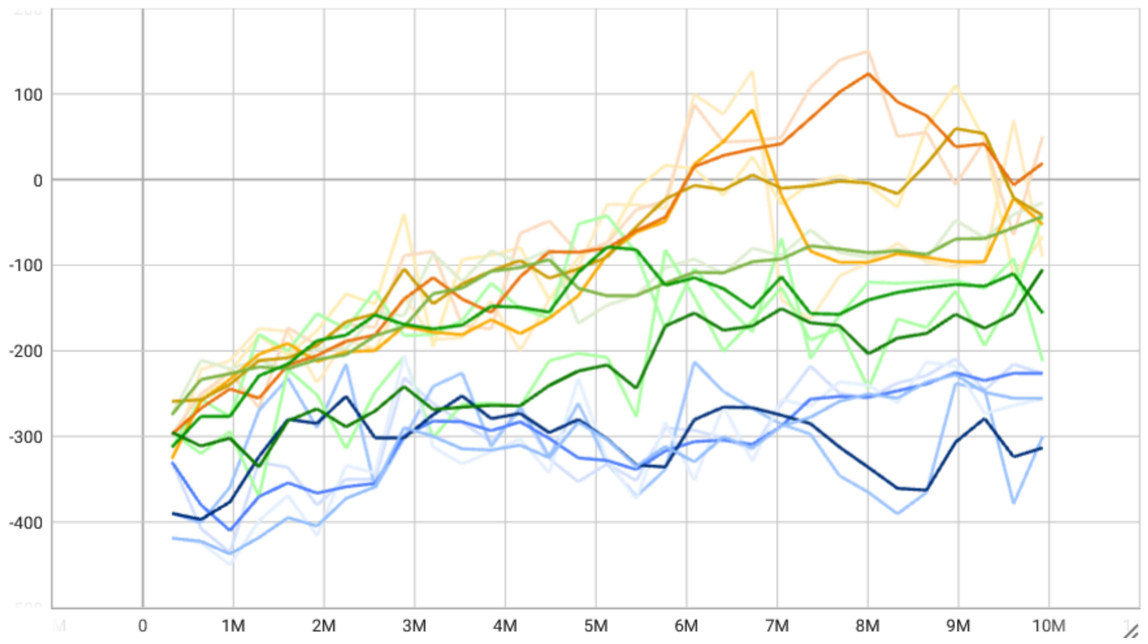

Figure 10.5: Y-axis: Mean Reward Value, X-axis: Training timesteps. Mean reward over time for different hyperparameter configurations. The yellow series represents the three best models, the blue series represents the three worst models, and the green series represents the three models closest to the average objective value.

Well-performing models consistently showed a rising trend in mean rewards, indicating effective learning and strategy optimization. In contrast, poorly-performing models exhibited flat or declining mean rewards, reflecting poor learning dynamics.

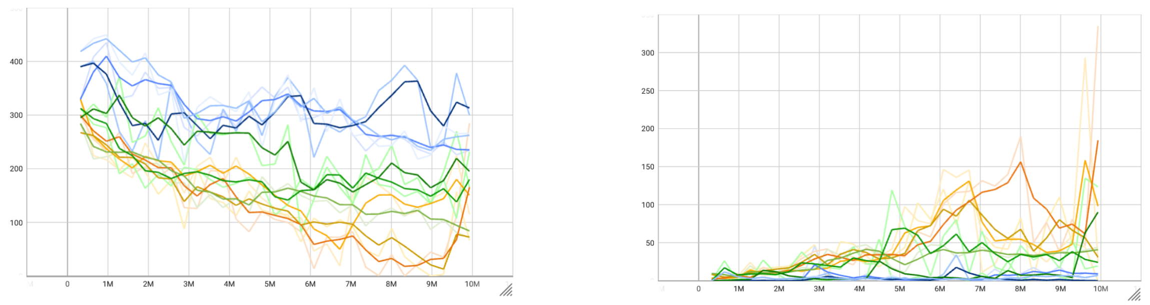

Figure 10.6: Left: Average health over training timesteps for the red team. Right: Average health over training timesteps for the blue team.

The health metrics for the red and blue teams highlight the resilience and strategic effectiveness of the agents. Well-performing models maintained higher and more stable health levels, particularly for the red team, indicating robust strategies and effective unit management.

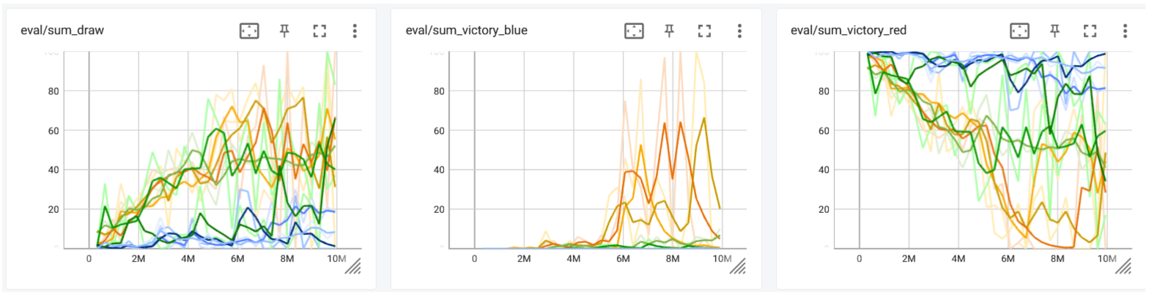

Figure 10.7: Mean win rates over training timesteps for different hyperparameter configurations. Agents are evaluated across 100 episodes. Left) Rate at which the agent draws, Center) Rate at which blue successfully wins the episode, Right) Rate at which the red heuristic agent wins the episode.

The win rate graphic further supports these findings, showing that well-performing models had higher and more consistent win rates compared to poorly-performing models.

10.3 Example of Learned Strategies

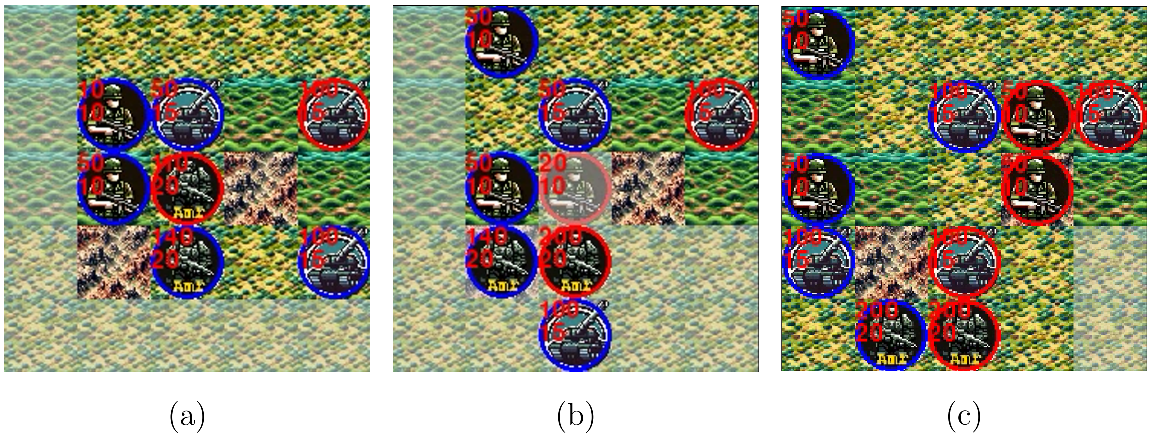

Figure 10.8: Examples of learned strategies: (a) Surrounding strategy where units surround the enemy from multiple directions, (b) Flanking strategy exploiting heuristic vulnerabilities, (c) Kiting strategy where units attack from a distance and then retreat to avoid retaliation.

This section provides a qualitative analysis of various tactics employed by the model to outmaneuver a heuristic policy. Demonstrated strategies include flanking, the exploitation of heuristic vulnerabilities, and a maneuver known as kiting. Additionally, the strategic exclusion of weaker units, such as spearmen, is employed to preserve them from direct confrontations.

These tactics seem to be effective in smaller environments to gain advantages after encounters end and also scale to larger ones.

In expanded spaces, units adopt hit-and-run tactics and exploit the heuristic policy's tendency to move towards the nearest enemy unit, effectively stalling them. This occasionally results in a draw, which, while sub-optimal, is recognized as an interesting adaptive behavior by the learned agent.

This adaptive strategy has prompted the integration of a reward term at the end of games, hypothesized to mitigate such stalling tactics and encourage more decisive outcomes. The use of visual aids in this subsection highlights the practical application and effectiveness of learned strategies in various environmental contexts.

Chapter 11

Conclusions

The hyperparameter study provided valuable insights into the impact of different settings on the performance and learning dynamics of reinforcement learning agents. Consistent and well-balanced hyperparameters, such as those used by the well-performing models, led to superior performance, effective strategy formulation, and robust learning. In contrast, high variability and extreme settings resulted in suboptimal performance, highlighting the importance of careful hyperparameter tuning in reinforcement learning.

The findings from this study will guide future research and application of reinforcement learning models in more complex environments, ensuring that optimal hyperparameter configurations are used to achieve the best possible outcomes.

11.1 Limitations

Overfitting to One Scenario

The models developed and trained in this study may have overfitted to the specific grid world scenario used during training. Although the hyperparameter tuning aimed to find robust configurations, the agents' performance in different scenarios or environments with varying rules and complexities was not evaluated.

Simple Rules, Small Strategy Space

The environments used in this study featured relatively simple rules and a small strategy space. While this allowed for effective learning and strategy development, it also limited the agents' ability to generalize to more complex environments with richer strategic possibilities.

11.2 Future Work

Better Architectures

Future research should explore more advanced neural network architectures that can better capture the complexity of strategic decision-making in varied environments. This could include deeper networks, attention mechanisms, or architectures designed specifically for sequential decision-making.

Large Language Model (LLM) Guided Play

Integrating large language models (LLMs) to guide reinforcement learning agents could enhance their strategic understanding and decision-making capabilities. LLMs can provide contextual insights and high-level strategies that complement the agents' learning from environment interactions.

More Complex Game Mechanics

Expanding the game mechanics to include more complex rules, varied terrain types, and additional unit types will provide a richer environment for training and evaluating reinforcement learning agents. This can help in understanding how agents adapt to and learn in more intricate strategic scenarios.

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL `https://arxiv.org/abs/1907.10902`. 10.1

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017. ISSN 1053-5888. doi: 10.1109/msp.2017.2743240. URL `http://dx.doi.org/10.1109/MSP.2017.2743240`. 7, 8.1.1, 8.1.1, 8.1.1

[3] Neill DF Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I 10*, pages 766–779. Springer, 2008. 3.2

[4] Zisong Chen, Chunyu Lin, Nie Lang, Kang Liao, and Yao Zhao. Unsupervised omnimvs: Efficient omnidirectional depth inference via establishing pseudo-stereo supervision. *arXiv preprint arXiv:2302.09922*, 2023. 3.3

[5] Bardienus P Duisterhof, Yaoyu Hu, Si Heng Teng, Michael Kaess, and Sebastian Scherer. Tartancalib: Iterative wide-angle lens calibration using adaptive subpixel refinement of apriltags, 2022. 4.1, 5.1.2

[6] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32 (8):1362–1376, 2009. 3.2

[7] Shiyu Gao, Zhaoxin Li, and Zhaoqi Wang. Cost volume pyramid network with multi-strategies range searching for multi-view stereo. In *Computer Graphics International Conference*, pages 157–169. Springer, 2022. 3.2

[8] Xiaodong Gu, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. Cascade cost volume for high-resolution multi-view stereo and stereo matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504, 2020. 3.2

[9] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. *Advances in neural information processing systems*, 30, 2017. 3.2

[10] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *International journal of computer vision*, 38:199–218, 2000. 3.2

[11] Maxime Lhuillier and Long Quan. A quasi-dense approach to surface reconstruction from uncalibrated images. *IEEE transactions on pattern analysis and machine intelligence*, 27(3):418–433, 2005. 3.2

[12] Jingliang Li, Zhengda Lu, Yiqun Wang, Ying Wang, and Jun Xiao. Ds-mvsnet: Unsupervised multi-view stereo via depth synthesis. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 5593–5601, 2022. 3.2

[13] Ming Li, Xueqian Jin, Xuejiao Hu, Jingzhao Dai, Sidan Du, and Yang Li. Mode: Multi-view omnidirectional depth estimation with 360° cameras. In *European Conference on Computer Vision (ECCV)*, October 2022. 4.4

[14] Jiacai Liu, Wenye Li, and Ke Wei. Elementary analysis of policy gradient methods, 2024. URL https://arxiv.org/abs/2404.03372. 8.1.2, 8.1.2

[15] Andreas Meuleman, Hyeonjoong Jang, Daniel S Jeon, and Min H Kim. Real-time sphere sweeping stereo from multiview fisheye images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11423–11432, 2021. (document), 2, 3.3, 4.5, 5.4, 5.2, 5.1, 5.2.1, 5.2

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602. 7

[17] Tyler Nuanes, Matt Elsey, Aswin Sankaranarayanan, and John Shen. Soft cross entropy loss and bottleneck tri-cost volume for efficient stereo depth prediction. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2840–2848, 2021. doi: 10.1109/CVPRW53098.2021.00319. 4.6

[18] Rui Peng, Rongjie Wang, Zhenyu Wang, Yawen Lai, and Ronggang Wang. Rethinking depth estimation for multi-view stereo: A unified representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8645–8654, 2022. 3.2

[19] Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, Olivier Pietquin, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning, 2024. URL https://arxiv.org/abs/2312.01072. 7, 10.1

[20] Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, Nian Liu,

Wei Wang, Yaodong Yang, and Song-Chun Zhu. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents, 2024. URL https://arxiv.org/abs/2401.10568. 8.3.2, 9.1

[21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22(1), jan 2021. ISSN 1532-4435. 9.2, 10.1

[22] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL https://doi.org/10.1038/s41586-020-03051-4. 7

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017. URL https://arxiv.org/abs/1502.05477. 8.1.3

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347. 8.1.3, 8.1.3, 9.2

[25] Steven M Seitz and Charles R Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35:151–173, 1999. 3.2

[26] Xiaojie Su, Shimin Liu, and Rui Li. Omnidirectional depth estimation with hierarchical deep network for multi-fisheye navigation systems. *IEEE Transactions on Intelligent Transportation Systems*, 2023. 3.3, 4.6

[27] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning, 2021. URL https://arxiv.org/abs/2009.14471. 9.2, 10.1

[28] Vladyslav Usenko, Nikolaus Demmel, and Daniel Cremers. The double sphere camera model. *CoRR*, abs/1807.08957, 2018. URL http://arxiv.org/abs/1807.08957. 4.1, 5.1.2

[29] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre,

Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, Nov 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL `https://doi.org/10.1038/s41586-019-1724-z`. 7, 8.3.1

[30] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. Tartanair: A dataset to push the limits of visual slam. 2020. 4.8

[31] Changhee Won, Jongbin Ryu, and Jongwoo Lim. Omnimvs: End-to-end learning for omnidirectional stereo matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8987–8996, 2019. (document), 2, 3.3, 4.2, 4.5, 4.6, 4.8, 5.2

[32] Changhee Won, Jongbin Ryu, and Jongwoo Lim. Sweepnet: Wide-baseline omnidirectional depth estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6073–6079. IEEE, 2019. 3.3, 4.5

[33] Changhee Won, Jongbin Ryu, and Jongwoo Lim. End-to-end learning for omnidirectional stereo matching with uncertainty prior. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3850–3862, 2020. 3.3, 4.5

[34] Sheng Xie, Daochuan Wang, and Yun-Hui Liu. Omnividar: Omnidirectional depth estimation from multi-fisheye images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21529–21538, 2023. 2, 3.3, 4.6

[35] Qingshan Xu and Wenbing Tao. Multi-scale geometric consistency guided multi-view stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5483–5492, 2019. 3.2

[36] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783, 2018. 3.2, 3.3, 4.2, 4.5, 5.2

[37] Yao Yao, Zixin Luo, Shiwei Li, Tianwei Shen, Tian Fang, and Long Quan. Recurrent mvsnet for high-resolution multi-view stereo depth inference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5525–5534, 2019. 3.2

[38] Hongwei Yi, Zizhuang Wei, Mingyu Ding, Runze Zhang, Yisong Chen, Guoping Wang, and Yu-Wing Tai. Pyramid multi-view stereo net with self-adaptive view aggregation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX 16*, pages 766–782.

Springer, 2020. 3.2