

Reinforcement Learning for Heliostat Control in Solar Tower Power Plants

Master's Thesis

for the degree of

Master of Science (M.Sc.)

Data Science

at the Faculty of Sciences of
Friedrich-Alexander-Universität Erlangen-Nürnberg

submitted on 30.09.2025

by **Safalya Pal**

Matrikel-Nr. 23193966

Supervisor: Prof. Dr. DhC. Enrique Zuazua
External Supervisor: Dr.-Ing. Max Pargmann

Acknowledgment

I would like to thank my supervisor Prof. Enrique Zuazua, for his support and the opportunity to pursue this topic at the Chair for Dynamics, Control, Machine Learning and Numerics at FAU. I am also grateful to Dr. Max Pargmann and the colleagues at the German Aerospace Center (DLR) for their support and the opportunity to work at the intersection of Reinforcement Learning and Solar Energy. And I would also like to immensely thank Ziqi Wang at the Chair for Dynamics, Control, Machine Learning and Numerics for his immense guidance. This work would not have been possible without their guidance, encouragement, and the many insights they generously shared.

Abstract

Solar thermal tower power plants present a promising solution for scalable renewable energy. These power plants concentrate sunlight onto a central receiver using a field of controllable mirrors called heliostats. However, due to the nature and scale of these fields, even slight misalignments or mechanical imperfections result in significant losses in absorbed power. Traditional open-loop control strategies require extensive calibration of each individual heliostat, which is time inefficient and can take anywhere between a few weeks and a few months, and drives up operational cost. Recent work demonstrates the ability of model-free RL methods to dynamically distribute heliostat aim-points on the receiver’s surface and achieve substantial improvements in terms of annual absorbed power. Despite the promising results, model-free RL methods suffer from high sample inefficiency and do not converge reliably. In this thesis, we address the more difficult task of directly controlling heliostat orientations. By leveraging analytical gradients from a differentiable simulator, our agents not only exhibit sample-efficient and reliable convergence but also outperform model-free RL methods and model predictive control.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Contributions	3
1.3	Thesis Structure	3
2	Theoretical Foundation	5
2.1	Concentrated Solar Power Plants	5
2.2	Heliostat Control	5
2.3	Differentiable Simulations	7
2.4	Alignment Deviation	7
2.5	Markov Decision Process	7
2.5.1	Partially Observable Markov Decision Processes	8
2.6	Reinforcement Learning	8
2.7	Machine Learning Training Techniques	9
2.7.1	Learning-Rate Scheduler	10
2.7.2	Weight-Decay Regularization	10
2.7.3	Dropout	10
2.7.4	Layer Normalization	10
2.8	Neural Networks	11
2.8.1	Multilayer Perceptrons	11
2.8.2	Transformer	11
2.8.3	Recurrent Neural Networks and LSTM	11
2.8.4	Residual Networks	11
3	Related Work	12
3.1	Differentiable Raytracing For Solar Tower Plants	12
3.2	Reinforcement Learning for Heliostat Aiming	12
3.3	Reinforcement Learning in Differentiable Simulations	13
4	Simulation Environment	15
4.1	Coordinate System	15
4.2	Scene Geometry	15

4.2.1	Mathematical Formulation of the Environment	15
4.2.2	Orientation Errors (in mrad) and Rotation	18
4.3	Flux density map Computation	19
4.3.1	Specular Reflection	19
4.3.2	Ray–Plane Intersection	19
4.3.3	Target Plane Grid	19
4.3.4	Distance–Dependent Gaussian Kernel (Per Ray)	20
5	Problem Formulation	21
5.1	Partially Observable Markov Decision Process (POMDP) for Heliostat Aiming	21
5.1.1	State space	21
5.1.2	Action Space	22
5.1.3	Observation Space	22
5.1.4	Transition Model	22
5.1.5	Observation Model	22
5.1.6	Reward Function	23
5.1.7	Reward Discount	23
5.2	Evaluation metric (not optimized during training).	23
6	Architecture and Model Training	25
6.1	Overview	25
6.2	Network Architecture	25
6.2.1	Sequence Input	25
6.2.2	Center-of-Mass (CoM) Block	26
6.2.3	Temporal Encoding	27
6.2.4	Final MLP Head and Action Normalization	27
6.2.5	Gradient Stop Operation	28
6.2.6	Fine-Adjustment Layer	29
6.3	Training Procedure	29
6.3.1	Analytical Policy Gradients	29
6.3.2	Sampling a Trajectory over Fixed Horizon	29
6.3.3	Backpropagation Through Time	31

6.3.4	Test-Time Adjustment Layer (Fine Adjustment)	32
6.4	Summary of Design Choices	34
6.5	Assumptions	34
7	Results	35
7.1	Experimental Setup	35
7.1.1	Training and Test Sun Directions	36
7.2	Comparison against Baseline	36
7.3	Flux Density Shift over Steps	38
7.4	Effects of Varying Parameters	39
7.4.1	Effect of Heliostat Distance	39
7.4.2	Effect of Induced Error Scale	40
7.4.3	Effect of Fine Adjustment	40
7.4.4	Effect of Fine-Adjustment Step Count	42
7.4.5	Effect of Training Horizon	43
7.4.6	Effect of Sequence Length (k past flux density maps)	44
7.4.7	Effect of Model Architecture	45
8	Discussion	47
8.1	Robust closed-loop control	47
8.2	Role of the fine-adjustment layer	47
8.3	Horizon and temporal context	47
8.4	Choice of Architecture	48
8.5	Distance and induced-error sweeps	48
8.6	Implications for CSP operations	48
8.6.1	Latency and compute budget	48
8.6.2	Shifting from per-heliostat sensors to aggregate signals	48
8.7	Why our pipeline works	48
8.8	Limitations	49
8.8.1	Single-heliostat model	49
8.8.2	Simplified optics and error model	49
8.8.3	Episode stationarity	49
8.8.4	Metric floor	49

8.9	Design recommendations	50
9	Conclusion	51
10	Outlook	53
10.1	Changing Sun position over time	53
10.2	Multiple Heliostat Control	53
10.3	Fine Tuning in a higher fidelity raytracing simulations	53
	Bibliography	57
A	Appendix	58
A.1	Derivative of h_t w.r.t. θ_π (additive carry)	58
A.2	ReLU activation function	58
A.3	Dirac delta distribution	58
A.4	Policy Network	59

List of Figures

1	Over 2000 mirrors concentrate incoming sunlight onto the tips of the solar tower at Juelich, Germany. Credit: DLR, CC BY-NC-ND 3.0.	1
2	Closed Loop Control for Solar Tower. A sensor measures the alignment error and passes it onto a controller which corrects for it and updates the real-world object	6
3	Open Loop Control for Solar Tower. Alignment error are measured and and passed onto an optimizer which creates and adapted alignment model. This model is then used to predict alignment errors and correct them.	6
4	Overview of the differentiable heliostat simulation, depicting the geometric model (error-perturbed normals) and the optical model (ray reflection, receiver intersection, and flux density map computation).	15
5	Global coordinate system of the differentiable heliostat environment, showing the East (X), North (Y), and Up (Z) axes	16
6	Sun-direction parameterization. The azimuth ϕ (red) and elevation θ define the central axis \mathbf{a}	17
7	Left: Desired heliostat normal. Right: Heliostat normal after inducing alignment error on an axis	18

8	Reinforcement-learning formulation of the heliostat-control task. The agent observes the flux-density images and auxiliary features, selects mirror-normal actions, and receives rewards based on the environment’s reward function.	21
9	The policy architecture of our agent. Consisting of a Center of Mass Block, a transformer encoder block (optionally replaceable with LSTM / MLP) a final MLP head and a fine-adjustment layer	25
10	The unrolled trajectory graph for our Analytical Policy Gradient setup. The initial o_0 is obtained by taking the ideal normal as the action and passing it to our environment f	30
11	Test-time adjustment layer: a small persistent offset ϵ^{fine} is optimized online via a few gradient steps, first using a distance loss and then MSE, with normalization after each update.	33
12	Sun direction sampling strategy. Training positions (black) are placed on a structured azimuth–elevation grid. The test cases are highlighted in red: the average of the training distribution (Interpolation) and an out-of-distribution shifted position (Extrapolation).	37
13	Comparison for MSE over time-steps for our APG framework v/s Model Predictive Control. For MPC, we set the initial guess for the action to be our ideal normal vector and optimize henceforth. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position	37
14	Center of mass for 60 instances rolled out over 30 steps and plotted at every 3 steps. All instances have induced errors sampled from a uniform $[0, 25]$ mrad distribution	38
15	Average test alignment error over training episodes, for heliostate distance = 15m, 150m and 1500m. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position	40
16	Average test alignment error over training episodes, for a max induced alignment error of 5mrad, 10mrad, 25mrad, and 45mrad: Performance over extrapolated sun position, Right: Performance over interpolated sun position. The performances after inducing a max of 5mrads and 10mrads remain mostly similar	41
17	Average test alignment error over training episodes. This plot shows performance over three settings, one where fine alignment is always enabled, one where it is always disabled, and one when it’s only enabled during test time. Left Performance over extrapolated sun position, Right: Performance over interpolated sun position	42

18	Average test alignment error over training episodes. This plot shows performance over three settings for our training’s horizon-length, namely $T=5$, $T=10$ and $T=15$. Left Performance over extrapolated sun position, Right: Performance over interpolated sun position	43
19	Tradeoff Plot between average min-error and average time to best loss, evaluated over all T . The plots cross over at T	44
20	Average test alignment error over training episodes, for Transformers, LSTMs and MLP backbones. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position	46

List of Tables

1	Nominal training sun positions (azimuth and elevation angles).	36
2	Final pointing error (mrad) for unseen sun directions.[lower is better]	38
3	Average minimum angular error (mrad) at different heliostat distances for extrapolated and interpolated sun positions. Values are averaged over multiple runs.[lower is better]	39
4	Achieved minimum angular error (mrad) with different fine-adjustment settings.[lower is better]	41
5	Average relative time to best loss (min) with different fine-adjustment settings.	41
6	Minimum angular error (mrad) with different numbers of fine-adjustment steps.[lower is better]	42
7	Average relative time to best loss (minutes) with different numbers of fine-adjustment steps.	42
8	Minimum angular error (mrad) [lower is better] with different training horizons.	43
9	Average relative time to best loss (minutes) with different training horizons.	44
10	Minimum angular error (mrad) vs. sequence length k . [lower is better]	45
11	Average relative time to best loss (minutes) vs. k	45
12	Minimum angular error (mrad) by architecture. [lower is better]	45

1 Introduction

The decarbonization of global energy systems requires low-cost renewable energy generation and sources of stability that ensure dispatchable power. Rapid cost reductions and large-scale deployment have been achieved by solar photovoltaics and wind power; however, their intermittency poses challenges for maintaining a reliable supply [6, 4, 20]. This gap is addressed by Concentrated Solar Power (CSP), which concentrates sunlight into high-temperature thermal energy that can be stored cost-effectively and converted to electricity on demand [5, 16, 19, 12, 18]. A major advantage of CSP is its compatibility with thermal energy storage (e.g., molten salt), enabling dispatchable power generation during periods without sunlight, supporting grid stability and energy-demand matching.



Figure 1: Over 2000 mirrors concentrate incoming sunlight onto the tips of the solar tower at Juelich, Germany. Credit: DLR, CC BY-NC-ND 3.0.

In this study, we focus on Solar Tower CSPs (Power Towers). A solar power tower (or central receiver system) consists of a field of heliostats (mirrors) that track the sun and reflect sunlight toward a central receiver located on top of a tall tower. However, in solar tower plants, heliostats must continuously reflect sunlight onto a central receiver. This results in a major problem in heliostat control since even minute tracking errors or structural misalignments cause flux to spill outside the receiver, resulting in efficiency losses, increased stress on the receiver, and in worst cases, safety risk. Current approaches to mitigate this issue requires heliostats to be calibrated with high accuracy. However, current calibration procedures are time-consuming, where adjusting a single heliostat can take about an hour, and considering the fact that this can only be done when the sun is out, calibrating an entire field of thousands of heliostats may span several weeks or months. Figure.1 shows one such experimental tower located at Juelich, Germany.

This traditional approach is inherently open-loop[17, 24], i.e., heliostats are calibrated based on static measurements under sunlight and the corrections are assumed to re-

main valid over time, without continuous feedback. While a closed-loop system[24] for heliostats can significantly reduce the time needed for calibration and improve tracking accuracy, such methods need sophisticated sensors or camera on each heliostat, which drive up operational costs significantly.

1.1 Motivation

Recent advancements in Reinforcement Learning (RL)[28] have demonstrated promising results in the area of realtime closed-loop control across a wide range of domains, such as robotics, autonomous vehicles, process optimization, etc. RL agents can map observed states directly to corrective actions, which enables them to adapt continuously to changing system dynamics and external noise. This eliminates the reliance on static, one-time calibrations and instead provides a robust, feedback-driven approach to heliostat control.

In the context of heliostat fields, RL can bring similar improvements. Once an RL policy has been trained, it can continuously adjust heliostat orientations in real time based on the observed flux density at the receiver, fulfilling the role of an effective closed-loop controller. Unlike traditional closed-loop control requiring per-heliostat sensors, RL policies can operate on aggregate signals such as receiver-images or simulated heatmaps, reducing hardware cost while providing a method for dynamic correction.

However, while deep learning in the context of heliostat calibration such as image-based misalignment detection, geometric error modeling, or heliostat surface deformity modelling, has been explored quite increasingly, very little attention has been given to Reinforcement Learning in the context of heliostat control for tower CSPs. To the best of our knowledge, the only closely related work is by Carballo et al. [2], who apply a deep RL (Soft Actor Critic) approach to optimally distribute aiming points on the receiver surface to maximize absorbed power. However, this approach does not directly control heliostat orientations, but rather, it operates at the level of the receiver's flux distribution. RL for direct heliostat-level control, enabling continuous real-time correction of tracking errors and misalignments, thus remains largely unexplored.

It is also important to note that model-free RL methods, while effective, are often sample inefficient and suffer from unstable convergence. Training such agents usually requires a large number of interactions with the environment. Additionally, convergence is not always guaranteed and agents can become stuck in suboptimal policies. To address these limitations, recent research such as Analytic Policy Gradient[31], Short Horizon Actor Critic [34], and Schnell and Thuerey (2024) [25], leverage differentiable simulators to train RL policies more efficiently.

In the context of solar tower CSPs, Pargmann et al. [21] introduced a differentiable ray-tracing framework that models heliostat-receiver interactions in a physically accurate and differentiable manner. By integrating differentiable simulation with RL, it becomes possible to improve sample efficiency, stabilize convergence, and providing a framework for scaling to real-world heliostat fields where interaction data is limited

and costly to obtain. While this simulator is highly accurate and highly parallelizable, it remains feature rich and computationally expensive when executed over many training iterations. In practice, this implies that faster training can be achieved by employing a simplified, less accurate surrogate simulation that trade physical detail for computational efficiency.

1.2 Contributions

The overarching goal of this thesis is to advance the control of heliostat fields in solar tower power plants by leveraging reinforcement learning (RL) within a fully differentiable simulation framework. To achieve this, the two main contribution includes:

1. **Differentiable Simulation Environment for Heliostat Optics:** A fast, simplified and highly parallelizable simulator that models the physics of heliostat-tower interactions, and is computationally more efficient than full-scale ray-tracing models. This environment serves as a groundwork for pre-training closed-loop control policies, to be deployed to a real heliostat field.
2. **Model-based Reinforcement Learning Framework:** A novel RL framework that directly exploits the differentiable simulator to improve sample efficiency and performance during training.

1.3 Thesis Structure

The remainder of this thesis is organized as follows: In Chapter 2, we describe the theoretical foundations necessary to understanding our framework. In Chapter 3, we talk about the background and past works relevant to the context of our framework, relating to CSPs, RL, and Differentiable Simulations. In Chapter 4, we define the structure of our end-to-end differentiable simulation for heliostat optics and describe how it performs relevant computations. In Chapter 5, we mathematically formulate our heliostat control task as a Partially Observable Markov Decision Process. In Chapter 6, we talk about our agent’s neural architecture and describe our learning algorithm (APG). In Chapter 7, we define our experiment setup and provide results for the same. In Chapter 8, we summarize the empirical findings with respect to the objectives of our thesis. In Chapter 9 we summarize the contributions of our thesis. In Chapter 10, we talk about the still open questions and possible future works.

2 Theoretical Foundation

This research lies at the intersection of scientific areas of Concentrated Solar Power Plants, Control Theory, and Deep Reinforcement Learning. Hence, the theoretical foundation of these topics, relevant to this thesis, is summarized within the next section. Additionally, as the above mentioned areas follow different approaches and have different terminologies, a unified notation and terminology are adopted throughout the thesis to avoid any ambiguity.

2.1 Concentrated Solar Power Plants

Concentrated Solar Power (CSP) systems can be broadly classified into *Parabolic Trough* and *Solar-Tower* systems. A Solar Tower plant collects solar radiation and focuses that onto a central receiver area, generating zones of intense energy density (known as *flux*), that is much higher than its surrounding irradiance. The captured thermal energy can then be routed to a reactor as heat, where it can either be directly converted into electricity, stored in a thermal reservoir, or supply high-temperature industrial process heat.

A key element of a Solar Tower plant is its collector, which in the context of this thesis is the **heliostat**. Each heliostat is a set of mirror facets, called concentrator, mounted on a sun-tracking mechanism that reflects and concentrates the solar radiation along the direction of the receiver's surface. Multiple heliostats together form a *heliostat field*. The heliostat field is required to deliver a uniform, well-regulated flux profile at the receiver, to minimize efficiency losses, reduce overheating and avoiding structural damages.

2.2 Heliostat Control

Improving the Solar Tower's efficiency relies on precise heliostat aiming, which is managed by a Concentrated Solar Power Operating System (CSP-OS), which is the industry standard for heliostat control. CSP-OS uses ray-tracing to model each of the heliostat's expected reflection, and coordinates the heliostat field for optimal flux delivery. For each heliostat, a digital twin is maintained, which is calibrated against past field performance data.

The CSP-OS platform implements two control strategies: Open-Loop and Closed-Loop heliostat control. Closed-Loop systems (Figure. 2) continuously update the actuator positions of the heliostat based on feedback from sensors about deviations from the desired flux density distribution for each heliostat. An Open-Loop system (Figure. 3) on the other hand updates the actuator positions solely based on calculations performed on the heliostat's digital twin, calibrated using past data. Our proposed framework focuses on solutions for Closed-Loop systems.

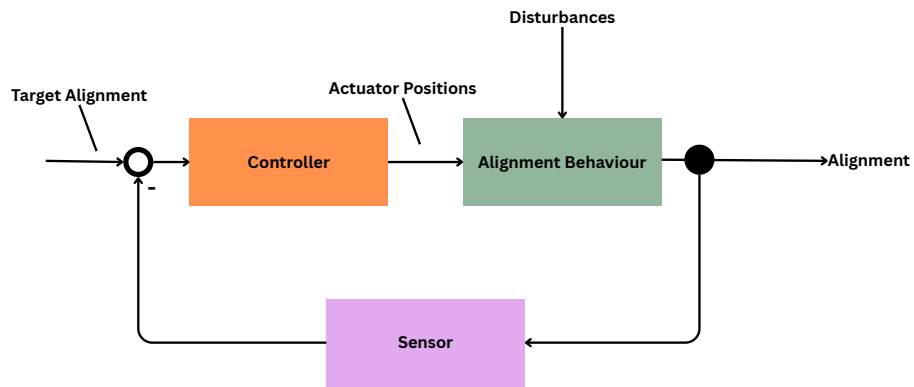


Figure 2: Closed Loop Control for Solar Tower. A sensor measures the alignment error and passes it onto a controller which corrects for it and updates the real-world object

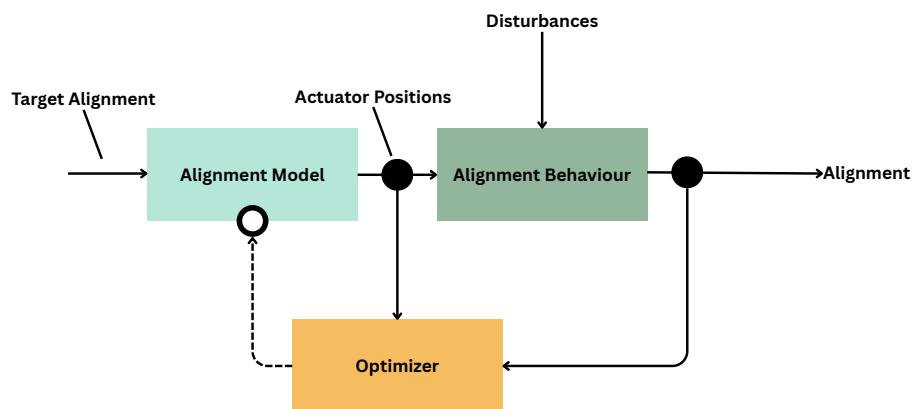


Figure 3: Open Loop Control for Solar Tower. Alignment error are measured and passed onto an optimizer which creates and adapted alignment model. This model is then used to predict alignment errors and correct them.

2.3 Differentiable Simulations

In general, a simulation which approximate the behaviour of physical system over time, which enables controlled experimentation without directly relying on the real-world system. Widely popular due to *Degrave et al. (2016, NeurIPS)*, differentiable simulators are simulators which are end-to-end differentiable, i.e., they're capable of calculating gradients throughout the entire duration of the simulation, for a desired objective function, with respect to any desired parameter. Hence, they provide a direct way to be integrated with deep-learning frameworks and gradient based optimization methods.

2.4 Alignment Deviation

In the heliostat calibration literature, *pointing error* (or *alignment deviation*) is typically defined as the angular difference between a heliostat's alignment model prediction and its actual physical orientation. In our simulation environment, however, heliostats are modeled without such intrinsic deviations, which creates a challenge when training control policies intended to operate under real-world conditions. To address this, we explicitly introduce orientation errors by sampling error angles from a prescribed probability distribution.

These errors are modeled as small rotations about the East and Up axes. The corresponding angular deviations are parameterized in milliradians (mrad), where

$$1 \text{ mrad} = 10^{-3} \text{ rad} \approx 0.0573^\circ.$$

The effects of alignment deviations become significantly greater with increase in distance from the receiver. For large heliostat fields, commonly exceeding a receiver-heliostat distance of 1000m, even the smallest alignment deviation might lead to heliostats potentially missing the receiver [15]. Heliostat calibration literature suggests that in a commercial setting, a tracking error of $\leq 1\text{mrad}$ [14, 11] should typically be aimed for.

2.5 Markov Decision Process

A *Markov Decision Process* (MDP) provides the mathematical framework for sequential decision making under full observability. An MDP is defined as the 5-tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma),$$

where \mathcal{S} is the set of all possible environment states, \mathcal{A} is the set of actions available to the agent, $P(s' | s, a)$ is the transition kernel giving the probability of going from state s to s' after taking action a , $R(s, a)$ is the expected immediate reward obtained by taking action a in state s , $\gamma \in [0, 1]$ is the discount factor that weights future rewards.

The Markov property requires that the next state and reward depend only on the current state–action pair, not on the states before it.

The agent tries to find an optimal policy $\pi^*(a | s)$ that maximizes the expected return

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_k \right].$$

2.5.1 Partially Observable Markov Decision Processes

In many real-world dynamic systems the true state of the environment cannot be observed directly. A *Partially Observable Markov Decision Process* (POMDP) is an extension to MDP which handle this uncertainty. A POMDP is defined as the 7-tuple

$$\mathcal{M}_{\text{POMDP}} = (\mathcal{S}, \mathcal{A}, \Omega, \mathcal{O}, P, R, \gamma),$$

where the components \mathcal{S} , \mathcal{A} , P , R , and γ are as above, and the additional elements are

- \mathcal{O} , the observation space containing all possible observations available to the agent,
- $\Omega(o | s, a)$, the observation model giving the probability of receiving observation o after taking action a and arriving in state s .

Because the agent receives only observations o_k , it maintains a *belief state* $b_k(s)$, a probability distribution over \mathcal{S} that summarizes the history of actions and observations. A POMDP policy $\pi(a | b)$ maps belief states to actions so as to maximize the expected discounted return,

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_k \right].$$

These generic definitions provide the foundation for the heliostat-aiming formulation presented in the following section.

2.6 Reinforcement Learning

First introduced by by Sutton and Barto, Reinforcement Learning (RL)[28] provides a computational framework to learn from interaction, in which an *agent* learns to achieve a goal by trial and error while receiving evaluative feedback from the environment. This area is inspired by the way animals and humans learn from *rewards and punishment*. A standard reinforcement-learning framework is defined in terms of the following interaction elements:

- **Agent** - The decision-making entity which selects actions in order to maximize the long-term reward. In our work, the agent's objective is to learn to control the heliostat orientation.
- **Environment** - Everything external to the agent which it interacts with. The environment changes with respect to the agent's actions and provides feedback in form of observation and rewards. In our framework, the environment is the differentiable heliostat optics simulation.
- **Action** (a_k) - The control signal chosen by the agent at the time step k (which in our framework, is discrete). A policy π defines the method which the agent uses to choose its actions.
- **Reward** (r_k) - An evaluation metric returned by the environment after each action. This indicates the immediate quality of the agent's choice. The agent tries to maximize the cumulative reward over time.
- **Observation** (o_k) - The information which the agent actually receives from the environment, based on which it must act. Note that in a general RL framework, this is called "state" instead. However, in our POMDP formulation, the state refers to the latent state of the environment.

The agent is not told which action to take, instead it must discover, through repeated interaction with its environment, which actions yield the greatest long-term (cumulative) return.

RL algorithms can be classified into one of two categories: Model-Free RL and Model-Based RL. **Model-free** methods don't learn or utilize a transition model of the environment. The agent improves its policy solely based on the observations and the rewards received. These approaches typically require a large amount of replay data to achieve good performance. On the other hand, **Model-Based RL** methods explicitly learn or have access to a model of the environment's dynamics (e.g., transition function and reward model). The model can then be used for planning or performing look-ahead search to evaluate future rewards. Model-Based RL methods typically achieve higher sample efficiency compared to Model-Free methods, i.e., they require less amount of data to achieve a comparable level of performance. For our framework, we focus on implementing a Model-Based RL method for heliostat control.

2.7 Machine Learning Training Techniques

Other than our RL agent's architecture and optimizer, there exist several techniques in Machine Learning literature to optimize a model's training capacity and generalization performance by adjusting its hyperparameters. Hyperparameters are values which control the model training behaviour, which are defined outside the training loop. Such techniques relevant to our framework are summarized below.

2.7.1 Learning-Rate Scheduler

The learning rate controls the step size of each parameter update for our neural network. A *scheduler* automatically adjusts this rate during training. Typically, a scheduler starts with a larger value for the learning rate to encourage rapid progress and gradually reducing it to allow fine convergence. In our experiments we use a **Reduce-on-Plateau** scheduler, which monitors a validation metric and lowers the learning rate whenever the metric fails to improve for a predefined number of epochs. This adaptive scheme helps avoid stagnation and often leads to better final performance compared with a fixed learning rate schedule.

2.7.2 Weight-Decay Regularization

In machine learning, the complexity of the patterns to be learned, remains generally unknown. Hence, the model's complexity should be at least as high as to match the pattern's complexity. However, a model much more complex than required can lead to overfitting, where the model starts to "memorize" the training data and when evaluated on the training, can predict correctly. However when evaluated on data point unseen during training, the predictions are mostly incorrect.

Weight decay penalizes a model's complexity by adding an ℓ_2 penalty to the loss function, discouraging large parameter magnitudes and thus reducing overfitting. It effectively pulls the weights toward zero during optimization, leading to smoother, more generalizable models.

2.7.3 Dropout

Dropout randomly "disables" a subset of activations during each forward pass, preventing units from over-reliance on any particular activation. At inference time all units are used, with their outputs scaled to account for the dropped connections. This regularization is simple but highly effective at improving generalization.

2.7.4 Layer Normalization

Layer normalization standardizes the activations within a layer by subtracting their mean and dividing by their standard deviation. It normalizes across features *within* a single training example, making it well suited to for recurrent networks and small batch sizes. This stabilizes gradients and often accelerates convergence. It is also a standard practice for training modern *Transformer* architectures, where layer normalization is applied before or after each attention and feed-forward block to maintain stability during training.

2.8 Neural Networks

Modern reinforcement-learning agents frequently use deep neural networks as function approximators. Below we briefly summarize several key architectures that are relevant to our work. A complete technical description of Neural Networks is beyond the scope of this thesis, hence the only the topics in the context of Neural Networks which are relevant to our framework is described below.

2.8.1 Multilayer Perceptrons

A *Multilayer Perceptron* (MLP) [23] or a feed-forward network composed of stacked layers of affine transformations followed by non-linear activation functions. MLPs are universal function approximators and form the basic building block for Neural Network architectures.

2.8.2 Transformer

The *Transformer* [29] architecture substitutes recurrent computation with self-attention mechanisms, allowing each element of a sequence to attend to all others in parallel. This design enables efficient modeling of long-range dependencies and is the foundation of most current state-of-the-art sequence models, including large language models.

2.8.3 Recurrent Neural Networks and LSTM

Recurrent Neural Networks (RNNs) process sequences by maintaining a hidden state that is updated at each time step. They are well suited to tasks where temporal context is important. The *Long Short-Term Memory* (LSTM) [8] variant introduces gating mechanisms (input, forget, and output gates) to alleviate vanishing- and exploding-gradient problems, enabling the network to capture long-term dependencies more reliably.

2.8.4 Residual Networks

Residual Networks (ResNets) [7] incorporate *skip connections* that add the input of a layer to its output. These identity shortcuts ease the training of very deep networks by mitigating gradient degradation, allowing stable optimization of hundreds or even thousands of layers.

While this section only provides a high-level overview, these architectures serve as fundamental building blocks for the models later described in our reinforcement learning framework.

3 Related Work

Due to the popularity of Reinforcement Learning, and the high impact of CSPs as a source of renewable energy, multiple advancements relating to heliostat modelling, differentiable simulators, and reinforcement learning in differentiable simulators have been published. Relevant to our work are three aspects of these publications. The first aspect being designing a fast, end-to-end differentiable optics simulation which models the interactions between the sun, heliostats and the receiver. The second aspect being comparison to RL baselines heliostat aiming. While no work in literature addresses the task of directly controlling the heliostat orientation directly via RL, we talk about the next most similar task of selecting aim-points on the receiver surface. The third aspect being algorithms leveraging differentiable simulations, which act as a basis for this work’s methodology.

3.1 Differentiable Raytracing For Solar Tower Plants

Pargmann et al. (2023) [21], developed an end-to-end differentiable raytracer for applications within solar towers. This raytracer is designed using PyTorch and allows automatic differentiation. Automatic differentiation is a method that builds a gradient graph by adding each node’s gradient. The chain rule can be used to extract the gradient for a node’s children using the gradient graph. Each equation is broken down into its most basic calculations, which are then added to the gradient graph. This allows for a gradient of a chosen loss function, with respect to some other parameters in the graph, to be passed along the graph. Hence allowing us to update such parameters with gradient based optimizers such as gradient decent, or Adam.

The nature of this ray-tracer, allows us to calculate the gradients of the flux-density at the receiver, and the alignment error, with respect to our heliostat actuators’ positions.

3.2 Reinforcement Learning for Heliostat Aiming

Carballo et al. (2025)[2] Proposes using a model-free deep RL algorithm called Soft Actor–Critic (SAC) to optimizing the aim-points for heliostats over the receiver remains to maximize energy capture while ensuring operational safety.

However, distributing aim-points on a planar surface is a much easier task for an RL agent compared to controlling heliostat orientations, since in the former task, the agent does not need to account for the sun’s position, the distance and orientation of the receiver, and the mechanical errors while rotating the heliostat on its axes.

3.3 Reinforcement Learning in Differentiable Simulations

Wiedemann et al. (2022)[31] proposes Analytical Policy Gradient learning (APG), which exploits the availability of differentiable simulators by training a controller offline with gradient descent on the tracking error.

They also provide comparison to both Model Predictive Control (MPC) and Proximal Policy Optimization (PPO) for its tasks, which also offers a useful benchmark for our own framework.

4 Simulation Environment

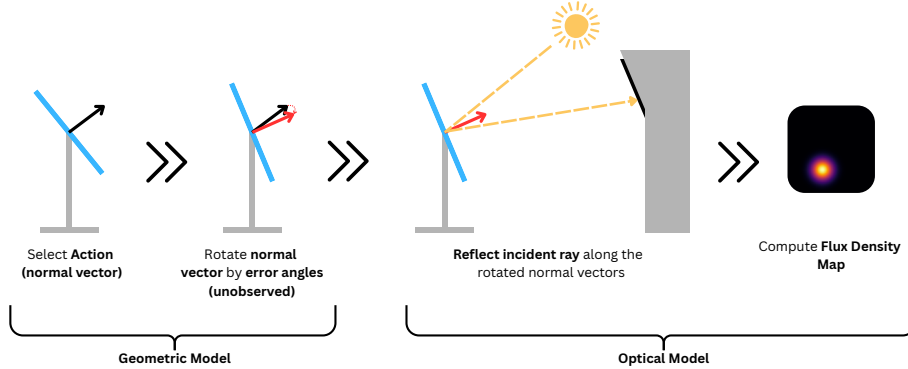


Figure 4: Overview of the differentiable heliostat simulation, depicting the geometric model (error-perturbed normals) and the optical model (ray reflection, receiver intersection, and flux density map computation).

In this section, we define how our simulation performs forward calculations. We first start with by formally defining components of our heliostat field, the sun position, the receiver, and the heliostat, then we describe how we calculate the desired ray direction and the ideal normal vector, and then talk about how rotational errors around the heliostat axis are sampled and applied to our desired normal vector to get a misaligned normal vector. We conclude with details about computing the actual reflected ray direction, its interaction with the receiver surface and the flux density map generation. All of these computations were implemented in PyTorch[22], ensuring end-to-end differentiability. The overall workflow of our environment is summarized in Figure.4.

4.1 Coordinate System

The simulation environment is defined in a right-handed Cartesian coordinate system (Figure.5), where the **X-axis** points East (E), the **Y-axis** points North (N), and the **Z-axis** points Up (U). All distances and geometric quantities, such as heliostat position, target position, receiver dimensions, etc., are expressed in meters. Heliostats are always positioned at the horizontal plane $Z (\text{Up}) = 0$

4.2 Scene Geometry

4.2.1 Mathematical Formulation of the Environment

Heliostat Since our framework does not deal with surface properties of the heliostat, and we only have one heliostat in our current framework, the heliostat can be completely represented by a position vector $\mathbf{h} \in \mathbb{R}^3$.

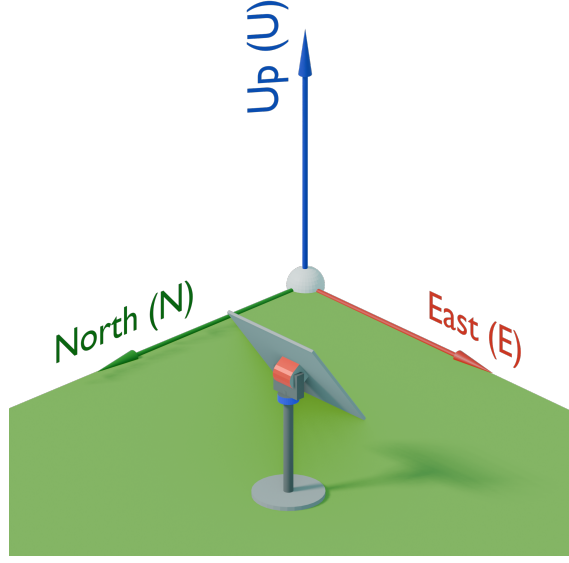


Figure 5: Global coordinate system of the differentiable heliostat environment, showing the East (X), North (Y), and Up (Z) axes

Target (receiver) The target is defined by its position and normal: $\mathbf{c}_{\text{rec}} \in \mathbb{R}^3$, $\hat{\mathbf{n}}_{\text{rec}} \in \mathbb{R}^3$ with $\|\hat{\mathbf{n}}_{\text{rec}}\| = 1$. The target plane is spanned by orthogonal basis vectors $\hat{\mathbf{u}}_{\text{rec}}, \hat{\mathbf{v}}_{\text{rec}}$ (aligned with Up–Z, East–X axes). In our experiments, these vectors are:

$$\mathbf{c}_{\text{rec}} = \begin{bmatrix} 0 \\ -5 \\ 0 \end{bmatrix}, \quad \hat{\mathbf{n}}_{\text{rec}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \hat{\mathbf{u}}_{\text{rec}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \hat{\mathbf{v}}_{\text{rec}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Hence we define the target as the tuple $\mathbf{y} = (\mathbf{c}_{\text{rec}}, \hat{\mathbf{n}}_{\text{rec}}, \hat{\mathbf{u}}_{\text{rec}}, \hat{\mathbf{v}}_{\text{rec}})$.

Sun vector The sun position (Figure.6) $\vec{s} \in \mathbb{R}^3$ is sampled from a cone and then normalized. First, a central axis \mathbf{a} is constructed from azimuth $\phi \in [0, 360^\circ)$ and elevation $\theta \in [0, 90^\circ]$ angles (in degrees) as

$$\mathbf{a} = \begin{bmatrix} \cos(\theta) \cos(\phi) \\ \cos(\theta) \sin(\phi) \\ \sin(\theta) \end{bmatrix}, \quad \hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|}.$$

To obtain a uniform distribution of sun directions within a cone, we parameterize the cone in spherical coordinates relative to its central axis. A naïve uniform sampling of the polar angle would bias points towards the cone’s rim; instead, one samples $\cos \vartheta$ uniformly. Concretely, we draw

$$\mu \sim \mathcal{U}[0, 1], \quad \cos \vartheta = 1 - \mu(1 - \cos \alpha),$$

which ensures that $\cos \vartheta$ is uniformly distributed on the interval $[\cos \alpha, 1]$, corresponding to polar angles $\vartheta \in [0, \alpha]$ with correct density. A locally azimuthal angle

$$\varphi \sim \mathcal{U}[0, 2\pi)$$

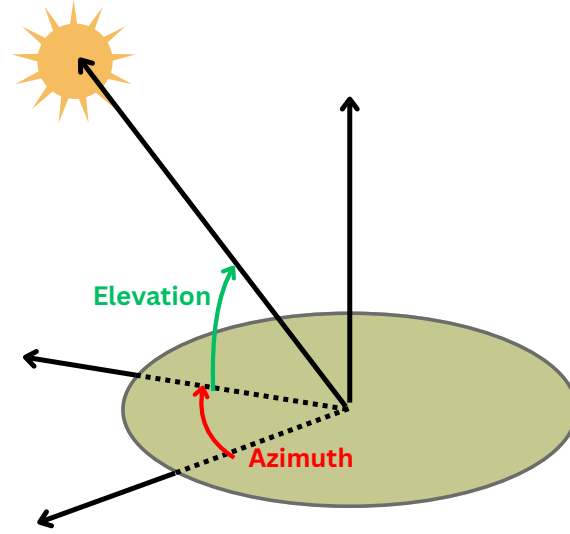


Figure 6: Sun-direction parameterization. The azimuth ϕ (red) and elevation θ define the central axis \mathbf{a}

is drawn independently to provide a random rotation around the cone axis. Together, the pair (ϑ, φ) specifies a direction vector inside the spherical cap of half-aperture α in a uniform manner.

Then, we define the local direction as

$$\hat{\vec{s}} = \cos \vartheta \hat{\mathbf{a}} + \sin \vartheta (\cos \varphi \hat{\mathbf{u}}_{\text{sun}} + \sin \varphi \hat{\mathbf{v}}_{\text{sun}}),$$

where $\{\hat{\mathbf{u}}_{\text{sun}}, \hat{\mathbf{v}}_{\text{sun}}, \hat{\mathbf{a}}\}$ is an orthonormal basis with axis $\hat{\mathbf{a}}$. Finally, the sun vector is obtained as

$$\vec{s} = \frac{\hat{\vec{s}}}{\|\hat{\vec{s}}\|}.$$

This procedure ensures that \vec{s} is uniformly distributed on the spherical cap defined by the cone around the azimuth–elevation direction.

Ideal Normal Vector Calculation For the heliostat \mathbf{h} , the incident ray direction \mathbf{d}^{inc} and the desired reflected ray direction $\mathbf{d}^{\text{*ref}}$ are defined as

$$\mathbf{d}^{\text{inc}} = \frac{\mathbf{h} - \vec{s}}{\|\mathbf{h} - \vec{s}\|}, \quad \mathbf{d}^{\text{*ref}} = \frac{\mathbf{c}_{\text{rec}} - \mathbf{h}}{\|\mathbf{c}_{\text{rec}} - \mathbf{h}\|}.$$

The corresponding ideal unit normal is computed as

$$\mathbf{n}_{\mathbf{h}}^* = \frac{\mathbf{d}^{\text{*ref}} - \mathbf{d}^{\text{inc}}}{\|\mathbf{d}^{\text{*ref}} - \mathbf{d}^{\text{inc}}\|}.$$

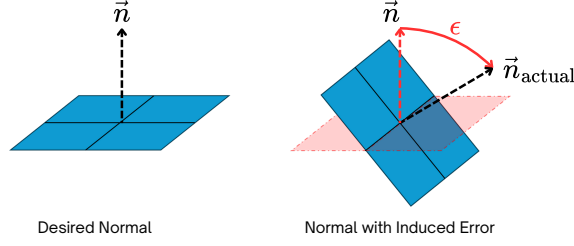


Figure 7: Left: Desired heliostat normal. Right: Heliostat normal after inducing alignment error on an axis

4.2.2 Orientation Errors (in mrad) and Rotation

To enable our policy to adapt to varying mechanical misalignment, for each heliostat, small orientation errors about the Up (Z) and East (X) axes are sampled. Figure.7 illustrates how a heliostat rotates after an error of huge magnitude is induced into one of its axis.

$$\delta_e, \delta_u \underset{\text{i.i.d.}}{\sim} \mathcal{U}[0, \Theta^\delta] \text{ [mrad]},$$

and converted to radians via $\theta_e = \delta_e \cdot 10^{-3}$, $\theta_u = \delta_u \cdot 10^{-3}$. The normal is rotated by $R_x(\theta_e)R_z(\theta_u)$, applied component-wise to a batch. Writing $\mathbf{n} = (x, y, z)^\top$,

$$R_z(\theta_u)\mathbf{n} = \begin{bmatrix} \cos \theta_u & -\sin \theta_u & 0 \\ \sin \theta_u & \cos \theta_u & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{n} = \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix},$$

$$R_x(\theta_e) \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_e & -\sin \theta_e \\ 0 & \sin \theta_e & \cos \theta_e \end{bmatrix} \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}.$$

Up-component rectification Since the heliostat’s normal vector pointing downwards is neither realistic, and is neither physically feasible, after rotation, the z -component (“Up”) of the heliostat normal vector is passed through a ReLU(subsection A.2) and re-normalized. We define this function as follows:

$$\text{rect}([x, y, z]) = \frac{(x, y, \text{ReLU}(z))}{\|(x, y, \text{ReLU}(z))\|}$$

Actual mirror normal Hence the resulting normal after applying the above perturbations to our selected normal vector \mathbf{n}_h is :

$$\mathbf{n}_h^{\text{actual}} = \text{rect}(R_x(\theta_e)R_z(\theta_u)\mathbf{n}_h)$$

In both our experiments and practical applications, the vector $\mathbf{n}_h^{\text{actual}}$ is not directly observable.

4.3 Flux density map Computation

4.3.1 Specular Reflection

For a given incident direction \mathbf{d}^{inc} and a surface normal $\mathbf{n}_h^{\text{actual}}$, the reflected *unit* direction used in our environment is

$$\hat{\mathbf{d}}^{\text{ref}} = \mathbf{d}^{\text{inc}} - 2(\mathbf{d}^{\text{inc}} \cdot \mathbf{n}_h^{\text{actual}}) \mathbf{n}_h^{\text{actual}}.$$

4.3.2 Ray–Plane Intersection

The reflected ray is parametrized as $\rho(\kappa) = \mathbf{o} + \kappa \hat{\mathbf{d}}^{\text{ref}}$, where $\mathbf{o} = \mathbf{h}$ is the ray *origin* (the heliostat position) and $\hat{\mathbf{d}}^{\text{ref}}$ is the *unit direction vector* of the reflected ray. The target surface is represented as a plane

$$\{\mathbf{x} : (\mathbf{x} - \mathbf{c}_{\text{rec}}) \cdot \hat{\mathbf{n}}_{\text{rec}} = 0\},$$

with \mathbf{c}_{rec} a *reference point* on the plane (e.g. the receiver center) and $\hat{\mathbf{n}}_{\text{rec}}$ the *unit normal* of the plane.

The intersection parameter κ is found via

$$\text{denom} = \hat{\mathbf{d}}^{\text{ref}} \cdot \hat{\mathbf{n}}_{\text{rec}}, \quad \kappa = \frac{(\mathbf{c}_{\text{rec}} - \mathbf{o}) \cdot \hat{\mathbf{n}}_{\text{rec}}}{\text{denom}},$$

guarded by the condition $|\text{denom}| > \varepsilon$ with small ε to avoid division by nearly zero. If this condition is violated, the ray is considered *invalid* (parallel to the plane). Otherwise, the intersection point is

$$\mathbf{q} = \mathbf{o} + \kappa \hat{\mathbf{d}}^{\text{ref}}.$$

For implementation, we keep a per-ray *validity mask* $m \in \{0, 1\}$, equal to 1 for valid intersections and 0 otherwise.

4.3.3 Target Plane Grid

Let the image resolution be $N_u \times N_v$ pixels. Define equispaced coordinates

$$u_m \in \left[-\frac{L_u}{2}, \frac{L_u}{2}\right], \quad v_n \in \left[-\frac{L_v}{2}, \frac{L_v}{2}\right], \quad m = 1, \dots, N_u, \quad n = 1, \dots, N_v,$$

and construct the grid points

$$\mathbf{X}_{m,n} = \mathbf{c}_{\text{rec}} + u_m \hat{\mathbf{u}}_{\text{rec}} + v_n \hat{\mathbf{v}}_{\text{rec}}.$$

4.3.4 Distance-Dependent Gaussian Kernel (Per Ray)

For the intersection \mathbf{q} , define the heliostat-to-intersection distance

$$D = \|\mathbf{q} - \mathbf{h}\|, \quad \sigma = \max\{\kappa D, \epsilon\},$$

with small ϵ for numerical stability and a scale factor $\kappa > 0$. The (unnormalized) Gaussian kernel on the grid is then

$$G(m, n) = \exp\left(-\frac{\|\mathbf{X}_{m,n} - \mathbf{q}\|^2}{2\sigma^2}\right) \cdot m,$$

5 Problem Formulation

In this chapter, we provide a mathematical formalization for our heliostat-control task as an optimization problem. In the following sections, we formulate our control task as a *Partially Observable Markov Decision Process* (POMDP), taking into account the limited access to observable inputs for our agent. The objective for the agent is to discover a policy which maximizes the long-term alignment performance for heliostats. Figure.8 demonstrate the complete RL setup for our task.

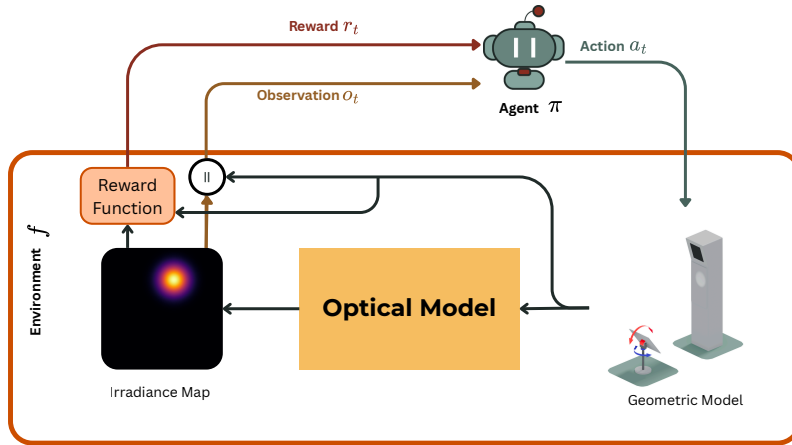


Figure 8: Reinforcement-learning formulation of the heliostat-control task. The agent observes the flux-density images and auxiliary features, selects mirror-normal actions, and receives rewards based on the environment’s reward function.

5.1 Partially Observable Markov Decision Process (POMDP) for Heliostat Aiming

We formulate control as a discrete-time finite-horizon Partially Observable Markov Decision Process. Formally, we define the POMDP as a 7-tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \Omega, \mathcal{O}, P, \mathcal{R}, \gamma), \quad t = 0, 1, 2, \dots$$

Where the components are defined as follows:

5.1.1 State space

The state space \mathcal{S} is the set of all possible states s_t of the environment at time step t . This is expressed as the Cartesian product:

$$\mathcal{S} = \mathcal{C} \times H \times \Phi \times \Psi,$$

where \mathcal{C} is the set of all receiver configurations, defined as the set of all possible tuples \mathbf{y} . H is the set of all heliostat positions $\mathbf{h} \in H$, $H \subset \mathbb{R}^3$. Φ is the set of all

possible alignment errors, such that $\Phi = [0, \Theta^\delta]^2$, $\delta = (\delta_e, \delta_u) \in \Phi$, where Θ^δ is the maximum alignment error in mrad. Ψ is the set of all sun positions $\vec{s} \in \Psi$, $\Psi \subset \mathbb{R}^3$. Intuitively, \mathcal{S} is the set of latent states, which includes components which the agent might not observe. Within an episode, the true state $s = (\mathbf{y}, \mathbf{h}, \delta, \vec{s})$ remains fixed and does not change over the timesteps.

5.1.2 Action Space

The action space \mathcal{A} is a set of all possible actions which the agent can perform. In our framework, the agent produces the normal vector for our heliostat, we can formally define our action space as

$$\mathcal{A} = \mathcal{N}, \mathcal{N} = \{\mathbf{n} \in \mathbb{R}^3 : \|\mathbf{n}\| = 1\}$$

5.1.3 Observation Space

The Observation Space \mathcal{O} is the set of all possible observations which the agent receives in our framework. This is defined as the Cartesian product

$$\mathcal{O} = \mathbb{R}^{N_u \times N_v} \times \Psi \times H \times \mathcal{N}$$

with $o_t = (\mathbf{I}_t, \vec{s}, \mathbf{h}, \mathbf{n}_{t-1})$, $o_t \in \mathcal{O}$, where \mathbf{I}_t is the flux density map observed after taking the action a_{t-1} .

5.1.4 Transition Model

Because \mathbf{y} , δ , \vec{s} and \mathbf{h} are stationary within an episode, the latent state does not change after taking an action. Hence the state is static, and the observation model carries all action-dependencies via the rendering pipeline. where $\delta_{s_t}(s_{t+1})$ is the Dirac delta distribution w.r.t. s_t (Appendix A.3).

$$P(s_{t+1} | s_t, a_t) = \delta_{s_t}(s_{t+1}),$$

5.1.5 Observation Model

In our framework, we consider the simulator as a differentiable function

$$f : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}^{N \times N} \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R},$$

which maps the current latent state and applied action to the next observation and reward. Given the current (constant) state $s_t = s = (\mathbf{y}, \mathbf{h}, \delta, \vec{s})$, and the action a_t , the environment forms the actual normal, using our error model

$$o_t = (\mathbf{I}_t, \vec{s}, \mathbf{h}, \mathbf{n}_{t-1}) = f(s_t, a_t)$$

(according to our environment model) Thus, our observation kernel is deterministic given s_t and can be expressed as a $\Omega(o_{t+1} | s_t, a_t) = \delta_{(\mathbf{I}_{t+1}, \vec{s}, \mathbf{h}, a_t)}(o_{t+1})$

5.1.6 Reward Function

The agent is trained to *maximize* the alignment quality of the heliostat orientation. At each time step t , we define the alignment error as the angular deviation (in milliradians) between the actual mirror normal and the ideal one:

$$\ell_{\text{align}}(t) = 10^3 \arccos\left(\frac{\mathbf{n}_t^{\text{actual}} \cdot \mathbf{n}_{h,t}^*}{\|\mathbf{n}_t^{\text{actual}}\| \|\mathbf{n}_{h,t}^*\|}\right) \text{ [mrad]}. \quad (1)$$

It's important to note that in practice, a small value of $\varepsilon = 10^{-7}$ is added to the denominator of the $\arccos(\cdot)$ function to ensure numerical stability during training, however as a result of this, the numerical floor of our alignment loss function is > 0 (which was tested and found to be 0.3462mrad). The instantaneous reward is chosen as the *negative* alignment error, so that maximizing reward is equivalent to minimizing the misalignment:

$$r_t = \begin{cases} -\ell_{\text{align}}(t), & \text{if } t = T, \\ 0, & \text{otherwise,} \end{cases} \quad (2)$$

Thus the return maximized by the agent is

$$J(\pi) = \mathbb{E}_{\pi,P} \left[\sum_{t=0}^T \gamma^t r_t \right] = - \mathbb{E}_{\pi,P} \left[\sum_{t=0}^T \gamma^t \ell_{\text{align}}(t) \right].$$

Intuitively, this means that the agent is rewarded whenever its predicted mirror normal closely matches the ideal one. The closer the alignment (i.e., the smaller the angular error), the higher the reward. By maximizing cumulative reward, the policy learns to minimize systematic orientation errors over time.

5.1.7 Reward Discount

In our formulation we set the discount factor to $\gamma = 1$. This choice is motivated by the fact that we only get a single reward value at the terminal episode, hence calculating a weighted average over rewards provides no visible advantage. Moreover, implementing a non-zero γ in practice in our current setup is not very straightforward.

5.2 Evaluation metric (not optimized during training).

For reporting only, we compute the mean squared error (MSE) between the target flux map and the flux induced by the agent's action:

$$\text{MSE}(t) = \frac{1}{N_u N_v} \sum_{m=1}^{N_u} \sum_{n=1}^{N_v} (I_{\text{pred}}[m, n; t] - I_{\text{targ}}[m, n; t])^2,$$

where I_{pred} is obtained from the reflected rays defined by a_t and the current scene, and I_{targ} is the desired flux density map.

6 Architecture and Model Training

6.1 Overview

In this chapter of the thesis, we talk about the current neural-network architecture and the training procedures used for our heliostat-control policy. We define how our model processes sequences of flux density maps, extracts geometric features, encodes temporal dynamics, and outputs normalized surface normals for our heliostat. We further discuss the analytical policy gradient formulation, backpropagation through time, gradient stopping for inputs, and the fine-adjustment layer used in our training.

6.2 Network Architecture

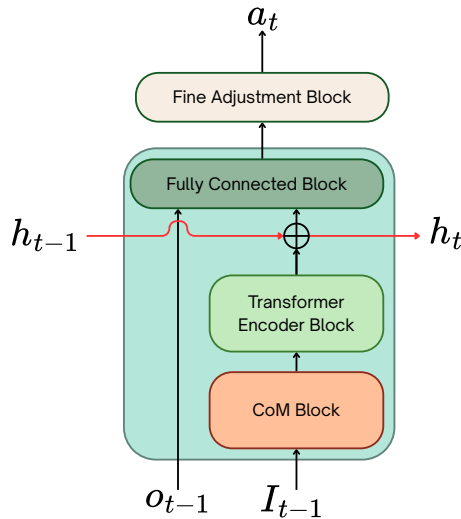


Figure 9: The policy architecture of our agent. Consisting of a Center of Mass Block, a transformer encoder block (optionally replaceable with LSTM / MLP) a final MLP head and a fine-adjustment layer

In the following sections, we describe the components of our agent’s policy (Figure.9) network and discuss how each of the component processes visual and auxiliary input to produce the agent’s actions.

6.2.1 Sequence Input

Instead of operating on a single flux density map in isolation, at each decision step t , the network maintains a history of the most recent k flux density maps:

$$\mathcal{H}_t = \{I_{t-k+1}, I_{t-k+2}, \dots, I_t\}.$$

This temporal context lets the policy exploit the change in flux distributions rather than relying solely on the instantaneous image. In addition to this, our architecture

also uses a dedicated mechanism for temporal dependencies through the hidden state that is updated across time-steps (detailed later in section 6.2.3). However, the flux density map sequence input lets us reinforce this temporal context so that our policy doesn't rely solely on the hidden state as the source of sequential information.

6.2.2 Center-of-Mass (CoM) Block

Each frame $I_\tau \in \mathcal{H}_t$ is passed through a differentiable *center-of-mass (CoM)* operator that reduces the 2D flux density map to a compact geometric descriptor. The CoM operator extracts the flux centroid, which provides a computationally fast way of preserving alignment-relevant spatial information while ignoring irrelevant pixel-level details.

Center of Mass operator Given an image $x \in \mathbb{R}^{H \times W}$ with non-negative pixel intensities $x_{i,j} \geq 0$ (treated as mass values), the two-dimensional center of mass (CoM) is defined as the intensity-weighted average of the pixel coordinates.

Let the horizontal and vertical coordinate grids be

$$X_{i,j} = j, \quad Y_{i,j} = i,$$

for $i = 0, \dots, H-1$ and $j = 0, \dots, W-1$.

The total mass of the image is

$$M = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} x_{i,j},$$

and the weighted sums of the coordinates are

$$X^{(w)} = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} j x_{i,j}, \quad Y^{(w)} = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} i x_{i,j}.$$

The center of mass coordinates $(x_{\text{com}}, y_{\text{com}})$ are then

$$x_{\text{com}} = \frac{X^{(w)}}{M + \varepsilon}, \quad y_{\text{com}} = \frac{Y^{(w)}}{M + \varepsilon},$$

where $\varepsilon > 0$ is a small constant (e.g. 10^{-12} in our experiments) added for numerical stability when M is close to zero.

In the degenerate case $M = 0$ (no mass in the image), the CoM is defined as the sentinel value

$$(x_{\text{com}}, y_{\text{com}}) = (-1, -1).$$

Output The resulting features are then projected into an embedding space via a linear layer with nonlinearity and dropout.

6.2.3 Temporal Encoding

The CoM block provides a compact representation of each flux density map frame, however, the policy also needs to utilize how the flux distribution pattern changes over time when our agent keeps taking actions over the steps.

In order to do so, as a first step, the agent maintains a history of the last k frames, which are mapped into feature vectors and stacked to form a temporal sequence:

$$\{\mathcal{v}_{t-k+1}, \mathcal{v}_{t-k+2}, \dots, \mathcal{v}_t\}.$$

The purpose of the temporal encoder is to extract sequential dependencies from this history, thereby complementing the purely instantaneous information of the CoM features. We investigate three encoder families:

Transformer A 2-layer Transformer encoder takes the feature sequence $(\mathcal{v}_{t-\tau:t})$ (context length τ) to capture temporal dynamics. Additionally, we augment the Transformer encoder with a persistent hidden-feature state h_t at its output. This enables temporal credit assignment even with a single-frame context window ($\tau = 1$).

Let z_t denote the last-token embedding output by the encoder given the current image window. We update

$$h_t = z_t + h_{t-1}, \quad h_0 = \mathbf{0},$$

A self-attention encoder capable of modeling long-range dependencies within the k -frame window. In addition to the standard transformer formulation, our architecture introduces a temporal carry-over mechanism that allows information to persist across rollouts.

LSTM A recurrent encoder consisting of hidden and a cell states that are updated at every timestep. This is designed to naturally retain memory of past frames and enforces temporal consistency across predictions.

MLP A feed-forward baseline that takes in only the most recent τ frames. The temporal context of this is limited to the number of frames it takes in. This provides a lightweight comparison point and demonstrates the value of incorporating other explicit temporal contexts the policy.

By design, our architecture can interchangeably employ any of these encoders with ease depending on our experimental setting.

6.2.4 Final MLP Head and Action Normalization

The temporal encoder outputs a feature vector $\mathbf{z}_t \in \mathbb{R}^{d_z}$ at timestep t . This is concatenated with an auxiliary input $\mathbf{x}_t \in \mathbb{R}^{3+3}$, formed by joining the unit sun direction

$\vec{s} \in \mathbb{R}^3$ with the applied action from the previous step $a_{t-1} \in \mathbb{R}^3$:

$$\mathbf{x}_t = [\vec{s} \| a_{t-1}].$$

The combined representation

$$\mathbf{u}_t = [\mathbf{z}_t \| \mathbf{x}_t] \in \mathbb{R}^{d_z+6}$$

is passed through a multi-layer perceptron (MLP), which outputs

$$\tilde{a}_t \in \mathbb{R}^3.$$

The action is updated recursively by adding the new prediction to the previous action and normalizing the final output:

$$a_t = \frac{a_{t-1} + \tilde{a}_t}{\|a_{t-1} + \tilde{a}_t\|},$$

This formulation can be interpreted as a residual update rule[10], i.e., the network predicts an *incremental correction* \tilde{a}_t to the previous action a_{t-1} , rather than a full action from scratch. Such residual connections are known to stabilize training and improve convergence by encouraging incremental refinements instead of large changes.

At the first timestep $t = 1$, we consider the action a_0 to be the 3-dimensional zero vector

$$a_0 = \mathbf{0}.$$

6.2.5 Gradient Stop Operation

To improve stability, we apply a gradient-stop operator $\text{sg}(\cdot)$ to both the flux density map history and the auxiliary input before they are passed into the policy network:

$$\tilde{\mathbf{I}}_{t-k+1:t} = \text{sg}(\mathbf{I}_{t-k+1:t}), \quad \tilde{\mathbf{x}}_t = \text{sg}([\vec{s} \| a_{t-1}]).$$

Here, $\text{sg}(\cdot)$ is defined as

$$\frac{\partial}{\partial \mathbf{z}} \text{sg}(\mathbf{z}) = \mathbf{0},$$

This implies that the forward pass is unaffected ($\text{sg}(\mathbf{z}) = \mathbf{z}$) but no gradients are propagated backward through \mathbf{z} .

While unrolling long sequences, this mechanism prevents exploding or unstable gradients, particularly under backpropagation through time (BPTT) [30]. It ensures that temporal dependencies are modeled primarily through the recurrent/transformer state, while the inputs to the policy network themselves act as fixed conditioning signals during gradient updates.

6.2.6 Fine-Adjustment Layer

We consider the prediction produced by the above network as a *coarse prediction*, i.e., it is already sufficient to direct flux onto the receiver, ensuring that images are neither blank nor far from the desired focal spot. However, to refine this estimate further, we introduce a *fine-adjustment* (test-time compute)[9] layer. This layer applies a small correction directly in the action space, nudging the flux density closer to the intended target flux density location on the receiver surface. The mechanism leverages the differentiability of the environment to optimize this correction with respect to an image-based error while keeping the policy parameters fixed. It is only through the inclusion of this refinement step that our method qualifies as a fully *model-based* approach. A detailed description of this component is provided later in Section. 6.3.4.

6.3 Training Procedure

6.3.1 Analytical Policy Gradients

We adopt the *Analytic Policy Gradient (APG)*[31] approach, which leverages the differentiability of both the policy π_θ and the environment model f to compute exact gradients of the resulting reward function with respect to the policy parameters via the chain rule:

$$\nabla_\theta J(\pi_\theta) = \frac{\partial r}{\partial a_t} \cdot \frac{\partial a_t}{\partial \theta},$$

where a_t is the action predicted by our policy π at time t and r_t denotes the reward signal produced by our differentiable environment f .

Unlike likelihood–ratio methods such as REINFORCE[32, 33], Actor-Critic[13], PPO[27], or TRPO[26] which rely on Monte Carlo sampling to estimate $\nabla_\theta J$, APG provides us with the exact gradient information, letting us update our policy directly using standard gradient-based methods (e.g. SGD, Adam), without the stochastic noise typically associated with sampling-based estimators, and thus leads to more stable training, higher sample efficiency, and faster convergence.

In our setting, the environment simulator f is fully differentiable, making APG a straightforward choice.

6.3.2 Sampling a Trajectory over Fixed Horizon

Training proceeds by rolling out trajectories of fixed length T . At time t , the policy-network produces a *raw* output

$$\tilde{a}_t = \pi_\theta(o_{t-1}, h_{t-1}) \in \mathbb{R}^3, \quad \text{with } \frac{\partial \tilde{a}_t}{\partial o_{t-1}} = 0 \text{ (gradient stop operation),}$$

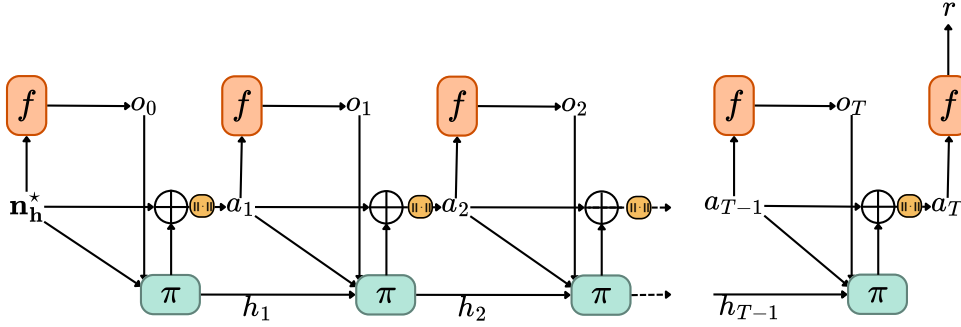


Figure 10: The unrolled trajectory graph for our Analytical Policy Gradient setup. The initial o_0 is obtained by taking the ideal normal as the action and passing it to our environment f .

which is added to the previous applied action via a residual update and then normalized as

$$v_t = a_{t-1} + \tilde{a}_t \in \mathbb{R}^3, \quad a_t = \frac{v_t}{\|v_t\|} \in \mathbb{R}^3,$$

Thus the network predicts *increments* \tilde{a}_t and the residual path $a_{t-1} \rightarrow a_t$ enforces temporal smoothness.

Environment transition and reward We distinguish the latent state s_t from the observable o_t . Given the current state and applied action, the differentiable simulator provides the next observation and reward,

$$(o_t, r_t) = f(s_{t-1}, a_t),$$

while the (possibly separate) differentiable dynamics update the latent state,

$$s_t = g(s_{t-1}, a_t).$$

In our experiments, the reward is terminal-only:

$$r_t = \begin{cases} 0, & t < T, \\ f_r(s_T, a_T), & t = T, \end{cases}$$

so the objective reduces to $J = r_T$. As noted earlier, gradients do not flow through observations ($\partial \tilde{a}_t / \partial o_{t-1} = 0$) due to the stop-gradient operator applied at the policy input.

The environment provides the initial observation o_0 , at the beginning of the episode, which consists of the initial flux density map, the sun position, and the ideal normal vector (as a best initial guess) as features. We initialize the applied action as the zero vector

$$a_0 = \mathbf{0} \in \mathbb{R}^3,$$

At the first decision step, the policy π_θ takes the observation o_0 and produces an action $a_1 \in \mathbb{R}^3$. This action is passed into the differentiable simulator f , which applies it to

the heliostat field, renders the resulting flux density map distribution on the receiver plane, and returns the next observation o_1 together with a reward signal r_1 .

The same procedure is repeated iteratively for $t = 1, \dots, T$: at each timestep, the policy π_θ maps the current state s_{t-1} to an action a_t , the environment simulator f produces the new observation o_t , and a reward r_t is computed. Thus, we represent a full trajectory as

$$(o_0, a_1, r_1, o_1, a_2, r_2, \dots, o_{T-1}, a_T, r_T, s_T).$$

This rollout (Figure.10) procedure forms the basic training sample for the analytical policy gradient updates described in the previous section.

6.3.3 Backpropagation Through Time

Local Jacobians for normalization The Jacobian of block-normalization $\hat{v} = \text{norm}(v) = v/\|v\|$ is

$$\nabla_{v_t} a_t = \frac{\partial a_t}{\partial v_t} = \frac{1}{\|v_t\|} (I_3 - a_t a_t^\top) \in \mathbb{R}^{3 \times 3},$$

where $v_t = a_{t-1} + \tilde{a}_t$. Hence

$$\nabla_{a_{t-1}} a_t = \frac{\partial a_t}{\partial a_{t-1}} = \frac{\partial v_t}{\partial a_{t-1}} \frac{\partial a_t}{\partial v_t} = \nabla_{v_t} a_t, \quad \text{and} \quad \nabla_{\tilde{a}_t} a_t = \frac{\partial a_t}{\partial \tilde{a}_t} = \frac{\partial v_t}{\partial \tilde{a}_t} \frac{\partial a_t}{\partial v_t} = \nabla_{v_t} a_t,$$

Backprop from Terminal Reward Our differentiable simulator f provides the gradients of the reward $r = f_r(s_T, a_T)$ with respect to last action a_T directly as:

$$\nabla_{a_T} r = \frac{\partial f_r(s_T, a_T)}{\partial a_T}.$$

We can then recursively backpropagate through our previous action a_T and all the actions before as:

$$\nabla_{a_t} r = \frac{\partial r}{\partial a_t} = \frac{\partial a_t}{\partial a_{t-1}} \frac{\partial r}{\partial a_t} = \nabla_{a_{t-1}} a_t \nabla_{a_t} r = \nabla_{v_t} a_t \nabla_{a_t} r$$

Gradients w.r.t. policy output, parameters, and hidden state Since a_t depends on \tilde{a}_t through the same normalization,

$$\nabla_{\tilde{a}_t} r = \frac{\partial r}{\partial \tilde{a}_t} = \frac{\partial a_t}{\partial \tilde{a}_t} \frac{\partial r}{\partial a_t} = \nabla_{\tilde{a}_t} a_t \nabla_{a_t} r = \nabla_{v_t} a_t \nabla_{a_t} r$$

For our policy-network's parameters θ_π ,

$$\nabla_{\theta_\pi}^{(a)} r^{(t)} = \frac{\partial r}{\partial \theta_\pi} = \frac{\partial \tilde{a}_t}{\partial \theta_\pi} \frac{\partial r}{\partial \tilde{a}_t} = \nabla_{\theta_\pi} \tilde{a}_t \nabla_{\tilde{a}_t} r,$$

Gradients of the reward w.r.t. the policy parameters $\nabla_{\theta_\pi} r^{(t)}$ also flows through the hidden state h_t as follows:

$$\begin{aligned}\nabla_{h_t} r &= \frac{\partial r}{\partial h_t} = \frac{\partial \tilde{a}_t}{h_t} \frac{\partial r}{\partial \tilde{a}_t} = \nabla_{h_t} \tilde{a}_t \nabla_{\tilde{a}_t} r \\ \nabla_{\theta_\pi}^{(h)} r^{(t)} &= \frac{\partial h_t}{\partial \theta_\pi} \frac{\partial r}{\partial h_t} = \frac{\partial h_t}{\partial \theta_\pi} \nabla_{h_t} r\end{aligned}$$

We provide a complete derivation of $\frac{\partial h_t}{\partial \theta_\pi}$ for our augmented transformer architecture in the Appendix. Finally, we accumulate our gradients $\nabla_{\theta_\pi} r$ as follows:

$$\nabla_{\theta_\pi} r = \sum_{t=1}^T \left(\underbrace{\nabla_{\theta_\pi}^{(a)} r^{(t)}}_{\text{direct action path}} + \underbrace{\nabla_{\theta_\pi}^{(h)} r^{(t)}}_{\text{hidden-state path}} \right) = \sum_{t=1}^T \left[\nabla_{\theta_\pi} \tilde{a}_t \nabla_{\tilde{a}_t} r + \frac{\partial h_t}{\partial \theta_\pi} \nabla_{h_t} \tilde{a}_t \nabla_{\tilde{a}_t} r \right].$$

6.3.4 Test-Time Adjustment Layer (Fine Adjustment)

At inference time we augment the policy output with a small, randomly initialized optimizable offset that is adapted online using the differentiable simulator. Let $\bar{a}_t \in \mathbb{R}^3$ denote the raw policy action at step t and $\text{reshape}(\cdot)$ split it into per-heliostat vectors. The applied action is

$$a_t = \text{Normalize}(\bar{a}_t + \epsilon_t^{\text{fine}}),$$

where $\epsilon_t^{\text{fine}} \in \mathbb{R}^{K \times 3}$ is a persistent *fine-adjustment* variable updated by a few gradient steps at each time step. This stems from the idea that \bar{a}_t gives us a coarse alignment normal, which produces a flux density near to the center of the receiver but not at the center of the receiver, and we iteratively update the small offset vector (which nudges our agent’s action output) ϵ_t^{fine} based on an image loss between the desired flux density map and the current flux density map.

Image Loss Given the rendered flux density map \mathbf{I}_j produced by our environment after taking action $a_t = \cdot$, and target flux density map \mathbf{Y} , to penalize dissimilar images, we define a distance map[3] weighted error as follows:

1. We normalize both maps by the maximum of the target flux: Let

$$Y_{\max} := \max_{x,y} \mathbf{Y}(x,y),$$

and define the normalized maps

$$\bar{\mathbf{Y}}(x,y) := \frac{\mathbf{Y}(x,y)}{Y_{\max}}, \quad \bar{\mathbf{I}}_k(x,y) := \frac{\mathbf{I}_k(x,y)}{Y_{\max}}.$$

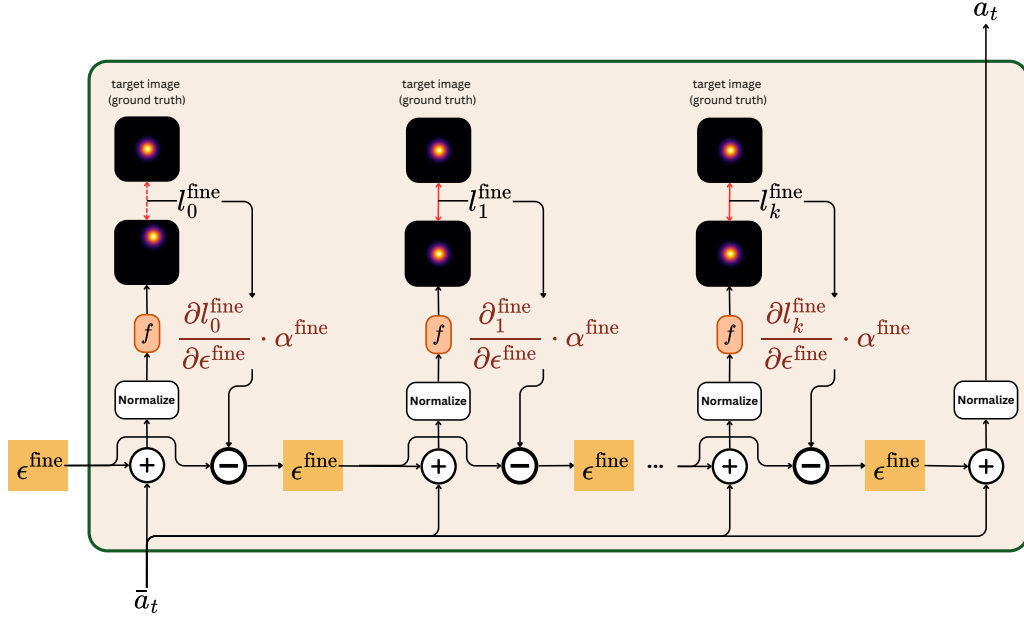


Figure 11: Test-time adjustment layer: a small persistent offset ϵ^{fine} is optimized online via a few gradient steps, first using a distance loss and then MSE, with normalization after each update.

2. For a threshold $\iota \in (0, 1)$ (default $\tau = 0.5$), define the binary mask

$$\mathbf{M}(x, y) := \mathbf{1}[\mathbf{Y}(x, y) > \iota Y_{\max}].$$

3. The associated Euclidean per pixel distance transform is

$$D(x, y) := \min_{(u, v): \mathbf{M}(u, v)=1} \sqrt{(x-u)^2 + (y-v)^2}.$$

4. The per-pixel absolute error is

$$E(x, y) := | \bar{\mathbf{I}}_k(x, y) - \bar{\mathbf{Y}}(x, y) |.$$

5. The distance map weighted error for a single heatmap is

$$\ell^{\text{fine}} := \sum_{x=1}^H \sum_{y=1}^W D(x, y) E(x, y).$$

Inner-loop update. With the policy held fixed (no gradients to θ), we run K steps of first-order optimization on ϵ^{fine} :

$$\epsilon_{t,0}^{\text{fine}} \sim \mathcal{U}(-\eta, \eta), \quad (3)$$

$$\epsilon_{t,j+1}^{\text{fine}} = \epsilon_{t,j}^{\text{fine}} - \alpha^{\text{fine}} \nabla_{\epsilon^{\text{fine}}} \ell_t^{\text{fine}}(\text{Normalize}(\bar{a}_t + \epsilon_{t,j}^{\text{fine}})), \quad j = 0, \dots, K-1. \quad (4)$$

We then apply n_t using $\epsilon_{t,K}^{\text{fine}}$. The gradients are computed through the differentiable optics model f ; the policy output \bar{a}_t is detached during this inner loop to keep test-time compute isolated.

Budget and stability. K is small (e.g. 5–20) and step size α^{fine} is tuned to respect a tight compute budget; we optionally clip ϵ^{fine} or project n_t to maintain unit norms and safety constraints. Figure 11 illustrates the layer.

6.4 Summary of Design Choices

To summarize:

- CoM block compresses flux density maps into low-dimensional features.
- Temporal encoding via LSTM, Transformer, or MLP variants.
- Gradient blocking stabilizes long rollouts.
- Fine-adjustment layer enables model-based refinement.
- Recursive action update ensures continuity across timesteps.
- Analytical policy gradients and BPTT provide efficient and stable optimization.

6.5 Assumptions

1. **Fixed sun position (per episode).** Sun azimuth/elevation are held constant within an episode to isolate control from celestial motion.
2. **Discrete time with step Δt .** Actions are piecewise constant over Δt , matching realistic actuator update rates.
3. **No exogenous disturbances.** We ignore wind, cloud transients, mirror soiling, and tower blockage to focus on alignment control.
4. **Heliostat error model.** The only intrinsic errors are small angular misalignments about the East and Up axes, sampled in milliradians.

7 Results

In this section, we provide the empirical evaluation of our proposed learning framework for heliostat control. Our experiments evaluate the ability of the trained policies to reliably minimize flux misalignment across a varying range of heliostat distances, induced error magnitudes, and evaluation time-steps. We first define our training and testing setup. We then report quantitative results in terms of misalignment reduction, image-loss reduction and compute times.

7.1 Experimental Setup

All experiments were conducted under the following default settings, unless stated otherwise:

- **Heliostat geometry:** A single heliostat placed at a distance of 1500 m from the receiver, aligned on the ground plane.
- **Receiver:** Square target of side 15 m, centered at $\mathbf{c}_{\text{rec}} = (0, -5, 0)^\top$ with surface normal $\hat{\mathbf{n}}_{\text{rec}} = (0, 1, 0)^\top$.
- **Resolution:** Flux maps rendered at 128×128 pixels.
- **Error model:** Mirror orientation perturbations sampled with standard deviation 5 mrad around the ideal normal.
- **Training procedure:** Rollouts of horizon $T = 10$ with context length $k = 2$.
- **Policy architecture:** Transformer backbone with hidden size 128, trained with AdamP optimizer (learning rate $2 \cdot 10^{-4}$, gradient clipping at 10^{-7}).
- **Training Sun Positions:** We select 8 nominal azimuth/elevation pairs. For each pair, 500 sun positions were uniformly sampled within a $\pm 2^\circ$ cone around the nominal direction.
- **Testing Sun Positions:** To evaluate the generalization of our policy, we select 2 nominal azimuth/elevation pairs unseen during training:
 1. An interpolation case, defined as the average of the training azimuth/elevation pairs (unseen by the policy but lies within the spanned range).
 2. An extrapolation case, chosen outside the training range.

For each of these, 60 sun positions were sampled within a $\pm 2^\circ$ cone around the nominal direction.

- **Sample Size:** Each metric was averaged over 5 different runs, corresponding to 5 different random seeds.

7.1.1 Training and Test Sun Directions

For training, we sample $N = 8$ nominal sun positions by varying the azimuth angle in steps of $\Delta\phi = 4^\circ$, starting from $\phi = 15^\circ$, and pairing them with corresponding elevation values such that the maximum deviation in elevation is bounded by 18° . The nominal training points are summarized in Table 1, given as azimuth–elevation pairs in degrees.

Table 1: Nominal training sun positions (azimuth and elevation angles).

Index	Azimuth ($^\circ$)	Elevation ($^\circ$)
1	15	45
2	19	59
3	23	63
4	27	63
5	31	59
6	35	45
7	39	27
8	43	5

In addition to these training points, we define two distinct test scenarios:

- **Interpolation:** The average sun direction across the training distribution, excluding the first three nominal positions. This yields an effective interpolation point at $(\phi \approx 33.0^\circ, \theta \approx 57.0^\circ)$, representing an unseen but in-distribution test configuration.
- **Extrapolation:** A shifted sun position outside the training distribution, defined by $(\phi = 9^\circ, \theta = 43^\circ)$, i.e. an azimuthal offset of $-1.5 \Delta\phi$ and an elevational offset of $-0.5 \Delta\phi$.

Figure 12 visualizes this setup: the black circles mark the training points, while the red highlighted circles correspond to the interpolation and extrapolation test directions.

7.2 Comparison against Baseline

As a baseline, we compare against a **model predictive control (MPC)** method, which optimizes the mirror orientation at each step starting from the ideal normal as initial action. This provides an upper bound on achievable performance under perfect initialization.

Figure.13 illustrates that our APG agent successfully minimizes flux misalignment in both extrapolated and interpolated sun direction scenario cases. Quantitatively, our APG achieves a much lower error levels to MPC within $T = 30$ steps (Table.2).

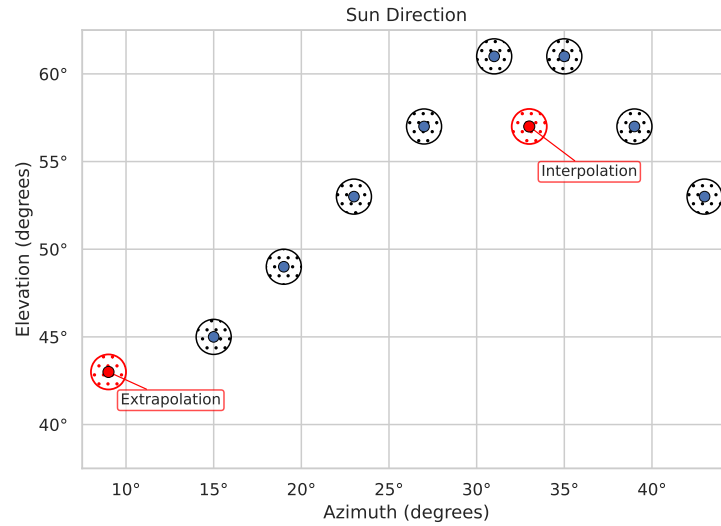


Figure 12: Sun direction sampling strategy. Training positions (black) are placed on a structured azimuth–elevation grid. The test cases are highlighted in red: the average of the training distribution (Interpolation) and an out-of-distribution shifted position (Extrapolation).

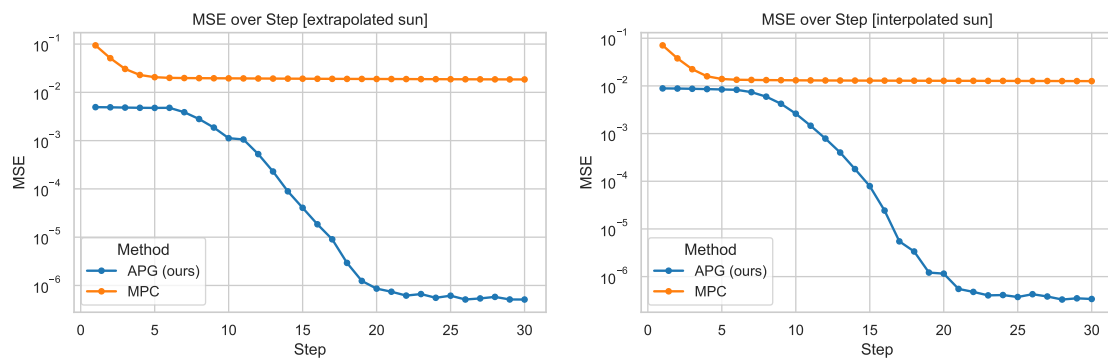


Figure 13: Comparison for MSE over time-steps for our APG framework v/s Model Predictive Control. For MPC, we set the initial guess for the action to be our ideal normal vector and optimize henceforth. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position

Method	Extrapolated sun	Interpolated sun
Agent	0.3462 mrad	0.3472 mrad
MPC	0.9932 mrad	0.8981 mrad

Table 2: Final pointing error (mrad) for unseen sun directions.[lower is better]

Model-free baselines. We also experimented with state-of-the-art model-free reinforcement learning algorithms, including TRPO, SAC, and PPO, using the same observation and reward structure. However, these approaches failed to produce meaningful results: in most runs the policies were unable to direct rays onto the receiver at all, resulting in empty or near-empty flux density maps. We attribute this failure to the high-dimensional continuous action space and the absence of strong geometric priors. We believe that with additional well-crafted constraints and a carefully designed curriculum learning[31, 1], model-free algorithms could become competitive, but due to time limitations, a deeper investigation into this direction was left for future work.

7.3 Flux Density Shift over Steps

We plot the Center of Mass calculated over the flux-density maps for 60 representative roll-out with similar settings. All 60 of these instances have an induced error sampled from $\mathcal{U}[0, 25]$ mrad. This is because later in Section.7.4.2, we see that $[0, 25]$ mrad is the range where our agent almost starts to fail to get to sub 1 mrad error, which is at a sweet spot where after the first step, the centroids are still quite away from the center, however we don't fail to converge to the center.

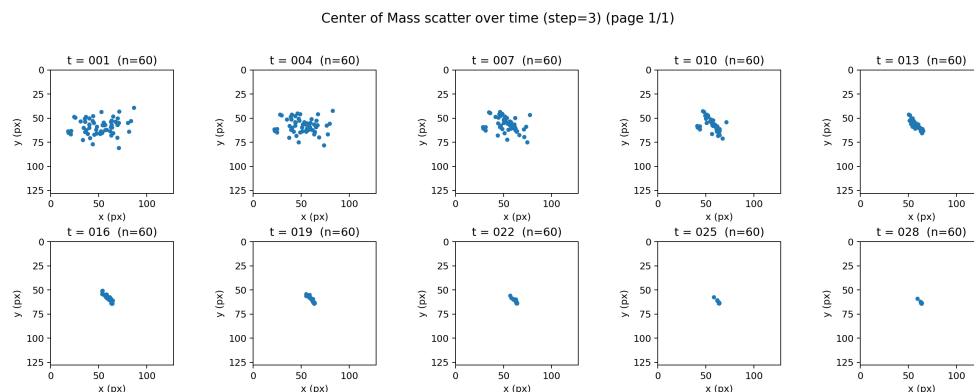


Figure 14: Center of mass for 60 instances rolled out over 30 steps and plotted at every 3 steps. All instances have induced errors sampled from a uniform $[0, 25]$ mrad distribution

We first evaluate the impact of varying the heliostat-receiver distance while keeping all other parameters fixed. Table.3 summarizes the minimum angular errors and the relative time required to reach them.

7.4 Effects of Varying Parameters

In this section, we want to study the effects of each key factors of our framework in. Hence when we vary each of the following property, we keep all the others the same as their default values:

- **Heliostat distance:** 15 m, 150 m, and 1500 m, from the reciever.
- **Induced Error scale:** $\delta \in \{5, 10, 25, 45\}$ mrad.
- **Fine Adjustment:** We test with Fine adjustment always on, always off and only on during testing.
- **Fine Adjustment steps:** $K \in \{5, 10\}$.
- **Training horizon:** Extended rollouts with $T \in \{5, 10, 15\}$.
- **Sequence Length:** Past frames $\tau \in \{1, 2, 4\}$.
- **Policy architecture:** Multilayer perceptron (MLP), recurrent LSTM, and Transformer encoder (More details about this is are in Appendix.A.4).

7.4.1 Effect of Heliostat Distance

Distance (m)	Extrapolated sun	Interpolated sun
15	0.3462 mrad	0.4060 mrad
150	0.3467 mrad	0.3477 mrad
1500	0.3462 mrad	0.3472 mrad

Table 3: Average minimum angular error (mrad) at different heliostat distances for extrapolated and interpolated sun positions. Values are averaged over multiple runs.[lower is better]

Across all distances(Figure. 15), the policies converge to a minimum angular error (Table.3) of approximately 0.346 mrad, which represents the numerical floor imposed by our implementation of the angular error function. This indicates that, once well-trained, the agent is able to reach the same effective accuracy regardless of heliostat distance.

Nevertheless, heliostat distance has a direct physical influence on the system. Because the same angular error translates into a larger positional offset at longer distances, misaligned rays at 1500m can spill over the receiver boundary or miss it entirely, whereas the same angular deviation at 15 m remains well contained. Thus, while the measured angular error metric saturates at the same level, the risk of flux spillage and efficiency loss increases substantially with distance. This effect is reflected in the boundary loss and in qualitative flux patterns, which become more sensitive to small deviations when the heliostat is placed farther away.

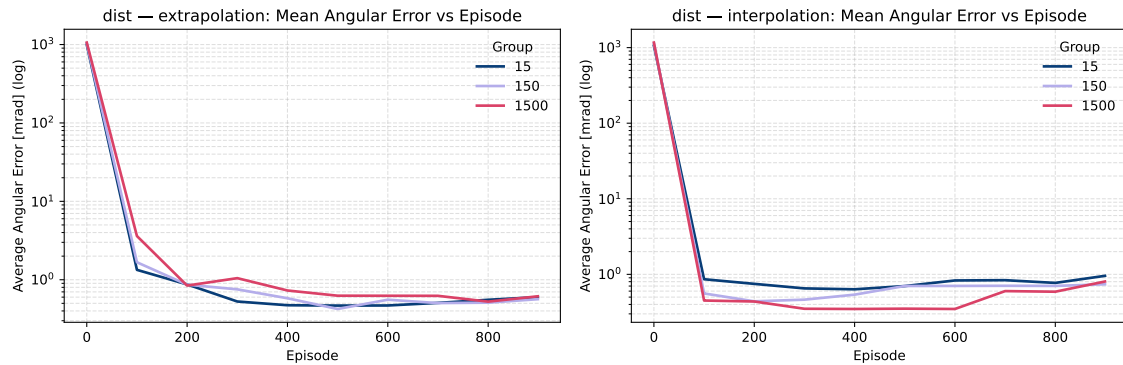


Figure 15: Average test alignment error over training episodes, for heliostate distance = 15m, 150m and 1500m. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position

7.4.2 Effect of Induced Error Scale

We next analyze robustness with respect to different levels of induced mirror orientation error. A baseline misalignment of 5 mrad is already considered a poor calibration scenario, since modern optical calibration techniques can typically achieve lower errors. However, the long-term goal of our framework is to enable calibration-free heliostat operation, so we evaluate at larger error scales of 10, 25, and 45 mrad. Table 7.4.2 summarizes the minimum angular errors achieved under extrapolated and interpolated sun positions.

Induced error (mrad)	Extrapolated sun	Interpolated sun
5	0.3462 mrad	0.3472 mrad
10	0.3477 mrad	0.3472 mrad
25	0.8125 mrad	0.4663 mrad
45	11.4414 mrad	3.9158 mrad

For both extrapolated and interpolated settings, the policies converge to the same numerical floor of ≈ 0.346 mrad for 5 and 10 mrad induced errors, indicating that the framework can fully compensate for these levels of miscalibration (Table. 7.4.2 and Figure.16).

The performance degradation only becomes evident at higher error scales: at 25 mrad the agent still recovers to below 1 mrad average misalignment in both cases, which is considered a good operating regime in the literature. At 45 mrad, however, performance drops sharply, with average errors of 3.9–11.4 mrad, confirming that extreme misalignments remain beyond the corrective capacity of the current policy.

7.4.3 Effect of Fine Adjustment

We finally evaluate the role of the fine-adjustment layer, comparing three modes:

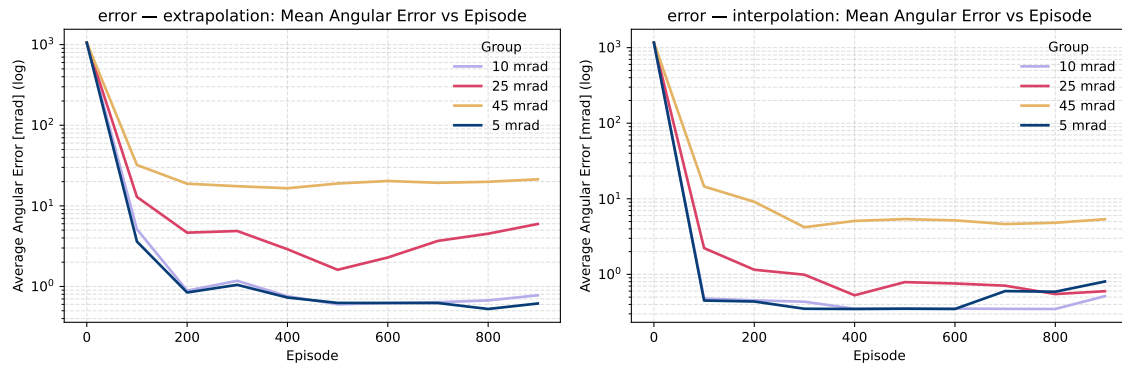


Figure 16: Average test alignment error over training episodes, for a max induced alignment error of 5mrad, 10mrad, 25mrad, and 45mrad: Performance over extrapolated sun position, Right: Performance over interpolated sun position. The performances after inducing a max of 5mrads and 10mrads remain mostly similar

1. **Always:** Fine adjustment is active during both training and testing.
2. **Off:** Fine adjustment is disabled throughout.
3. **Test-only:** Fine adjustment is used only during testing (not training).

When training with fine adjustment enabled, the first 50 episodes are always run without it to ensure that the agent learns a coarse alignment strategy before relying on fine-tuning.

Setting	Extrapolated sun	Interpolated sun
Always	0.3462 mrad	0.3472 mrad
Off	6.5210 mrad	8.8244 mrad
Test-only	0.3472 mrad	0.3457 mrad

Table 4: Achieved minimum angular error (mrad) with different fine-adjustment settings.[lower is better]

Table 5: Average relative time to best loss (min) with different fine-adjustment settings.

Setting	Extrapolated sun	Interpolated sun
Always	79.23	38.49
Off	48.12	25.31
Test-only	41.29	24.74

These results confirm that fine adjustment is a **key component** of our framework: without it, the agent remains several milliradians off-target (6–9 mrad), whereas enabling fine adjustment consistently reduces errors to the sub-1 mrad range (Table.4 and Figure. 17), which is considered acceptable in the literature.

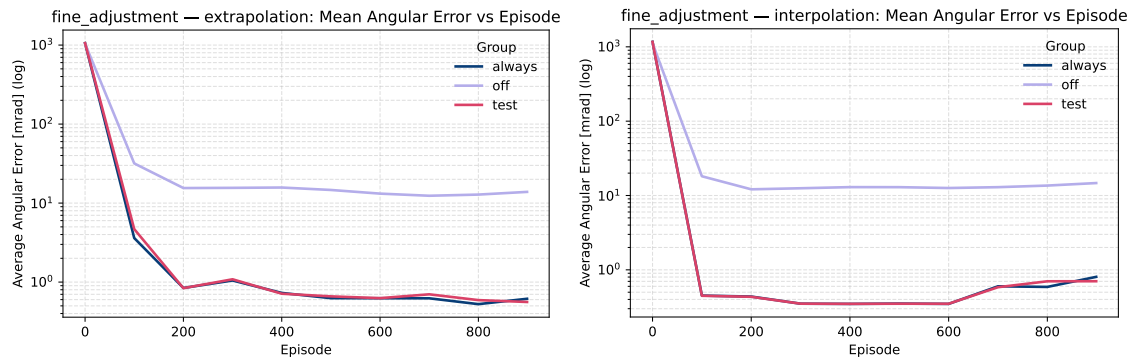


Figure 17: Average test alignment error over training episodes. This plot shows performance over three settings, one where fine alignment is always enabled, one where it is always disabled, and one when it’s only enabled during test time. Left Performance over extrapolated sun position, Right: Performance over interpolated sun position

Interestingly, enabling fine adjustment during training does not improve the final accuracy compared to using it only at test time: both settings converge to the same error floor of ≈ 0.346 mrad. However, training with fine adjustment comes at the cost of longer convergence times (Table. 5), since the inner optimization loop is repeatedly invoked during learning. Therefore, while test-time fine adjustment is essential, training-time fine adjustment is not strictly necessary for achieving high accuracy and may be avoided to accelerate training.

7.4.4 Effect of Fine-Adjustment Step Count

We vary the number of inner optimization steps performed in the fine-adjustment block, comparing $K = 5$ and $K = 10$.

Fine-adj. steps	Extrapolated sun	Interpolated sun
$K = 5$	0.3533 mrad	0.3637 mrad
$K = 10$	0.3462 mrad	0.3472 mrad

Table 6: Minimum angular error (mrad) with different numbers of fine-adjustment steps.[lower is better]

Fine-adj. steps	Extrapolated sun	Interpolated sun
$K = 5$	47.05	50.50
$K = 10$	79.23	38.49

Table 7: Average relative time to best loss (minutes) with different numbers of fine-adjustment steps.

Takeaways. Increasing the fine-adjustment iterations from 5 to 10 improves the achieved minimum error in both settings:

- Extrapolated: 0.3533 mrad \rightarrow 0.3462 mrad (improvement of ≈ 0.007 mrad).
- Interpolated: 0.3637 mrad \rightarrow 0.3472 mrad (improvement of ≈ 0.017 mrad).

The average relative time to reach best loss remains comparable across the two settings, suggesting that doubling the number of inner steps primarily refines the final alignment rather than increasing convergence cost. Thus, allocating 10 fine-adjustment steps yields consistently lower residual error at almost no penalty in runtime (Table.6 and Table.7), and is therefore preferable in practice.

7.4.5 Effect of Training Horizon

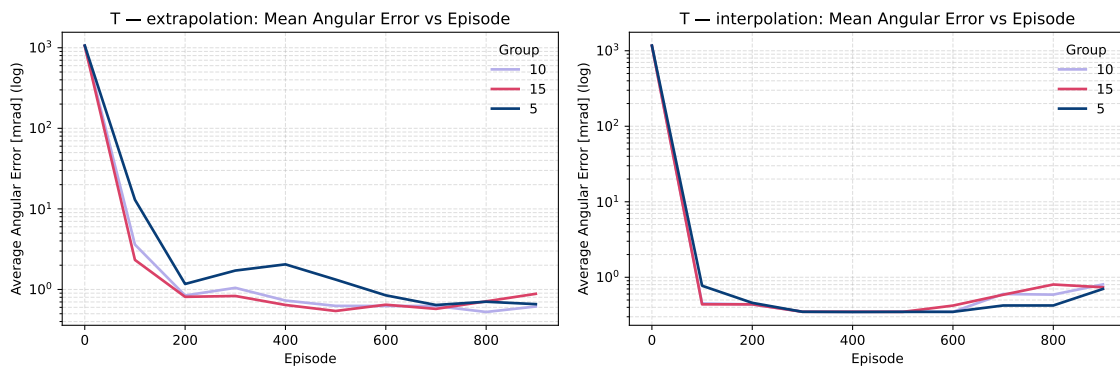


Figure 18: Average test alignment error over training episodes. This plot shows performance over three settings for our training’s horizon-length, namely $T=5$, $T=10$ and $T=15$. Left Performance over extrapolated sun position, Right: Performance over interpolated sun position

We now evaluate the influence of the training horizon T on performance (Figure.18 and Table.8). All policies are evaluated for 30 rollout steps, but trained with horizons of $T = 5$, $T = 10$, and $T = 15$.

Training horizon T	Extrapolated sun	Interpolated sun
5	0.4971 mrad	0.3453 mrad
10	0.3462 mrad	0.3472 mrad
15	0.3462 mrad	0.3467 mrad

Table 8: Minimum angular error (mrad) [lower is better] with different training horizons.

Training horizon T	Extrapolated sun	Interpolated sun
5	28.32	19.57
10	79.23	38.49
15	93.81	55.82

Table 9: Average relative time to best loss (minutes) with different training horizons.

The results highlight a tradeoff between training time and accuracy. Short horizons ($T = 5$) converge quickly but fail to reach the 0.346 mrad error floor in the extrapolated case, plateauing around 0.50 mrad. Longer horizons ($T = 10$ and $T = 15$) reliably reach the error floor, but require significantly more time to converge (Table.9).

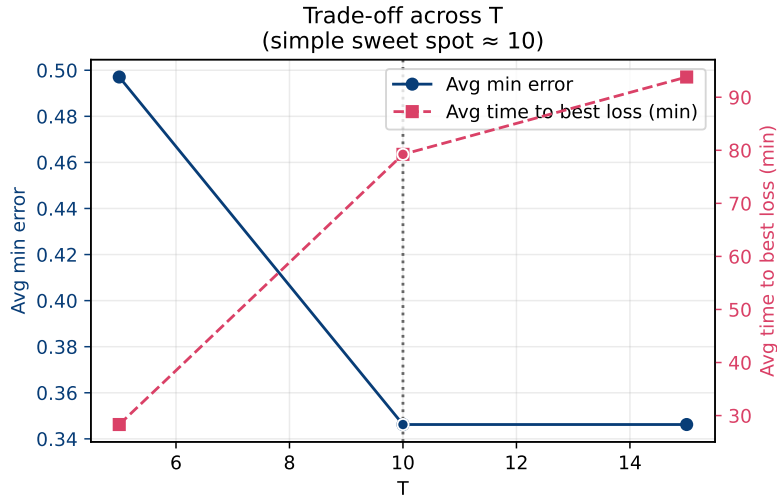


Figure 19: Tradeoff Plot between average min-error and average time to best loss, evaluated over all T. The plots cross over at T

By visually inspecting the tradeoff plot, Figure.19, we observed that the accuracy–time curves cross around $T = 8$, suggesting that this would have been the optimal compromise. However, due to time constraints and the high computational cost of re-running all experiments, we adopted $T = 10$ as the default training horizon in our framework.

7.4.6 Effect of Sequence Length (k past flux density maps)

We vary the number of past frames $\tau \in \{1, 2, 4\}$ fed to the policy, while keeping all other defaults fixed. Tables 10 and 11 report the achieved minimum angular error and the average time to reach it.

Takeaways. Across both extrapolated and interpolated settings, performance is essentially *invariant* to the history length: all configurations converge to the same ≈ 0.346 – 0.347 mrad error floor with modest variation in time-to-best. This indicates

k frames	Extrapolated sun	Interpolated sun
1	0.3462 mrad	0.3457 mrad
2	0.3462 mrad	0.3472 mrad
4	0.3472 mrad	0.3453 mrad

Table 10: Minimum angular error (mrad) vs. sequence length k . [lower is better]Table 11: Average relative time to best loss (minutes) vs. k .

k frames	Extrapolated sun	Interpolated sun
1	58.40	22.29
2	79.23	38.49
4	63.92	25.35

our augmented Transformer architecture can operate effectively with *just the current frame* ($\tau = 1$). Mechanistically, two factors explain this: (i) the rollout maintains a short image buffer but always includes the latest frame (so $\tau = 1$ already suffices to inform the step) and (ii) the Transformer path carries a latent state forward by adding the previous step’s feature as a residual (h_t) to the current token’s representation, thereby propagating information without needing long explicit histories. In code, the image history is packed into a sequence and the Transformer’s last-token features are residually combined with the prior state to yield h_t , which is then reused at the next step.

Implication. Using $\tau = 1$ lowers compute and memory while preserving accuracy. This is useful for real-time deployment. The learned carry-over h_t (plus the incremental update of predicted normals across steps) supplies the necessary temporal continuity even from single-frame inputs.

7.4.7 Effect of Model Architecture

We compare three policy backbones, namely, MLP, LSTM, and Transformer, under identical training and evaluation protocols. Table 12 reports the achieved minimum angular errors for both extrapolated and interpolated sun positions.

Architecture	Extrapolated sun	Interpolated sun
MLP	0.4086 mrad	0.3457 mrad
LSTM	0.3467 mrad	0.5678 mrad
Transformer	0.3462 mrad	0.3472 mrad

Table 12: Minimum angular error (mrad) by architecture. [lower is better]

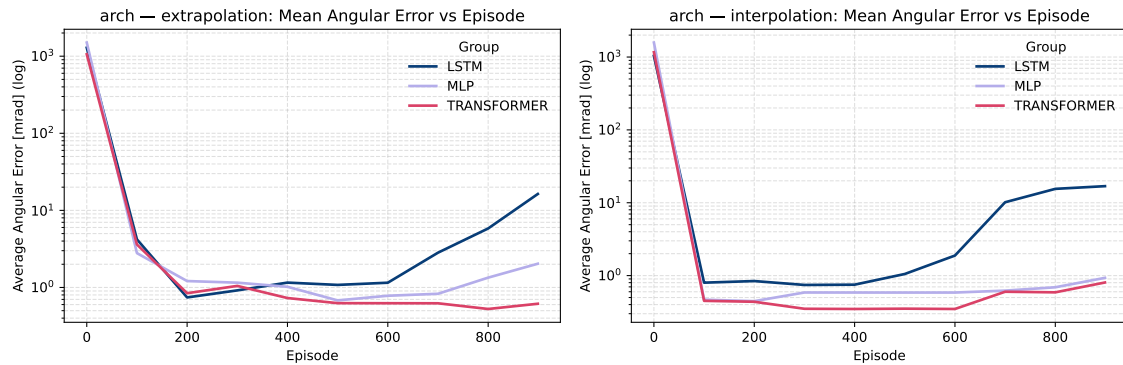


Figure 20: Average test alignment error over training episodes, for Transformers, LSTMs and MLP backbones. Left : Performance over extrapolated sun position, Right: Performance over interpolated sun position

Takeaways.

- All architectures reach sub-1 mrad in at least one setting, confirming that our training pipeline and fine-adjustment block can drive errors into the desirable regime.
- In the **extrapolated** case, LSTM and Transformer are essentially tied at the numerical floor (≈ 0.346 mrad), while MLP lags behind.
- In the **interpolated** case, Transformer and MLP are similar (both at the floor), whereas LSTM degrades (≈ 0.568 mrad).

Why we choose Transformer. The Transformer encoder provides (i) strong performance in both regimes and (ii) better *training-time generalization*: our logs show the test losses for MLP and LSTM begin to rise (overfitting) much earlier, i.e., around episode ~ 200 and ~ 400 for the interpolated setting, and ~ 500 and ~ 400 for the extrapolated setting. However, the Transformer resists overfitting until much later (Figure.20) (around episode ~ 600 interpolated and not until ~ 900 extrapolated, which is our maximum training step). These observations align with the architectural details in our code: the Transformer path aggregates frame features via attention and carries a compact latent forward (h_t) by residual addition at the last token, which stabilizes learning without requiring explicit recurrence state management.

8 Discussion

In this chapter, we summarize the empirical findings with respect to the objectives of the thesis and the assumptions of our framework, highlighting practical implications for closed-loop heliostat control and outlining the possible limitations for real-world deployment. We conclude with design recommendations summarized from our results.

8.1 Robust closed-loop control

Across varying distances and induced error ranges, our proposed model-based RL pipeline consistently drives the angular misalignment of the heliostat into the sub-1 mrad range, approaching the numerical floor of our alignment metric for both the interpolated and extrapolated sun directions for resting. This indicates that (i) our differentiable optics environment provides gradients which are sufficient for heliostat control, (ii) the learned policy generalizes well to sun positions not seen during training.

8.2 Role of the fine-adjustment layer

A key observation is that the test-time *fine adjustment* layer is decisive, i.e., disabling it raises the residual error by several mrads, whereas enabling it reduces misalignment error to the numerical floor of our alignment metric. This indicates a split of responsibilities: the APG policy learns fast, coarse alignment, while a low cost, inner-loop exploits image-based losses and the local geometry of the differentiable simulator to finish the alignment task. Training with fine adjustment enabled, however, did not improve final accuracy compared to test-only usage, and only increased convergence time, suggesting a strategy where we *train without* the fine alignment and *evaluate with* the fine alignment.

8.3 Horizon and temporal context

Training with a horizon of $T = 10$ yields better performance than training with $T = 5$, but increasing the horizon to $T = 15$ brings no further improvement in terms of performance. This suggests that the performance gains stagnate at around $T = 10$, while the computation cost keeps increasing, revealing a tractable performance-time trade-off. Surprisingly, performance is nearly invariant to the choice of length of the image history ($k \in \{1, 2, 4\}$). The transformer’s hidden-state carry over, together with the residual action update, appear to be sufficient for temporal credit assignment. Suggesting that single-frame inputs can be viable for real-time use.

8.4 Choice of Architecture

While all tested backbones can succeed in bringing the alignment error down to a sub-1 mrad range, the Transformer provided the most stable performance across the interpolated and extrapolated sun directions, with relatively delayed overfitting compared to MLP and LSTM. This supports using an attention-based temporal encoder when the observations are in the form of a compact geometric descriptor (CoM in our framework), along with the previous actions and the sun direction.

8.5 Distance and induced-error sweeps

Our RL pipeline exhibits the ability of the lower induced angular errors converge to the same numerical floor across the distances. The physical consequences of a fixed angular error increases greatly with distance due to the optical error arm. Hence, even small residual errors have a magnified effect. Scaling the induced error reveals a clear operating envelope: the policy compensates fully up to 10 mrad, remains robust at 25 mrad (sub-1 mrad averages), and degrades at 45 mrad. This sets a useful baseline for specifying expected calibration drift tolerances for closed-loop heliostat control.

8.6 Implications for CSP operations

8.6.1 Latency and compute budget

Since the learned policy executes in a single forward pass, and the test-time fine adjustment only adds a small inner loop, our pipeline would be quite effective for real time deployment. For embedded deployment: (i) an MLP can be used in place of a transformer to favour a smaller model size with minimal performance loss, (ii) latency can be further optimized by setting the number of fine-adjustment steps and the step size, with well crafted accuracy-latency trade-offs.

8.6.2 Shifting from per-heliostat sensors to aggregate signals

Our results show that policies trained on the receiver-plane's flux density map can correct heliostat misalignments without the need for per-heliostat on-board sensor. In practice, this could lower field instrumentation costs, given that the receiver sensing pipeline (camera or radiometer) is reliable, synchronized with the heliostat, and robust to glare, weather and soiling.

8.7 Why our pipeline works

Three modeling decisions appear to be the key for the functionality of our model:

1. **Residual, normalized actions:** predicting increments in action and renormalizing yields a well-conditioned Jacobian $\partial a_t / \partial v_t = \|v_t\|^{-1}(I - aa^\top)$, which regularizes gradient flow and discourages large, destabilizing jumps.
2. **Two-stage control:** coarse policy lets us exploit learned global priors and the fine-adjustment lets us exploit the local curvature (via differentiation), combining sample efficiency with final-accuracy guarantees.
3. **Center-of-Mass (CoM) features:** The CoM layer reduces each flux density map to compact geometric descriptors (flux centroids). This provides the policy with stable, low-dimensional, computationally fast inputs that emphasize the geometry of misalignment, improving efficiency.

8.8 Limitations

8.8.1 Single-heliostat model

All our experiments use a single heliostat. Multi heliostat interactions (including occlusion, inter-reflections, shadowing, tower blockage) is not modeled in our framework. Policies might require retraining or constraints when many heliostats interact and receive feedback through a shared flux density map.

8.8.2 Simplified optics and error model

We ignore the effects of facet-level surface errors, and speckle and restrict the intrinsic misalignments to rotations about two axes. Real fields exhibit wind gusts, wear and tear, and other time-varying biases. These omissions risk optimistic accuracy.

8.8.3 Episode stationarity

In our framework Sun and error remain fixed within an episode. Real-world operation includes slow drifts additional exogenous disturbances (clouds). Although the controller acts repeatedly, it has not been evaluated on realistic distribution shift within single episodes, which may degrade performance.

8.8.4 Metric floor

The angular metric saturates near a numerical floor. Hence, conclusions about “equality” between settings should be revisited with a measurement function which has a floor at 0 mrad.

8.9 Design recommendations

Based on the results of our experiments and practical considerations, we recommend:

- **Backbone:** Transformer encoder with single-frame CoM features, residual state carry over, and residual action updates.
- **Training:** A horizon $T \approx 8 - 10$ as a good performance v/s training time compromise. Train *without* fine adjustment.
- **Inference:** Enable test-time fine adjustment with ~ 10 inner steps.

9 Conclusion

The findings of this thesis affirm that reinforcement learning framework, combined with a differentiable heliostat-optics simulator, serves as a functional proof of concept, our proposed system establishes a practical baseline for future RL-based approaches for real-time, closed loop heliostat control.

To summarize our contributions, we developed (i) a **end-to-end differentiable heliostat optics simulator** can provide analytic gradients of flux densities, alignment error, and geometric properties of our heliostat fields, which enables direct integration with gradient-based optimization algorithms. (ii) a **model-based RL pipeline** that trains a residual, normalized-action policy network using APG and combines it with a differentiable fine-adjustment layer, achieving alignment errors well below 1mrad (the current standard for good calibration techniques) across a varying range of heliostat distances, induced error magnitude and unseen sun position.

Together these components deliver a controller that executes in real time, requires no per-heliostat sensors, and outperforms model-free RL techniques and model predictive control. In practical terms, heliostat control primarily runs on open-loop control, requiring a calibration time of about an hour for a single heliostat, and several weeks for a few months. Our closed-loop framework operating on a single flux density image, lays the groundwork for calibration-free, real-time heliostat control. Additionally, our simplified and parallelizable simulator provides a platform for pre-training RL or other ML based control strategies, which can later be fine-tuned on more accurate raytracing simulator to train more realistic controllers.

10 Outlook

The introduced framework within this thesis for model-based RL control enables further research that could not be investigated due to time constraints. Outlooks for such further investigations have been collected throughout the duration of this thesis and are summarized in the sections below

10.1 Changing Sun position over time

In this work, we assumed the sun direction to be fixed within an episode. However, the sun direction changes continuously, introducing a slow exogenous drift during operations. This also changes the ideal orientation vector over time. While our agent seems to generalize to any given sun position, the sun position being fixed over the entire duration of the episode implies that the agent has to guess the correct normal vector once and not change it. Hence further investigation is needed into the agent's capacity to adapt to a non-stationary sun position.

10.2 Multiple Heliostat Control

Our current differentiable simulator already allows us to have multiple heliostats and generate flux density map for the combined field, in parallel across multiple simulations. However due to sanity-checks failures and the time required to tune and fix bugs in the heliostat optics simulator code, we were largely constrained to working with a single heliostat for most of the duration of the thesis. This leaves a huge scope of adapting the current RL framework to multiple heliostats in parallel. However, since the current CoM block works with the overall flux density map's mask, and provides only one centroid, multiple heliostat control would require replacing the CoM block with a different feature extractor, e.g., Convolutional layers.

10.3 Fine Tuning in a higher fidelity raytracing simulations

Our simulator, while being fast and working with the basic laws of optics, does not provide realistic training scenarios such as surface deformations, weather conditions, blocking and shading from other heliostats, for our agent to be directly adapted to a real-world Solar Tower Plant. However there are two possible approaches to doing so, (i) to pre-train inside our simulation and to fine-tune inside a higher fidelity simulation which better match real world conditions, (ii) use our simulation to train the coarse alignment policy block and replace the model inside the fine-alignment block with a higher fidelity differentiable simulator. Approach (ii) would provide a way for the current simulator to be used in a more realistic scenario right out of the box with no further training required.

References

- [1] BENGIO, Yoshua ; LOURADOUR, Jérôme ; COLLOBERT, Ronan ; WESTON, Jason: Curriculum Learning. In: *ICML*, 2009
- [2] CARBALLO, J.A. ; BONILLA, J. ; CRUZ, N.C. ; FERNÁNDEZ-RECHE, J. ; ÁLVAREZ, J.D. ; AVILA-MARIN, A. ; BERENGUEL, M.: Reinforcement learning for heliostat aiming: Improving the performance of Solar Tower plants. In: *Applied Energy* 377 (2025), 124574. <http://dx.doi.org/https://doi.org/10.1016/j.apenergy.2024.124574>. – DOI <https://doi.org/10.1016/j.apenergy.2024.124574>. – ISSN 0306–2619
- [3] DANIELSSON, P-E.: Euclidean Distance Mapping. In: *Computer Graphics and Image Processing* 14 (1980), 227–248. <https://www.sciencedirect.com/science/article/pii/0146664X80900544>
- [4] GANGOPADHYAY, Anasuya ; SESHADRI, Ashwin K. ; PATIL, Balachandra: Wind-solar-storage trade-offs in a decarbonizing electricity system. In: *Applied Energy* 353 (2024), 121994. <http://dx.doi.org/https://doi.org/10.1016/j.apenergy.2023.121994>. – DOI <https://doi.org/10.1016/j.apenergy.2023.121994>. – ISSN 0306–2619
- [5] HAMDAN, Mustapha ; SEBASTIA-SAEZ, Daniel ; HAMDAN, Malak ; ARELLANO-GARCIA, Harvey: CFD Analysis of the Use of Desert Sand as Thermal Energy Storage Medium in a Solar Powered Fluidised Bed Harvesting Unit. Version: 2020. <http://dx.doi.org/https://doi.org/10.1016/B978-0-12-823377-1.50059-8>. In: PIERUCCI, Sauro (Hrsg.) ; MANENTI, Flavio (Hrsg.) ; BOZZANO, Giulia L. (Hrsg.) ; MANCA, Davide (Hrsg.): *30th European Symposium on Computer Aided Process Engineering* Bd. 48. Elsevier, 2020. – DOI <https://doi.org/10.1016/B978-0-12-823377-1.50059-8>. – ISSN 1570–7946, 349-354
- [6] HASSAN, Qusay ; ALGBURI, Sameer ; SAMEEN, Aws Z. ; SALMAN, Hayder M. ; JASZCZUR, Marek: A review of hybrid renewable energy systems: Solar and wind-powered solutions: Challenges, opportunities, and policy implications. In: *Results in Engineering* 20 (2023), 101621. <http://dx.doi.org/https://doi.org/10.1016/j.rineng.2023.101621>. – DOI <https://doi.org/10.1016/j.rineng.2023.101621>. – ISSN 2590–1230
- [7] HE, Kaiming ; ZHANG, Xiangyu ; REN, Shaoqing ; SUN, Jian: Deep Residual Learning for Image Recognition. In: *CVPR*, 2016
- [8] HOCHREITER, Sepp ; SCHMIDHUBER, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), Nr. 8, S. 1735–1780
- [9] HONG, Seokju u. a.: Deep Matching Prior: Test-Time Optimization for Dense Correspondence. In: *ICCV*, 2021

-
- [10] JOHANNINK, Tobias ; BAH, Shikhar ; NAIR, Ashvin ; LUO, Jianlan ; KUMAR, Avinash ; LOSKYLL, Matthias ; APARICIO OJEA, Juan ; SOLOWJOW, Eugen ; LEVINE, Sergey: Residual Reinforcement Learning for Robot Control. In: *ICRA*, 2019
- [11] JONES, S A. ; STONE, K W.: Analysis of Strategies to Improve Heliostat Tracking at Solar Two Sandia National Laboratories, Albuquerque, NM, and Livermore, CA, 1999
- [12] KHAN, Jibrán ; ARSALAN, Mudassar H.: Solar power technologies for sustainable electricity generation – A review. In: *Renewable and Sustainable Energy Reviews* 55 (2016), 414-425. <http://dx.doi.org/https://doi.org/10.1016/j.rser.2015.10.135>. – DOI <https://doi.org/10.1016/j.rser.2015.10.135>. – ISSN 1364–0321
- [13] KONDA, Vijay R. ; TSITSIKLIS, John N.: Actor-Critic Algorithms. In: SOLLA, Sara A. (Hrsg.) ; LEEN, Todd K. (Hrsg.) ; MÜLLER, Klaus-Robert (Hrsg.): *Advances in Neural Information Processing Systems 12*, MIT Press, 2000, 1008–1014
- [14] KRAUTH, Julian J. ; HAPPICH, Christoph ; ALGNER, Niels ; BRODA, Rafal ; KÄMPGEN, Andreas ; SCHNERRING, Alexander ; ULMER, Steffen ; RÖGER, Marc: Heliopoint – A Fast Airborne Calibration Method for Heliostat Fields. In: *Journal of Solar Energy Engineering* 146 (2024), 07, Nr. 6, 061005. <http://dx.doi.org/10.1115/1.4065868>. – DOI 10.1115/1.4065868. – ISSN 0199–6231
- [15] LEIBAUER, Moritz: *Enhancing Heliostat Calibration in Solar Tower Power Plants: A Novel Dataset Evaluation Metric and Hybrid Kinematic Modelling Technique*, RWTH Aachen, Diplomarbeit, May 2023. <https://elib.dlr.de/202957/>
- [16] MADHLOPA, Amos ; OKOROIGWE, Edmund: Solar Gas Turbine Systems. Version: 2017. <http://dx.doi.org/https://doi.org/10.1016/B978-0-12-409548-9.10150-2>. In: ABRAHAM, Martin A. (Hrsg.): *Encyclopedia of Sustainable Technologies*. Oxford : Elsevier, 2017. – DOI <https://doi.org/10.1016/B978-0-12-409548-9.10150-2>. – ISBN 978-0-12-804792-7, 377-388
- [17] MALAN, K. ; GAUCHÉ, P.: Model based Open-loop Correction of Heliostat Tracking Errors. In: *Energy Procedia* 49 (2014), 2118-2124. <http://dx.doi.org/https://doi.org/10.1016/j.egypro.2014.03.224>. – DOI <https://doi.org/10.1016/j.egypro.2014.03.224>. – ISSN 1876–6102. – Proceedings of the SolarPACES 2013 International Conference
- [18] MORALES PEDRAZA, Jorge: Chapter 3 - Solar energy for electricity generation. Version: 2022. <http://dx.doi.org/https://doi.org/10.1016/B978-0-12-823440-2.00006-8>. In: MORALES PEDRAZA, Jorge (Hrsg.): *Non-Conventional Energy in North America*. Elsevier, 2022. – DOI <https://doi.org/10.1016/B978-0-12-823440-2.00006-8>. – ISBN 978-0-12-823440-2, 137-174

- [19] OKOROIGWE, Edmund ; MADHLOPA, Amos: An integrated combined cycle system driven by a solar tower: A review. In: *Renewable and Sustainable Energy Reviews* 57 (2016), 337-350. <http://dx.doi.org/https://doi.org/10.1016/j.rser.2015.12.092>. – DOI <https://doi.org/10.1016/j.rser.2015.12.092>. – ISSN 1364–0321
- [20] OLAUSON, Jon ; AYOB, Mohd N. ; BERGKVIST, Mikael ; CARPMAN, Nicole ; CASTELLUCCI, Valeria ; GOUDE, Anders ; LINGFORS, David ; WATERS, Rafael ; WIDÉN, Joakim: Net load variability in Nordic countries with a highly or fully renewable power system. In: *Nature Energy* 1 (2016), Nr. 12, 16175. <http://dx.doi.org/10.1038/nenergy.2016.175>. – DOI 10.1038/nenergy.2016.175
- [21] PARGMANN, Max ; EBERT, Jan ; GÖTZ, Markus ; MALDONADO QUINTO, Daniel ; PITZ-PAAL, Robert ; KESSELHEIM, Stefan: Automatic heliostat learning for in situ concentrating solar power plant metrology with differentiable ray tracing. In: *Nature Communications* 15 (2024), Nr. 6997. <http://dx.doi.org/10.1038/s41467-024-51019-z>. – DOI 10.1038/s41467-024-51019-z
- [22] PASZKE, Adam ; GROSS, Sam ; CHINTALA, Soumith ; CHANAN, Gregory ; YANG, Edward ; DEVITO, Zachary ; LIN, Zeming ; DESMAISON, Alban ; ANTIGA, Luca ; LERER, Adam: Automatic differentiation in PyTorch. (2017)
- [23] RUMELHART, David E. ; HINTON, Geoffrey E. ; WILLIAMS, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), S. 533–536. <http://dx.doi.org/10.1038/323533a0>. – DOI 10.1038/323533a0
- [24] SATTLER, Johannes C. ; RÖGER, Marc ; SCHWARZBÖZL, Peter ; BUCK, Reiner ; MACKE, Ansgar ; RAEDER, Christian ; GÖTTSCHE, Joachim: Review of heliostat calibration and tracking control methods. In: *Solar Energy* 207 (2020), 110-132. <http://dx.doi.org/https://doi.org/10.1016/j.solener.2020.06.030>. – DOI <https://doi.org/10.1016/j.solener.2020.06.030>. – ISSN 0038–092X
- [25] SCHNELL, Patrick ; THUERREY, Nils: Stabilizing Backpropagation Through Time to Learn Complex Physics. In: *The Twelfth International Conference on Learning Representations, 2024*
- [26] SCHULMAN, John ; LEVINE, Sergey ; MORITZ, Philipp ; JORDAN, Michael ; ABBEEL, Pieter: Trust region policy optimization. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, JMLR.org, 2015 (ICML'15), S. 1889–1897*
- [27] SCHULMAN, John ; WOLSKI, Filip ; DHARIWAL, Prafulla ; RADFORD, Alec ; KLIMOV, Oleg: Proximal Policy Optimization Algorithms. In: *arXiv:1707.06347* (2017). <https://arxiv.org/pdf/1707.06347>
- [28] SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement Learning: An Introduction*. Cambridge, MA : MIT Press, 2018. – ISBN 978–0–262–03924–6

-
- [29] VASWANI, Ashish ; SHAZEER, Noam ; PARMAR, Niki ; USZKOREIT, Jakob ; JONES, Llion ; GOMEZ, Aidan N. ; KAISER, Lukasz ; POLOSUKHIN, Illia: Attention Is All You Need. In: *Advances in Neural Information Processing Systems (NeurIPS)*, 2017
- [30] WERBOS, P.J.: Backpropagation through time: what it does and how to do it. In: *Proceedings of the IEEE* 78 (1990), Nr. 10, S. 1550–1560. <http://dx.doi.org/10.1109/5.58337>. – DOI 10.1109/5.58337
- [31] WIEDEMANN, Nina ; WÜEST, Valentin ; LOQUERCIO, Antonio ; MÜLLER, Matthias ; FLOREANO, Dario ; SCARAMUZZA, Davide: Training Efficient Controllers via Analytic Policy Gradient. In: *2023 International Conference on Robotics and Automation (ICRA) IEEE*, 2023
- [32] WILLIAMS, Ronald J.: A class of gradient-estimating algorithms for reinforcement learning in neural networks. In: *Proceedings of the IEEE First International Conference on Neural Networks*. San Diego, CA, 1987, S. II–601–II–608
- [33] WILLIAMS, Ronald J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: *Machine Learning* 8 (1992), Nr. 3-4, S. 229–256
- [34] XU, Jie ; MAKОВIYCHUK, Viktor ; NARANG, Yashraj ; RAMOS, Fabio ; MATUSIK, Wojciech ; GARG, Animesh ; MACKLIN, Miles: Accelerated Policy Learning with Parallel Differentiable Simulation. In: *International Conference on Learning Representations*, 2021

A Appendix

A.1 Derivative of h_t w.r.t. θ_π (additive carry)

Recall the augmented state update

$$h_t = z_t(\theta_\pi) + h_{t-1}, \quad h_0 = \mathbf{0},$$

Since z_t does not depend on h_{t-1} . By the chain rule,

$$\frac{\partial h_t}{\partial \theta_\pi} = \frac{\partial z_t}{\partial \theta_\pi} + \frac{\partial h_{t-1}}{\partial \theta_\pi}.$$

Unrolling from h_0 yields the closed form

$$\boxed{\frac{\partial h_t}{\partial \theta_\pi} = \sum_{k=1}^t \frac{\partial z_k}{\partial \theta_\pi}}.$$

A.2 ReLU activation function

$$\text{ReLU}(x) = \begin{cases} 0, & x \leq 0, \\ x, & x > 0. \end{cases}$$

A.3 Dirac delta distribution

The Dirac delta $\delta(x)$ is a generalized function (distribution) that is zero everywhere except at the origin and integrates to one. Formally, for any smooth test function ϕ ,

$$\int_{-\infty}^{\infty} \delta(x) \phi(x) dx = \phi(0).$$

More generally, $\delta(x - a)$ represents a unit point mass at $x = a$ and satisfies

$$\int_{-\infty}^{\infty} \delta(x - a) \phi(x) dx = \phi(a).$$

In probability terms it is a distribution concentrated at a single point.

Here $\delta_\alpha(\beta)$ denotes a Dirac measure centered at α , indicating that the variable β equals α with probability one.

A.4 Policy Network

- **Input**
 - Flux-image sequence $\mathbf{I} \in \mathbb{R}^{B \times T \times C \times H \times W}$
 - Auxiliary vector $\mathbf{x}_{\text{aux}} \in \mathbb{R}^{B \times (3+3N)}$
- **Per-frame Encoder (COMEncoder)**
 - CenterOfMass2D \rightarrow 2-D coordinates
 - Linear ($2 \rightarrow d_{\text{enc}}$) — *depth: 1 layer*
 - Dropout (p)
 - GELU
 - Default: $d_{\text{enc}} = 128$, $p = 0.3$ (configurable)
- **Temporal / Sequence Block** (selected via `-architecture`)
 - *LSTM*: single-layer LSTM, hidden size d_{lstn} (default 128) — *depth: 1 recurrent layer*
 - *Transformer*: L encoder layers (default $L = 2$), each with H attention heads (default $H = 8$), model dimension d_{enc} — *depth: L stacked layers*
 - *MLP*: uses last-frame encoding only — *no additional depth*
- **Feature Fusion** Concatenate the temporal feature with \mathbf{x}_{aux} .
- **Prediction Head**
 - LayerNorm — *depth: 1*
 - Linear ($d_{\text{feat}} + d_{\text{aux}} \rightarrow 256$) — *depth: 1*
 - Dropout (p)
 - GELU
 - Linear ($256 \rightarrow 3 \times N_{\text{heliostats}}$) — *depth: 1*
- **Output** Surface-normal predictions $\hat{\mathbf{n}} \in \mathbb{R}^{B \times N_{\text{heliostats}} \times 3}$.

Declaration

I, _____, student registration number: _____, hereby confirm that I completed the submitted work independently and without the unauthorized assistance of third parties and without the use of undisclosed and, in particular, unauthorized aids. This work has not been previously submitted in its current form or in a similar form to any other examination authorities and has not been accepted as part of an examination by any other examination authority.

Where the wording has been taken from other people's work or ideas, this has been properly acknowledged and referenced. This also applies to drawings, sketches, diagrams and sources from the Internet.

In particular, I am aware that the use of artificial intelligence is forbidden unless its use as an aid has been expressly permitted by the examiner. This applies in particular to chatbots (especially ChatGPT) and such programs in general that can complete the tasks of the examination or parts thereof on my behalf.

Furthermore, I am aware that working with others in one room or by means of social media represents the unauthorized assistance of third parties within the above meaning, if group work is not expressly permitted. Each exchange of information with others during the examination, with the exception of examiners and invigilators, about the structure or contents of the examination or any other information such as sources is not permitted. The same applies to attempts to do so.

Any infringements of the above rules constitute fraud or attempted fraud and shall lead to the examination being graded "fail" ("nicht bestanden").

Place, date

Signature