
Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology and Methodology

# Autonomous Glider Control using Reinforcement Learning for Thermal Soaring in Realistic, Turbulent Environments

*Author:*

Zoltán Szarvas

Computer Science for Autonomous Systems

*Internal supervisor:*

Dr. Zoltán Istenes

Associate Professor

*External supervisor:*

Dr. Máté Nagy

Senior Research Fellow

*Budapest, 2023*

# Contents

# Chapter 1

# Introduction

## 1.1 Problem statement

In this work, I examine how to apply Transformer architecture as memory to create a control policy for a fixed-wing UAV that can catch and gain altitude using thermal flying in simulation. Additionally, I compare how the algorithm operates under different simulated wind conditions and turbulence. In the work I only consider one thermal and the policy's main goal is to catch the thermal and climb using it to reach the predefined maximum altitude as quickly as possible.

## 1.2 Preliminaries

### 1.2.1 What is soaring?

There is plenty of energy in the atmosphere, this energy is used by migratory birds to help their journey to their target. But also this energy is used by gliders, paragliders, and hang gliders to reach new and new world records both in achieved altitude and distance.

This energy is in the form of rising air currents with the following main types:

- Rising air currents which are induced by the heat radiated by the sun, are named *thermals*.

- Orographic lifts can be in the form of ridge lifts near slopes of mountains. On a larger scale, strong winds also can create waves on the mountains' lee side. Using these waves soaring pilots can reach altitudes over $10,000$ meters.
- There are other phenomena like convergence where air masses with opposite directions meet, then it forces the airmass to go higher altitude.
- Seabirds efficiently use wind speed differences to stay aloft for days with minimum energy. This is called dynamic soaring.
- Other phenomena also exist like polar vortex, which could be exploited to reach around $27,000m$ altitude [1] [2].

In this thesis, I am only concerned with thermal flying.

### 1.2.2 Thermals

#### How thermals formed

Thermals are triggered by the uneven warming of the ground. The sunlight heats up the ground, then the ground heats up the air particles close to the ground. This creates a hot air reservoir near the ground. Then any external disturbance, such as a light breeze, a tractor, or even a flock of birds can detach this layer of air. Since its temperature is higher than the surrounding air, it starts to rise.

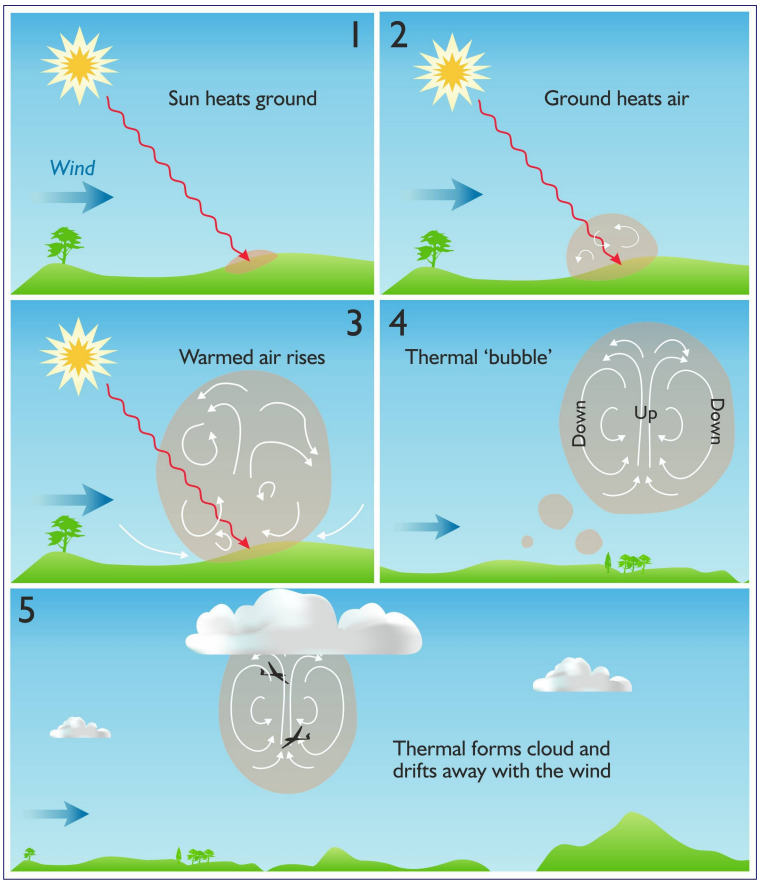



Figure 1.1: Birth of a thermal [3]

As it rises, the pressure decreases and it causes it to expand. The airmass uses energy to expand which causes its temperature to drop on average by around $0.98C°$ per 100 meters (Dry Adiabatic Lapse Rate) [4]. If the local temperature gradient of the environment also decreases (Environmental Lapse Rate), the rising air continues to ascend until the temperature of the thermal cools down to the environment's temperature. If the relative humidity in the rising air parcel reaches 100%, the water vapor in the air condenses, forming cumulus clouds. The condensation releases latent heat that was absorbed during evaporation, which increases the temperature of the rising air and its vertical velocity, allowing it to ascend even further. The ascending saturated air parcel drops its temperature on average around $0.6 - 0.7C°$ per $100m$ (Saturated Adiabatic Lapse Rate) in the middle troposphere [4]. It cools slower than the dry air parcel, so it can ascend even further. In extreme cases, it forms Cumulonimbus clouds which can cause thunderstorms.

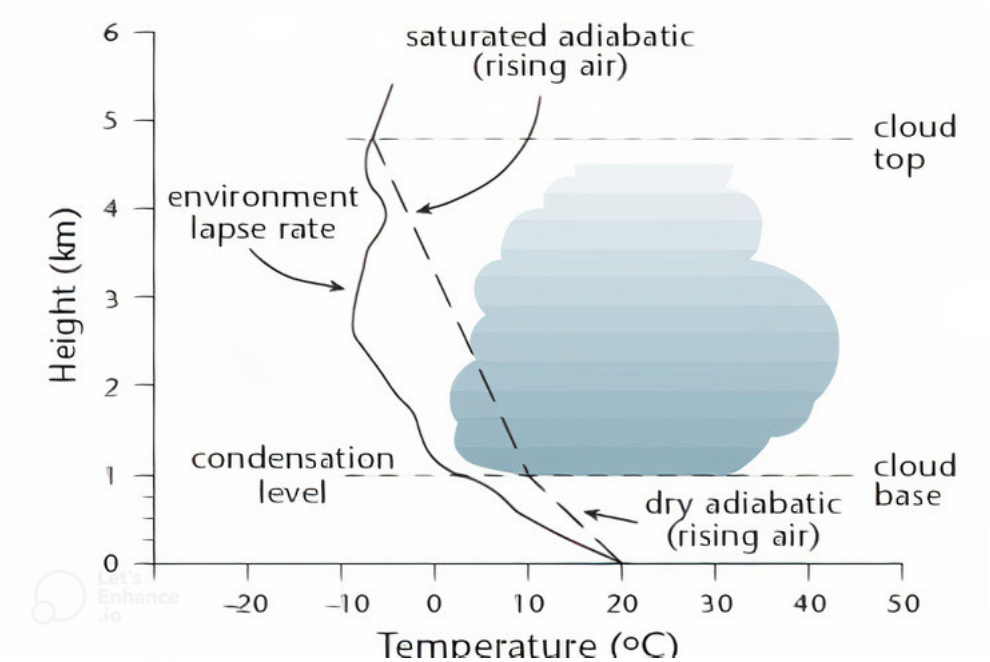
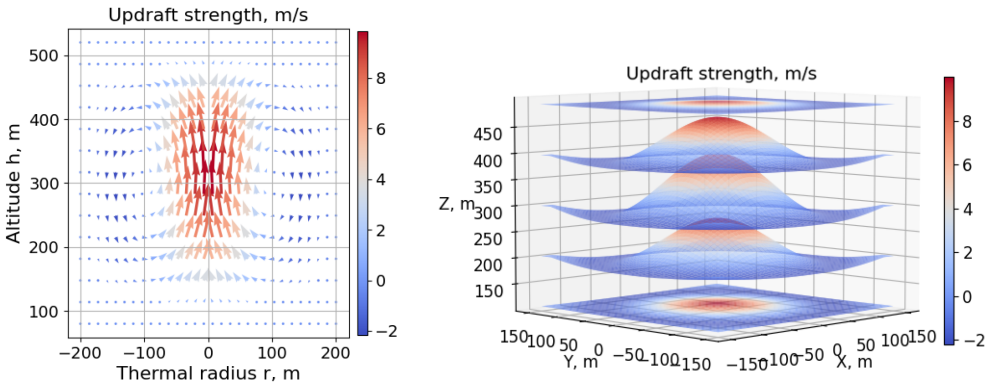



Figure 1.2: Lapse rates [5]

**Internals of thermals [6]**

There are no two identical thermals. While in a simplified model, thermals can be similar to a rising air chimney, in reality, the structure and dynamics are much more complex. [7] If there is not enough supply of hot air to the thermal, it does not create a tube, but only a bubble forms [8]. It is also possible that several smaller thermals branch out and merge at a certain altitude. The wind also has an effect on the shape of the thermal. As the air rises, the wind pushes it further away from the thermal's source location, so its shape can be likened to a tilted chimney in simplified models. The inside of the thermal is also not homogeneous, and in addition to being turbulent, the vertical velocity of the rising air is not constant in a horizontal cross-section of the thermal. In the center or core of the thermal, the upward air velocity is the highest, and as we move further away, it becomes smaller and smaller, and at the edge, we can find turbulent downdrafts. Vertically, the width of the thermal is also not uniform. As we ascend, it becomes wider. In Figure 1.3 you can see a simplified toroidal thermal model.




(a) Vector field (b) Vertical wind speed cross-sections

Figure 1.3: Torodial thermal model [9]

### What is thermaling

With the help of thermal updrafts, birds, humans, and UAVs can gain altitude and cover long distances by gliding to the next thermal. This is called thermal soaring. The goal during thermal soaring is to stay as close to the core of the thermal as possible because this is where the vertical air current is strongest. This is most efficiently achieved by circling. However, how close a given aircraft can circle to the core of the thermal depends greatly on the aerodynamic properties, weight, and airspeed of the aircraft. For example, a slower flying bird can circle on a much smaller radius than a larger glider which is heavier and flies faster.

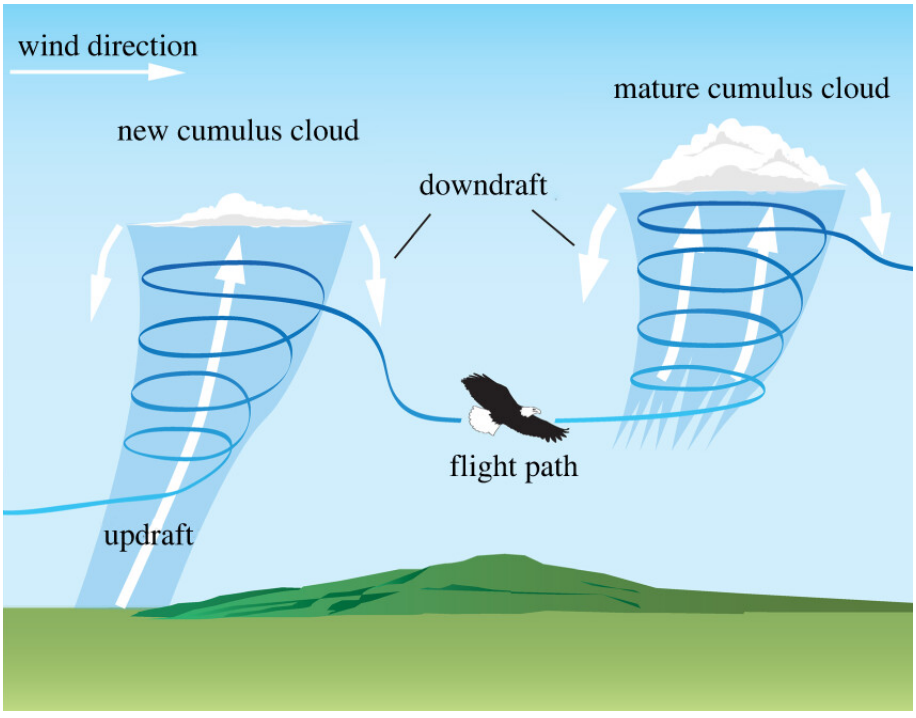



Figure 1.4: Thermal soaring [10]

### 1.2.3 Aerodynamics

#### Frame of references

When specifying the position and orientation of an aircraft, these are typically given in the following orthogonal coordinate systems or frame of references:

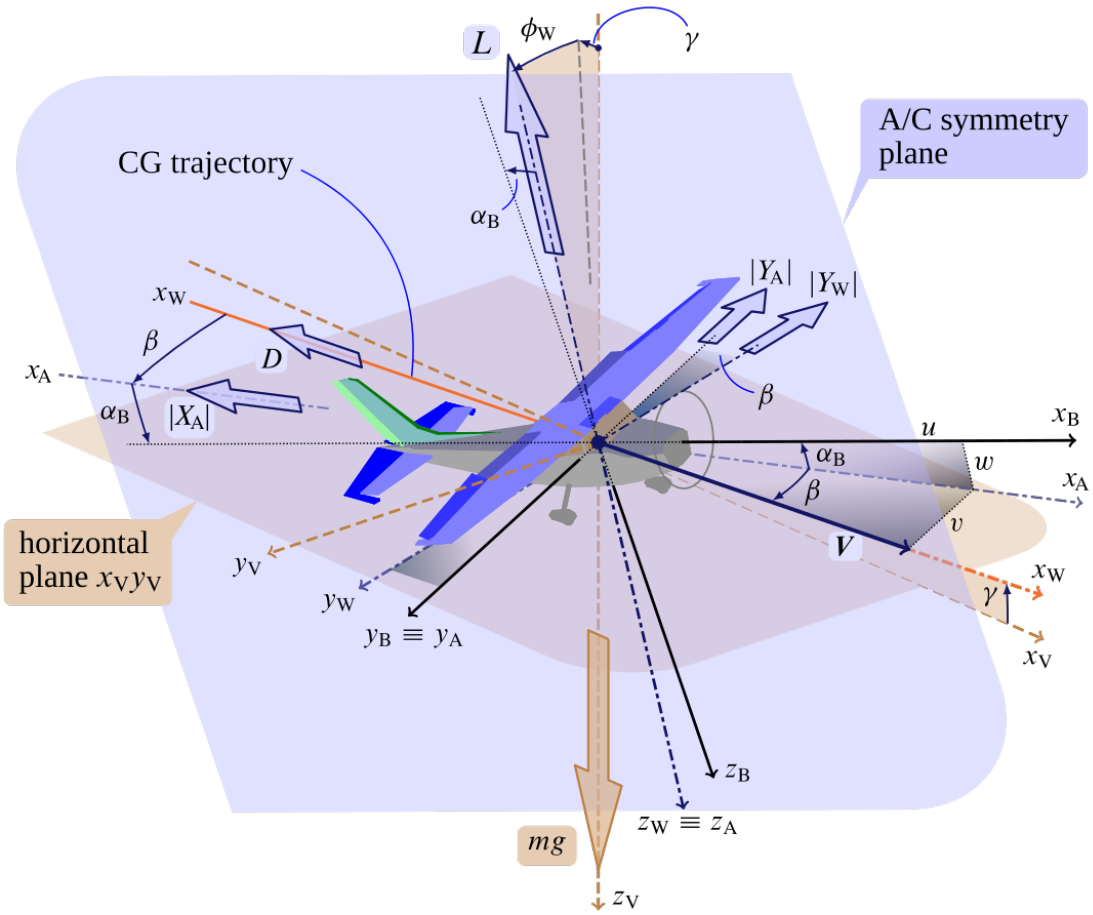



Figure 1.5: Frame of references of an aircraft. $(x_B, y_B, z_B)$ represents the *Body frame*, $(x_V, y_V, z_V)$ represents the local *Earth frame* and $(x_W, y_W, z_W)$ represents the *Wind frame*. $((x_A, y_A, z_A)$ represents the *Aerodynamic Frame* which is not discussed in detail) [11]

- Earth coordinate system (*Earth frame*): This reference frame is fixed to the Earth's surface, where the $x$ and $y$ define the horizontal plane of the Earth's surface, and the z-axis is aligned with the Earth's gravitational force. If it is aligned with the North Pole, then it is called NED (North-East-Down) frame. If we now assume that Earth's reference position (origin) is at the aircraft's center of gravity, then these coordinate vectors are the same as $(x_V, y_V, z_V)$ coordinates on Figure 1.5, in this case it is called local *Earth frame*.

- Body coordinate system (*Body frame*): This reference frame is attached to the airplane's body. The $x$-axis is aligned with the airplane's longitudinal axis, the $y$-axis is aligned with the lateral axis, and the $z$-axis is aligned with the airplane's vertical axis. This is denoted with $(x_B, y_B, z_B)$ coordinates on Figure 1.5.

- Flow or Wind coordinate system (*Wind frame*): The orientation of the airplane body does not always match the actual direction of motion relative to the air mass. The $x$-axis is aligned with the glider's velocity vector, the $y$-axis is aligned with the sideslip orientation, and the $z$-axis is aligned with the lift vector but in the opposite direction. This is denoted with $(x_W, y_W, z_W)$ coordinates on Figure 1.5.

We can perform the conversion of orientation between different coordinate frames using the following transformations.

**From Earth frame to Body frame** The rotation angles can be seen on Figure 1.6.

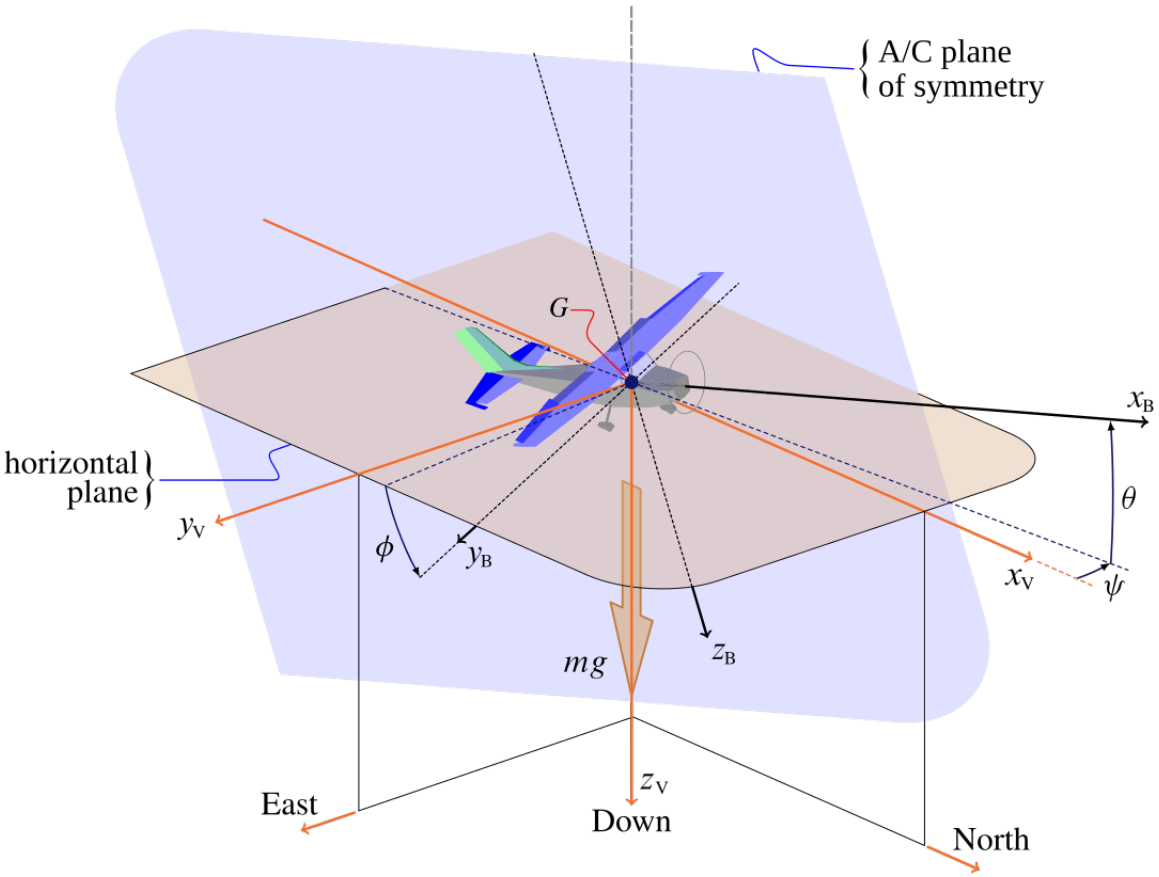



Figure 1.6: Rotation angles between (local) *Earth* $(x_V, y_V, z_V)$ and *Body frame* $(x_B, y_B, z_B)$ [11]

These are the followings:

- *Yaw ($\psi$)*: It is the angle between the north (in NED frame) and the projection of the aircraft's longitudinal axis onto the horizontal plane.
- *Pitch ($\theta$)*: It is the angle between the aircraft's longitudinal axis and the horizontal plane.
- *Roll ($\phi$)*: Rotation around the aircraft's longitudinal axis relative to the horizontal plane.

**From Wind frame to Body frame** This is the difference between the aircraft's longitudinal vector and the velocity vector, as seen in Figure 1.5.

- *Angle of Attack ($\alpha$)*: The angle between the aircraft's longitudinal axis and the horizontal plane of the *Wind frame*.
- *Sideslip angle ($\beta$)*: The angle between the velocity vector and the projection of the aircraft's longitudinal axis on the horizontal plane of the *Wind frame*.

**From Earth frame to Wind frame** This determines the Earth-relative flight path of the aircraft. These angles (except for heading), are shown in Figure 1.7.

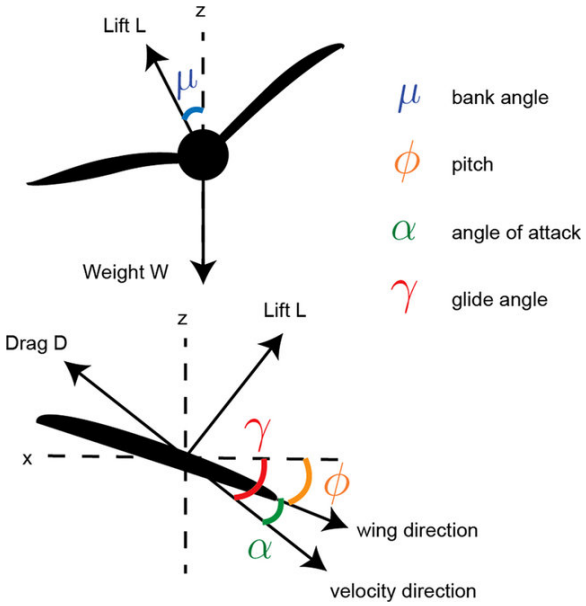



Figure 1.7: Rotation angles between (local) Earth $(x_V, y_V, z_V)$ and Wind frame $(x_W, y_W, z_W)$ [12]

- *Heading ($\rho$)*: It is the angle between the North and the horizontal component of the velocity vector.
- *Glide angle ($\gamma$)*: It is the angle between the horizontal plane of the *Earth frame* and the velocity vector.
- *Bank angle ($\mu$)*: Represents the rotation of the *Lift force* around the velocity vector relative to the Earth's gravitational force ($z_V$).

### Simplifications

In this thesis I use simplified aerodynamics, assumptions:

- The longitudinal axis of the aircraft and the velocity vector are collinear, therefore $\beta = 0$. The angle of attack ($\alpha$) is implicitly coded in lift and drag coefficients (see later in Equations under Section 1.2.3). The pitch angle is equal to the glide path angle ($\gamma = \theta$), if there is no wind, the yaw angle is equal to the heading angle ($\psi = \rho$), and the roll angle is equal to the bank angle ($\phi = \mu$). This further implies that all of the turns are fully coordinated, so there is no sideslipping in turns.

- The simulation uses point mass aerodynamics, meaning that the aerodynamic forces on different surfaces of the glider are not calculated separately. As a result, effects like torque around the wings due to differences in the lift are not captured.

- Turbulence and wind direction changes in the air velocity field do not affect the simulated glider's angle of attack ($\alpha$) and sideslip angle ($\beta$). Consequently, the simulation does not account for changes in lift, drag, or glider orientation change that would be induced by these effects.

- The air velocity field surrounding the glider, which includes wind and turbulence, only affects the point-like earth-relative velocity of the glider additively by the air velocity at that point.

- The only control input of the glider is the *bank angle* (*roll*). The angle of attack ($\alpha$) is uncontrolled, and the velocity and glide angle for the minimum sink rate is used. Turns are fully coordinated, there is no need to control the sideslip angle ($\beta$) to prevent slipping in turns.

### Equations

For every discrete time step, a new Earth-relative velocity vector is calculated based on the bank angle of the glider. Then the air velocity of the surrounding air velocity field is added to the glider's air velocity and then integrate over time to get a new position in *Earth frame*.

The equations of aerodynamics simulation are based on the work of Pedro Lacerda and András Zábó's MSc thesis[13].

During the flight, the following main forces acting on a glider, which balances each other in steady flight ($\sum F = 0$):

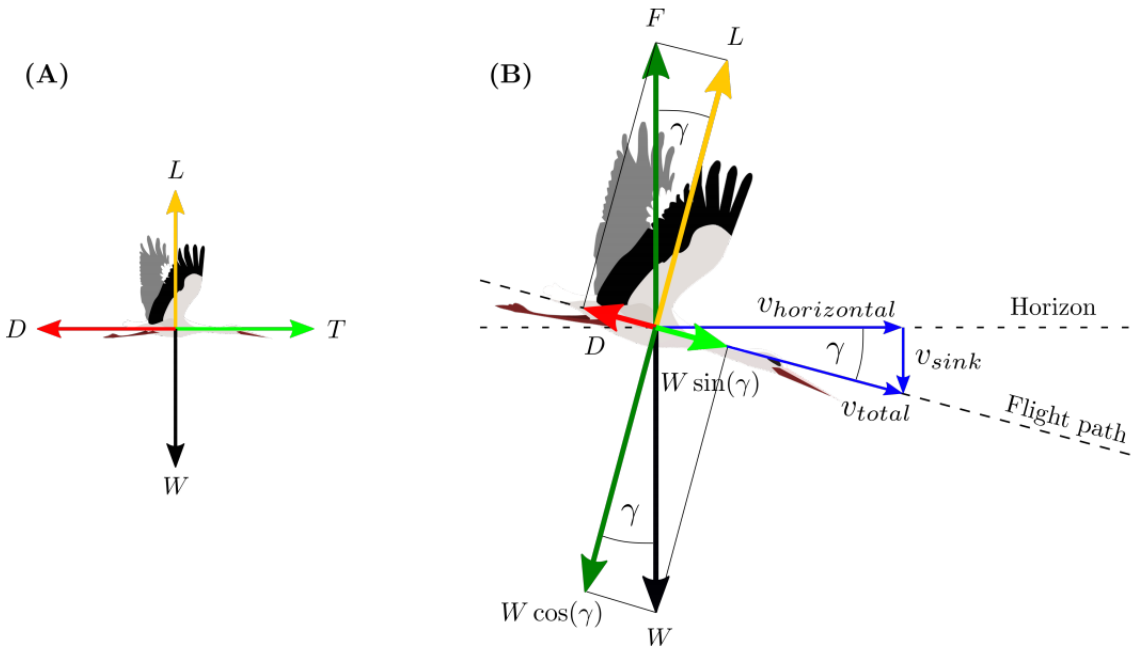



Figure 1.8: Forces acting on a flying object during level flight *(A)* and in gliding *(B)* [13]

- *Lift*: Due to the curved shape of the wing, there is a pressure difference between the upper and lower surfaces of the wing. Because of the curved shape, air moves faster at the top surface, resulting in lower pressure on top (Bernoulli's principle), and this suction effect contributes to lift, along with other factors like deflection of the air by the wing and characteristics of the air and its interaction with the wing.

- *Weight*: In level flight, weight balances the lift.

- *Drag*: Drag acts in the opposite direction to the glider's airmass relative motion. It consists of several components, including pressure drag resulting from the glider's shape, resistance caused by the glider's materials (skin drag), inference drag caused by the glider's structure, and induced drag caused by aerodynamic effects.

- *Thrust*: In level flight, thrust is required to maintain forward motion and balance the drag. However, in unpowered gliders, this is not possible. They can glide by maintaining a slight descent angle. This descent angle allows the component of weight that is collinear with the flight path ($W \cdot \sin(\gamma)$ in Figure 1.8) to act as a balancing force to counteract the drag.

The aerodynamic lift and aerodynamic drag can be calculated using the following equations:

$$L = C_L \cdot \frac{1}{2} \cdot \rho \cdot Sv^2 \tag{1.1}$$

$$D = C_D \cdot \frac{1}{2} \cdot \rho \cdot Sv^2 \tag{1.2}$$

where

- $L$ is the lift force in Newton ($\frac{kg \cdot m}{s^2} = N$)
- $D$ is the drag force in Newton ($\frac{kg \cdot m}{s^2} = N$)
- $C_L$ is the lift coefficient, dimensionless quantity. It is dependent on the shape and surface of the aircraft, angle of attack, and atmospheric parameters.
- $C_D$ is the drag coefficient, dimensionless quantity. It is dependent on the shape and surface of the aircraft, angle of attack, sideslip angle, and atmospheric parameters.
- $\rho$ is the air density ($\frac{kg}{m^3}$)
- $S$ is the wing area ($m^2$)
- $v$ is the true airspeed ($\frac{m}{s}$)

From these equations, we can decompose the airspeed along the flight path when the bank angle is zero.

$$v = \sqrt{2 \cdot \frac{W \cdot \cos(\gamma)}{S \cdot \rho \cdot C_L}} \approx \sqrt{2 \cdot \frac{W}{S \cdot \rho \cdot C_L}} \tag{1.3}$$

Where $W \cdot \cos(\gamma)$ represents the weight's component that balances the lift force. Since $\gamma \approx 0$, then $cos(\gamma) \approx 1$.

If the bank angle is not zero, meaning we are in a turn ($\mu \neq 0$), then the lift force's component which is equal to the weight force is given by $\frac{L}{\cos(\mu)}$. Therefore, the horizontal speed can be obtained by rearranging the Equation 1.3 as follows:

$$v_h = \sqrt{2 \cdot \frac{W}{S \cdot \rho \cdot C_L \cdot \cos(\mu)}} \tag{1.4}$$

To calculate the vertical speed, we use the glide ratio, which tells us how much forward travel ($d$ in Figure 1.9) corresponds to a unit of descent ($h$ in Figure 1.9).

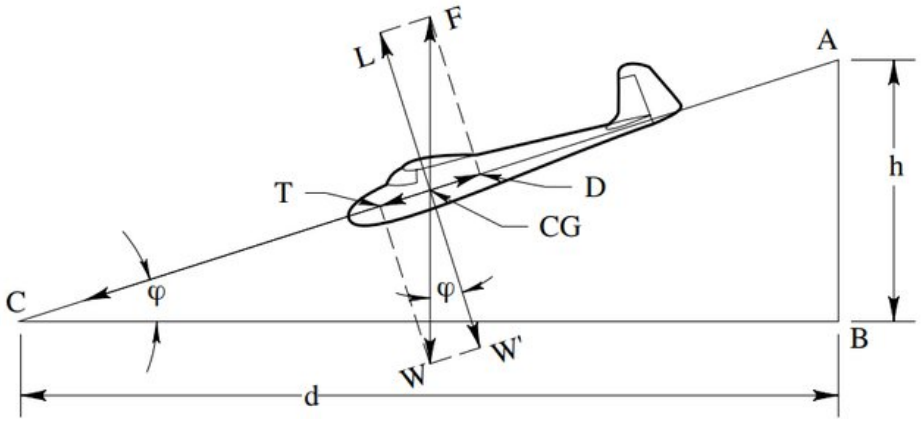



Figure 1.9: Glide ratio [14]

This can be expressed using the similarity of triangles:

$$E = \frac{L}{D} = \frac{C_L}{C_D} = \frac{d}{h} \tag{1.5}$$

From this, we can write the vertical speed as follows:

$$\frac{d}{h} = \frac{L}{D} \Rightarrow h = \frac{C_D}{C_L} \cdot d \Rightarrow v_v = \frac{C_D}{C_L} \cdot v_h \tag{1.6}$$

Using the formula from Equation 1.4, for the vertical speed at a given bank angle, we obtain the following (assuming $cos(\mu) \geq 0$):

$$\begin{aligned} v_v = \frac{C_D}{C_L \cdot \cos(\mu)} \cdot v_h &= \frac{C_D}{C_L \cdot \cos(\mu)} \cdot \sqrt{2 \cdot \frac{W}{S \cdot \rho \cdot C_L \cdot \cos(\mu)}} \\ &= \sqrt{\frac{C_D^2}{C_L^2 \cdot \cos(\mu)^2} \cdot 2 \cdot \frac{W}{S \cdot \rho \cdot C_L \cdot \cos(\mu)}} \\ &= \sqrt{\frac{C_D^2}{C_L^3 \cdot \cos(\mu)^3} \cdot 2 \cdot \frac{W}{S \cdot \rho}} \\ &= \frac{C_D}{C_L^{\frac{3}{2}} \cdot \cos(\mu)^{\frac{3}{2}}} \cdot \sqrt{2 \cdot \frac{W}{S \cdot \rho}} \end{aligned} \tag{1.7}$$

To get the new direction of the glider we use the Lift's horizontal component for a given bank angle $\sin(\mu) \cdot L$. Then using the current velocity's direction we can get the centripetal acceleration vector.

$$\vec{a_c} = \begin{bmatrix} -\cos(\rho) \\ \sin(\rho) \end{bmatrix} \frac{\sin(\mu) \cdot L}{m} \tag{1.8}$$

where

- $L$ is the lift force ($N$).
- $\rho$ is the current heading ($rad$).
- $\mu$ is the bank angle ($rad$).
- $m$ is the mass of the glider ($kg$).
- and $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ component in $\begin{bmatrix} -\cos(\rho) \\ \sin(\rho) \end{bmatrix}$ means that the acceleration vector is directed 90 degrees counterclockwise of the tangential velocity vector.

From this, the new heading of the glider can be calculated using the following equation:

$$\rho = \text{atan2}\left((\vec{v} + \vec{a_c} \cdot dt)_y, (\vec{v} + \vec{a_c} \cdot dt)_x\right) \tag{1.9}$$

Using the new airmass-relative horizontal velocity ($v_h$ in Equation 1.4), vertical velocity ($v_v$ in Equation 1.7) and direction ($\rho$ in Equation 1.9), we can calculate the new airmass-relative velocity vector. From this, we can get the Earth-relative position of the glider by adding it to the surrounding Earth-relative wind velocity vector.

### 1.2.4 Markov Decision Process (MDP)

We can describe the state of the thermal soaring by two main components. One of them describes the thermal itself, for example, in a simplified model, where its core is located, what its diameter is, how tilted it is, and how turbulent it is. The other component is the state of the glider, such as its relative position to the thermal core, altitude, bank angle, airspeed, and ground speed.

We are assuming that we know the state of the thermal and we know the state of the glider. If at each time instant using the current state only — without any information about the past — we can determine the next state from the current state, then the state has the Markov property.

The temporal and spatial changes of the thermal can be modeled as a stochastic process, and together with the glider flying in it, can be modeled as a continuous-time continuous-state space Markov chain. The Markov chain describes the probabilities of transitioning from one state to another.

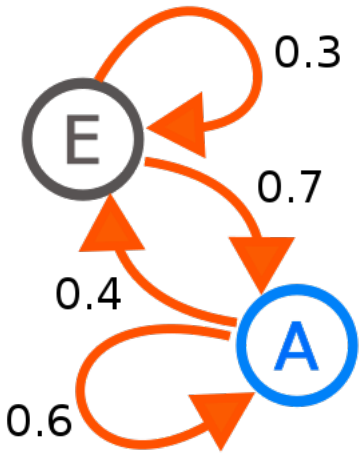



Figure 1.10: Markov Chain (Markov Process). Here $A$ and $E$ represent states, arrows represent state transitions and the numbers associated with the arrows represent the probability of that state transition.

If we add the possibility of control to this model, we obtain the continuous-time Markov Decision Process. In this thesis, I use discrete time steps, so by MDP, I mean the discrete-time Markov Decision Process from now on.

In an MDP an *Agent* interacts with the *Enviroment* by selecting *Actions* for the current *State* of the environment. After each action, the agent gets a *Reward* that represents feedback for the current state-action pair. However, this reward can be the result of an action in the distant past. This is called the credit assignment problem because the agent should somehow assign credit for actions in the past that resulted in a delayed reward.

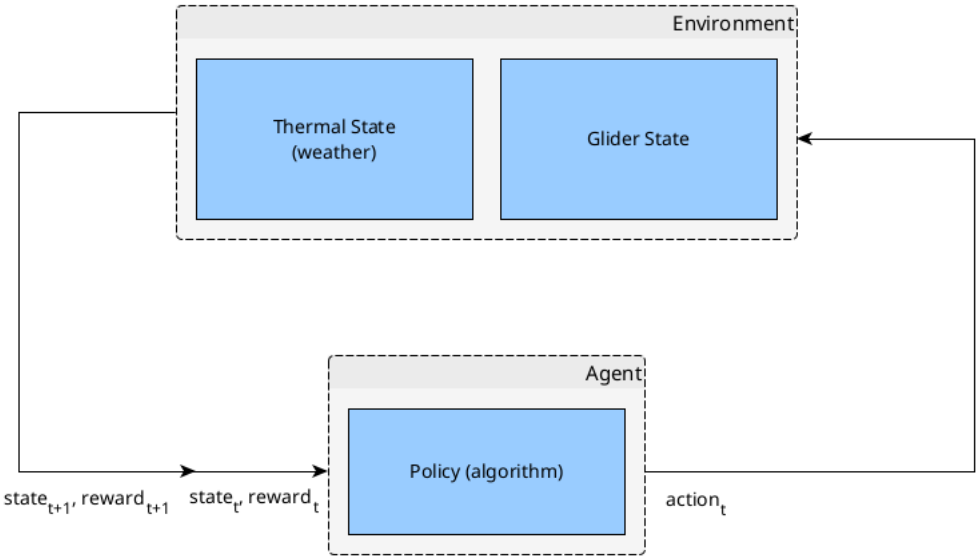



Figure 1.11: Thermaling problem as Markov Decision Process (MDP)

The Markov decision process can be represented by the 6-tuple $(\mathcal{S}, \mathcal{A}, P, R, s_0, \gamma)$, where:

- $\mathcal{S}$ represents the state space. In the thermaling problem, it contains both the thermal's and glider's state.
- $\mathcal{A}$ represents the action space. It contains the set of actions which the agent can select. In the thermaling problem, it can represent the glider's control inputs.
- $P$ is a mapping that describes the probability of the system going from a given state to a specific state if it gets a control input as action. It is called *state-transition probability* ($\doteq$ indicates "by definition")

$$\begin{gathered} P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] \\ P(s, a, s') \doteq Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) \\ \forall s', s \in \mathcal{S}, a \in \mathcal{A} \end{gathered} \tag{1.10}$$

  In the thermaling problem, it describes the system's dynamics including the thermal and the glider. If we could model this, we could get the optimal policy using the Bellman equation (see Equation 1.15 later).

- $R$ is a mapping that returns the immediate reward based on the selected action in a given state.

$$\begin{gathered} R : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \\ R(s,a) \doteq \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A} \end{gathered} \tag{1.11}$$

  In the thermaling context, the reward could mean that the glider reached the top of the thermal and also what is the current climb rate in the thermal.

- $s_0$ represents the starting state of the process.

- $\gamma \in [0,1]$ represents the discount rate. It discounts the future reward from time step $t$.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.12}$$

  It determines how future rewards are taken into account in the current value. A larger value of $\gamma$ means that the future rewards are given more weight, while $\gamma = 0$ means that only the immediate reward is taken into account in the calculation.

For an MDP, the Markov assumption holds, therefore:

$$Pr(S_t \mid S_0, A_0, S_1, A_1 \ldots S_{t-1}, A_{t-1}) = Pr(S_t \mid S_{t-1}, A_{t-1}) \tag{1.13}$$

To solve the thermaling problem framed as an MDP, Reinforcement Learning (RL) is used. The goal of the RL is to learn an optimal policy that gives the maximum expected cumulative reward for the agent.

$$\max \mathbb{E}_\pi \left[ \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right] \tag{1.14}$$

The policy $\pi$ is mapping which determines the next action to select given a state. $\pi : \mathcal{A} \times \mathcal{S} \to [0,1]$ $\pi(a,s) = Pr(A_t = a \mid S_t = s)$. It determines the probability of using a given action in a given state.

When an agent operates with an environment, using its policy it selects an action in every state, then receives the next state and immediate reward from

the environment, until it reaches a terminal state in the case of episodic tasks: $s_0 \to a_0 \to r_1, s_1 \to a_1 \to r_2, s_2 \to \dots$. The thermaling problem is considered an episodic task, where an episode ends when the glider reaches a certain altitude.

### Methods to find a Policy

All finite MDPs have at least one optimal policy [15] and if we know the transition probabilities ($P$) and the reward function ($R$), we can find this optimal policy using the Bellman Optimality Equation.

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} \sum_{s'} P(s, a, s') \left[ R(s, a) + \gamma V^*(s') \right] \\ Q^*(s, a) &= \sum_{s' \in \mathcal{S}} P(s, a, s') \left[ R(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \end{aligned} \tag{1.15}$$

If we do not know these, we can sample from the environment and choose action using, for example, value-based or policy-based methods. In value-based methods, we assign values to each state or state-action pair and choose the action with the highest value similar to the Bellman Optimality Equation but without knowing the system's dynamics. Many of these methods are off-policy which means that the policy used for sampling the environment is not the same as we improving, therefore we can reuse steps collected at any point of the training, which gives us sample efficiency.

In policy-based methods, we continuously fine-tune the parameters of a parameterized policy function, such as through approximation. Many of these methods are on-policy, which means that we use the policy which we improve for exploring the environment. In this case, we usually cannot reuse samples that are created using the previous version of the policy. However, because we directly optimize the policy it makes it more stable. Although, it has high variance which can destroy a training process. But, it has high variance which can disrupt the training process.

In addition to these methods, there are many others, such as imitation learning [16], which attempts to mimic the behavior of an expert agent, meta-learning methods, which aim to develop RL algorithms that can quickly adapt to new tasks (few-shot learners) [17], or these methods can be mixed, such as in the Actor-Critic method [15], where both policy-based and value-based methods are used together.

In this thesis I am using a Policy Gradient[18] based method Proximal Policy Optimization (PPO) [19].

### 1.2.5 Policy Gradient methods [18][20][21]

The goal of Reinforcement Learning is to maximize the expected reward while following a policy that can be parameterized. The objective function, denoted by $J(\theta)$, needs to be maximized with respect to the parameters $\theta$ of the policy $\pi$:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \tag{1.16}$$

Our goal is to approximate the optimal $\theta^*$, and to achieve this, we use *Gradient Ascent* where $\alpha$ is the step size:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} J(\theta_t) \tag{1.17}$$

For the gradient, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] \tag{1.18}$$

where $\tau$ trajectories are sampled using the policy parameterized by $\theta$.

The following equality helps the calculation of the gradient:

$$\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau)\frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) \tag{1.19}$$

Using the equality, the gradient of the expected value:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] &= \nabla_\theta \int \pi_\theta(\tau) R(\tau) d\tau \\
&= \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau \\
&= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right]
\end{aligned} \tag{1.20}$$

We can get the formula of $\log \pi_\theta(\tau)$ using the definition of $\pi_\theta(\tau)$:

$$\pi_\theta(\tau) = P(s_0)\prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t) \tag{1.21}$$

This is the product rule of probability, which holds because new actions are independent of the previous ones due to the Markov Property. If we take the logarithm of $\pi_\theta(\tau)$, then the product becomes sum:

$$\log \pi_\theta(\tau) = \log P(s_0) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T} \log P(s_{t+1}|s_t, a_t) \tag{1.22}$$

Taking the derivative with respect to $\theta$ gives:

$$\nabla_\theta \log \pi_\theta(\tau) = \sum_{t=1}^{T} \nabla \log \pi_\theta(a_t|s_t) \tag{1.23}$$

After substituting this into the 1.20 formula, we obtain the gradient of the Policy Gradient loss function:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \end{aligned} \tag{1.24}$$

A key takeaway from this is that it's not necessary to know the system dynamics to approximate the policy, which classifies this as a model-free algorithm.

We can approximate the expected value by sampling many trajectories. In its current form, $R(\tau)$ encompasses the entire trajectory (including the past time steps), which can introduce significant variance in sampling. Instead, we can use discounted future returns ($G_t$), which is the sum of rewards from the current time step $t$ into the future, where each future reward is discounted by $\gamma$. It helps the optimization to focus on maximizing the future reward from the current time step.

With this, we obtain the gradient of the objective function, which is at the core of REINFORCE [22] algorithm:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} G_t \nabla \log \pi_\theta(a_t \mid s_t)\right) \tag{1.25}$$

### Baseline and Generalized Advantage

The classic REINFORCE algorithm has high variance because formula 1.25 above favors not only actions that were completely good but also those that were bad but still resulted in a positive future discounted reward in the episode. This is because the $G_t \log \pi_\theta(a_t \mid s_t)$ will be positive, therefore helping to strengthen suboptimal actions too.

One way to reduce variance is to introduce a baseline, which achieves this without changing the expectation value. [15] [20] [23]

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(G_t - b\right) \right) \tag{1.26}$$

The difference $(G_t - b)$ is also known as the advantage for the given state and action, and can be calculated in multiple ways. If it is positive, choosing the current action is better than choosing others.

A good choice for the baseline is the State Value, which is the expected reward resulting from following the policy $\pi_\theta$. It is defined as

$$V(s) = \mathbb{E}_{\pi_\theta} \left[ G_t \mid S_t = s \right] \tag{1.27}$$

However, since we do not know the state value, we can use a learnable estimate parametrized by $\omega$ as $V_\omega(s)$. Thus, the estimated advantage would be $\hat{A} = G_t - V_\omega(s_t)$.

However, if we use this approach, there will be high variance because we are also estimating the State Value function, which could be incorrect in the initial stages and may not even converge. If we use bootstrapping for $G_t$, we can reduce this variance, but it comes with an increased bias, which means that the estimated advantage will be less accurate. Bootstrapping involves using the previous version of the estimated function recursively for the next time step during the update of the estimate. The single-step bootstrapped estimate of $G_t$:

$$G_t \approx r_{t+1} + \gamma V_\omega(s_{t+1}) \tag{1.28}$$

And this gives us the objective function of the single-step bootstrapped Actor-Critic Policy Gradient:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)\right)\right) \tag{1.29}$$

Here, the Actor's objective is to choose the best action, while the Critic's objective is to estimate the value function. The Critic's objective function is:

$$J^{vf}(\omega) = \frac{1}{2}\left(r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)\right)^2 \tag{1.30}$$

with gradient:

$$\nabla_\omega J^{vf}(\omega) = r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t) \tag{1.31}$$

We can reduce the variance of the advantage by increasing the number of bootstrap steps. The Generalized Advantage Estimation (GAE) [24] provides a good framework for configuring this. The formula for the GAE is as follows:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-1+1}\delta_{T-1} \tag{1.32}$$

where

$$\delta_t = r_t + \gamma V_\omega(s_{t+1}) - V_\omega(s_t) \tag{1.33}$$

It uses an exponential average, with $\lambda \in [0, 1]$ serving as the exponential weight discount. $\delta_t$ is the TD residual of V using discount $\gamma$.

- If $\lambda = 0$ then we get back the single-step bootstrapped value: $r_t + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)$. It will have low variance, but high bias.
- If $\lambda = 1$, we get back $G_t - V_\omega(s_t)$ with the discounted future reward for the whole episode. Here we get high variance but low bias.

Thus, we can control the bias-variance tradeoff using the $\lambda$ parameter.

### Proximal Policy Optimization (PPO) [19]

Baseline helped to solve the high variance problem, however, the training was too slow or too fast because of the fixed step size $\alpha$ of $\theta_{t+1} = \theta_t + \alpha\nabla_{\theta_t} J(\theta_t)$. Adaptive

schedulers could help, but they had no information about how the weight changed the distribution of the policy function $\pi(a_t \mid s_t)$. Proximal Policy Optimization's goal is to avoid excessively large policy updates to stabilize the learning process. It is doing it in a manner that ensures that the difference between the old policy and the new policy is not too large. This is where the term *proximal* is come from.

It is based on TRPO (Trust Region Policy Optimization) [25], which aims to determine a region around the objective function where the step is still safe. To achieve this, TRPO compares the old and new policies and ensures that the difference between them is small. In TRPO, this constraint was given as a penalty in the cost function, using the Kullback-Leibler divergence (KL divergence) [26]. The KL divergence measures how different one probability distribution is from another; therefore it didn't allow large policy changes.

Instead of using the KL divergence, PPO uses a clipped surrogate function as its objective function:

$$J^{clip}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\min(w_t(\theta)\hat{A}_t, clip(w_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right] \tag{1.34}$$

Here $w_t$ is the ratio that defines the importance sampling weight between the old and new policies. Therefore it is used to measure the change between the new and old policies.

$$w_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \tag{1.35}$$

If $w_t(\theta) > 1$, it means that the new policy is more likely to select $a_t$ at state $s_t$ than the old policy. If $0 < w_t(\theta) < 1$, then the action $a_t$ is selected less likely in the new policy for state $s_t$. If $w_t(\theta) = 1$, then there is no change in the probability. If $w_t(\theta)$ would be zero or undefined, then the clipping would prevent numerical instability caused by division by zero or zero value for $w_t(\theta)$.

The equation uses $w_t(\theta)$, instead of the log probability ($\log \pi_\theta(a_t \mid s_t)$), to define the amount and direction of the policy change. And it also takes the minimum of the clipped and unclipped parts so that the final objective will be a lower bound.

PPO's algorithm is simpler to implement than TRPO's and gives similar results; however on some fine-control continuous tasks, TRPO performs better. [27]

If we combine the clipped surrogate function from Equation 1.34 (Actor), the objective function of Value Function approximation from Equation 1.30 (Critic), and the Entropy loss objective from the Equation below 1.37, we get the following loss function, which should be minimized:

$$J^{ppo}(\theta) = J^{clip}(\theta) + c_1 J^{vf}(\theta) - c_2 H\left(\pi(\cdot \mid s_t)\right) \tag{1.36}$$

$H\left(\pi(\cdot \mid s_t)\right)$ represents the Shannon Entropy [28] of the policy function in state $s_t$ at time step $t$:

$$H\left(\pi(\cdot \mid s_t)\right) = -\sum_{a \in \mathcal{A}} \pi(a \mid s_t) \log \pi(a \mid s_t) \tag{1.37}$$

The $c_1$ and $c_2$ coefficients can be used to set weights for the different parts of the loss function. The Exploration-Exploitation tradeoff can be controlled by $c_2$, increasing it results in a higher exploration rate.

In this thesis, I am using PPO algorithm for learning a policy that can be used to control the glider.

### 1.2.6 Partial Observability

So far, we have assumed that the complete state of the system is known, which includes both the thermal and the glider's state variables. However, since the glider can only gather information about the environment through its sensors, it does not know the complete state of the system. Therefore, we treat the problem as a partially observable one. In this case, the agent does not know the current state of the system, but instead, it receives observation after each action: $o_t \in \mathcal{O}$. Thus the agent observes the following history:

$$H_t \doteq o_0 \to a_0 \to r_1, o_1 \to a_1 \to r_2, o_2 \to \dots \tag{1.38}$$

If we can create a state representation from this history using a mapping function $f$, i.e., $s_t = f(H_t)$, and the resulting state is Markovian, then the problem can be treated as MDP. Markovian means that if we have two different states generated

by different histories but with the same mapped state representation, executing the same action will result in the same observation: $f(h) = f(h') \Rightarrow Pr(O_{t+1} = o \mid H_t = h, A_t = a) = Pr(O_{t+1} = o \mid H_t = h', A_t = a)$ for all possible histories and actions.

While creating such an exact mapping may be unfeasible, we can approximate it. Furthermore, using the entire trajectory does not scale well; therefore we can use a context window that is sufficient for the problem at hand. This may violate Markov property, but it can still provide practically and computationally feasible solution [15].

In the context of thermaling, the glider's goal during circling is to stay as close as possible to the core, where the thermal is the strongest. Thus, by continuously exploring the vertical speeds during circling, the glider can determine the direction in which it needs to shift the center of the circle.

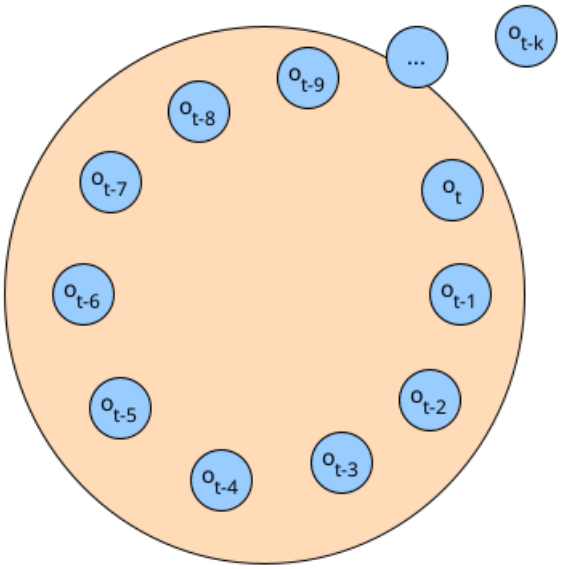



Figure 1.12: Thermaling history

Therefore, to practically solve the thermal soaring problem, it isn't necessary to keep track of the entire history to generate the MDP state using the $f$ function; instead, it's sufficient to use a relatively short context window. For example taking a fixed-wing UAV that flies at $10\frac{m}{s}$ as a basis, the radius of the circle in the case of 35° bank angle can be calculated using the following formula[29]:

$$r = \frac{v_t^2}{g \tan \phi} = \frac{10^2 \frac{m^2}{s^2}}{9.81 \frac{m}{s^2} \cdot 0.7} \approx 14.55m \tag{1.39}$$

From this, we can calculate the rate of turn, which indicates how many degrees the glider travels on the circle in a unit of time:

$$\omega = \frac{v}{r} = \frac{10\frac{m}{s}}{14.55m} \approx 0.686\frac{rad}{s} = 39.35\frac{deg}{s} \tag{1.40}$$

If we take a sample every 0.5 second, approximately $\frac{360°}{39.35°\cdot 0.5} \approx 18$ samples would be sufficient to represent one full circle. If the context window could be extended to encompass more circles, it might provide a better approximation of the thermal.

### 1.2.7 Attention mechanism

As the partially observable thermaling problem can be modeled as a sequence, it provides the opportunity to utilize the Attention mechanism and a part of the Transformer model to create the mapping function $f$ which encodes the history of observations to an approximated Markov state $s_t = f(H_t)$.

The attention mechanism was successfully used for machine translation tasks firstly with RNN models[30][31], and later it became the core of the Transformer model[32], which revolutionized Natural Language Processing (NLP) and other subfields of AI.

The essence of the attention is to weigh individual elements when processing sequences so that the weights are dynamically calculated. This allows us to dynamically determine which elements of an input sequence to attend to more than others.

An attention mechanism as described in "Attention is all you need"[32] can be thought of as a database system. Every element in the sequence is represented by a vector. In NLP, these are usually word or subword embeddings, the vector representations of words or subword tokens. From these vectors, we can learn the following feature vectors which represent different things:

- *Query*: When we would like to find something in a database, we usually use some kind of query for it. This is represented by this feature vector.
- *Key*: Following the previous analogy, the key represents what an element in the database offers. When we search for something, we compare the key with

the query and return how good the match is. We use a *scoring function* to determine the degree of how well the match is. The scoring function is usually implemented by simple similarity metrics like the dot product, but any other method can be used like Multilayer Perceptron (MLP) [33].

- *Value*: It represents the actual data that we can use for further processing after the selection.

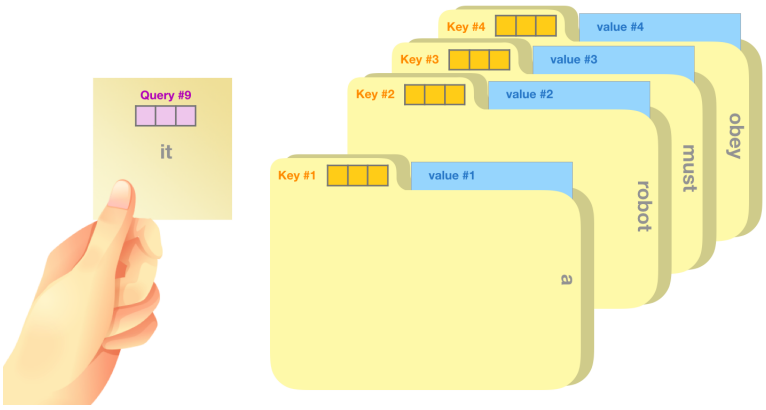



Figure 1.13: Attention as database analogy[34]

To calculate the actual weight which represents the degree of how a sequence element is attended softmax is used [35]:

$$\alpha_i = \frac{exp(f_{attn}(key_i, query)}{\sum_j exp(f_{attn}(key_j, query))} = softmax(f_{attn}(key_i, query)) \tag{1.41}$$

This gives a weight value for each sequence element for a given $query$, $\sum_{i=0}^{n} \alpha_i = 1$ for the whole sequence. $f_{attn}$ represents the scoring function, $key_i$ represents the key of the current sequence element.

After calculating the weights for each sequence element, these weights are used to calculate the weighted average of the values of the corresponding sequence elements.

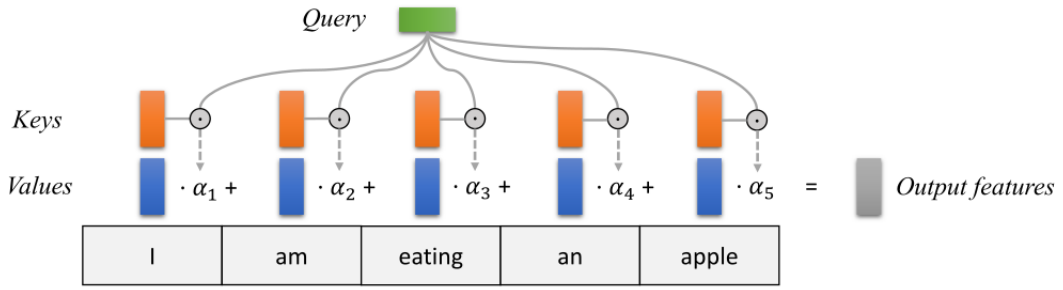



Figure 1.14: Calculating weighted average based on attention[35]

In "Attention is all you need" scaled dot product attention was used which is given by the equation:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1.42}$$

Here $Q, K$, and $V$ parameters are matrices and $QK^T$ represents the dot product, and $\frac{1}{d_k}$ gives the scaling by $d_k$ which represents the dimension of the key feature vector. In the matrices, the dimension of the rows represents the length of the sequence, while the columns represent the dimensionality of the learned feature vectors. If the $Q$ is created using the same sequence as $K$ and $V$, we are talking about self-attention, in this case, the attention mechanism determines how much importance each sequence element should assign to all other elements in the same sequence. It allows the model to capture complex relationships between several parts of the sequence.

### 1.2.8 Transformer Architecture

The self-attention became the core of the Transformer [32] model, it became the state-of-the-art model without using Recurrent Neural Networks (RNNs). It consists of two main parts: the *Encoder* and the *Decoder*. In the original paper, there were six of both the left-hand side *Encoder* module and the right-hand side *Decoder* module stacked on top of each other as shown in Figure 1.15.

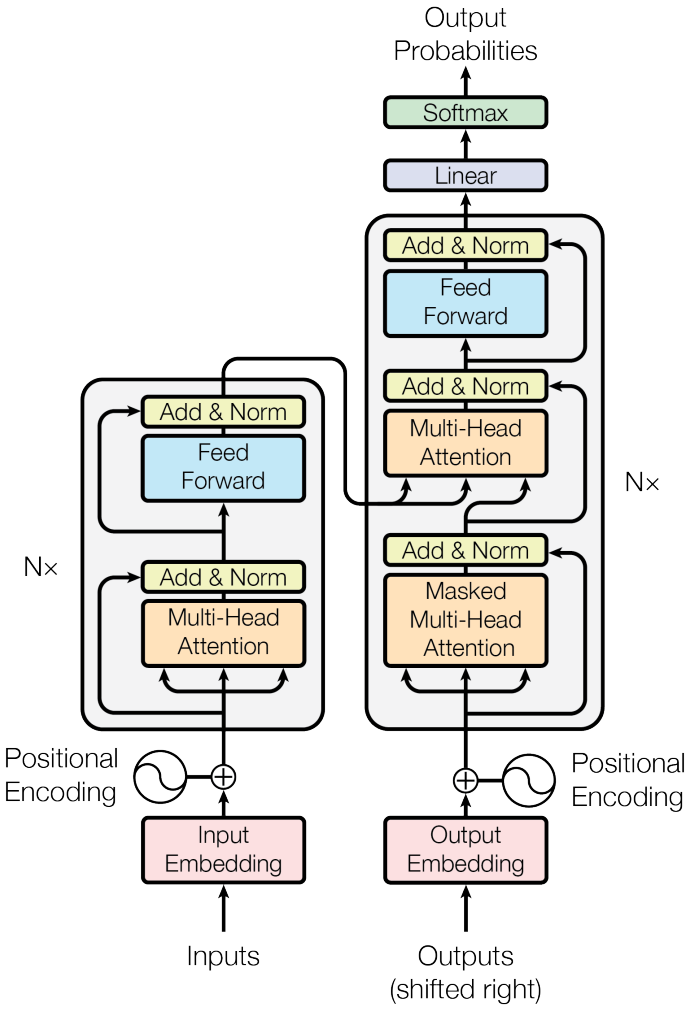



Figure 1.15: Transformer architecture[32]

In the *Encoder*, the input tokens are first transformed into embedding vectors using the *Input Embedding* layer, and then sinusoidal position information is added to these vectors using the *Positional Encoding* layer. It helps the model to take into account the relative positions of sequence elements:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{1.43}$$

After this, they fed into the *Multi-Head Attention*, which contains several attention layers that run in parallel. With the help of multi-head attention, the model can build different relationship subspaces between the positions simultaneously. The output dimension of the feature vector after the *Multi-Head Attention* has the same dimensionality as the input embedding vector. Following the residual connection and Layer Normalization [36], a Feed Forward Neural Network is applied, which is

then followed by another residual connection and layer normalization.

The input and output feature vector dimensions of the *Encoder* are the same as the input embedding's but are applied position-wise, meaning that the encoder module's output is a matrix where the row dimensionality is the same as the input sequence's length and column dimensionality is the same as the input embedding vector dimensionality.

In Figure 1.15, the architecture of the *Decoder* module on the right-hand side differs only slightly from the *Encoder*'s. The input received by the *Decoder* is the previous output generated by the *Decoder*. In the beginning, it receives an embedding corresponding to a start token. To take the input into account, an additional *Encoder-Decoder Attention* layer has been added where the *queries* come from the previous *Decoder* layer, while the *keys* and *values* come from the *Encoder*'s output. This allows every position in the *Decoder* to attend to all positions in the input sequence. Another difference is that the *Decoder*'s attention is causal, meaning that each position can only attend to the positions preceding it. This is important because the output is generated autoregressively as individual tokens, which cannot attend to elements of the sequence that is generated later.

In this thesis I am using the *Encoder* part of the Transformer model with causal attention to encode the observation history (see later in Section 2.1).

### 1.2.9 Transformer Architecture with Reinforcement Learning

Several approaches have been explored to utilize Transformers in the context of Reinforcement Learning, as described in the paper *A Survey on Transformers in Reinforcement Learning* [37]. These approaches include using Transformers to encode multi-entity observations into a single embedding [38], encoding temporal sequence [39], or employing them in Meta-Reinforcement Learning [40]. The [37] paper also discusses the challenges researchers have encountered when using the vanilla Transformer architecture and various modifications proposed to enhance training stability, particularly in on-policy training. For example, one of the approaches involves modifying the architecture through layer normalization reordering and intro-

ducing a gating mechanism [39], while another one eliminates layer normalization and addresses gradient stabilization through network weight initialization [41].

In this thesis, I employ the latter method, known as T-Fixup initialization. Details of this approach will be discussed later in Section 2.1.

## 1.3 Related work

There have been several studies on the subject, both in simulation and using real UAVs. Essentially, we can encounter two types of approaches. On the one hand, there are those that create some kind of model of the structure of thermals and continuously fit the thermal parameters while flying [42][43][44]. This is the model-based approach. The other approach is when the algorithm does not have a fixed model of the problem, but rather uses a trained policy [45][12][46]. This is called the model-free approach.

**Model-based approaches** The first successful known autonomous soaring project was at NASA in the Autonomous Soaring Project [42]. Their algorithm running on the UAV continuously estimated the thermal size and position and then selected the control inputs based on the estimated values. The first soaring controller which is integrated into publicly available autopilot software ArduPilot is the ArduSoar, which uses an Extended Kalman Filter (EKF) to estimate the parameters of a Gaussian thermal model [43]. In [44], the problem was formed as a Partially Observable Markov Decision Process, and EKF was used for belief update. It also implemented an exploration-exploitation tradeoff mechanism.

**Model-free approaches** Reinforcement Learning (RL) was used with SARSA in [45] to learn a policy to first fly in turbulent flows in a simulator and then in the field [12]. The first work which used Reinforcement Learning not only for updraft exploitation but also for cross-country soaring was [46], which used a hierarchical RL approach for decomposing the task into separate learnable subtasks. Here, LSTM was used to encode the agent's memory.

# Chapter 2

# Method

The method of integrating the thermal soaring task into the RL framework is illustrated by the main components of the Environment and the Agent in Figure 2.1.

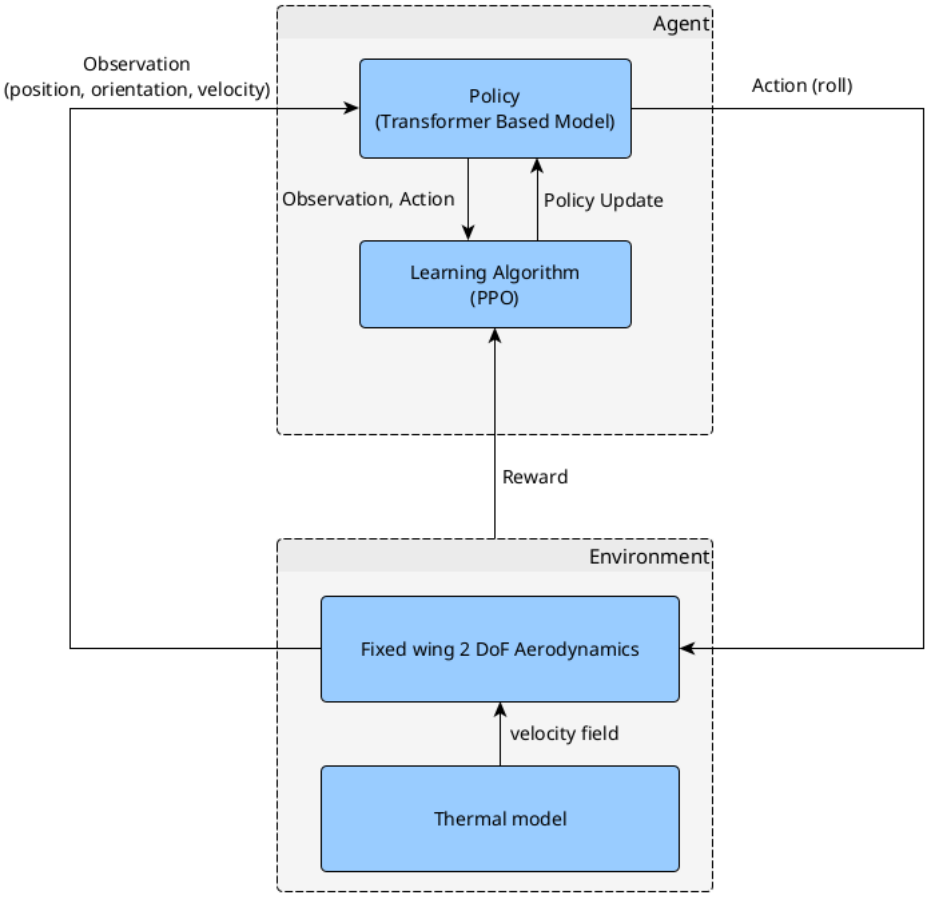



Figure 2.1: Reinforcement Learning architecture for the thermaling problem

The task is implemented in a simulation. The Environment runs the simulation

of the glider's aerodynamics and thermal's dynamics.

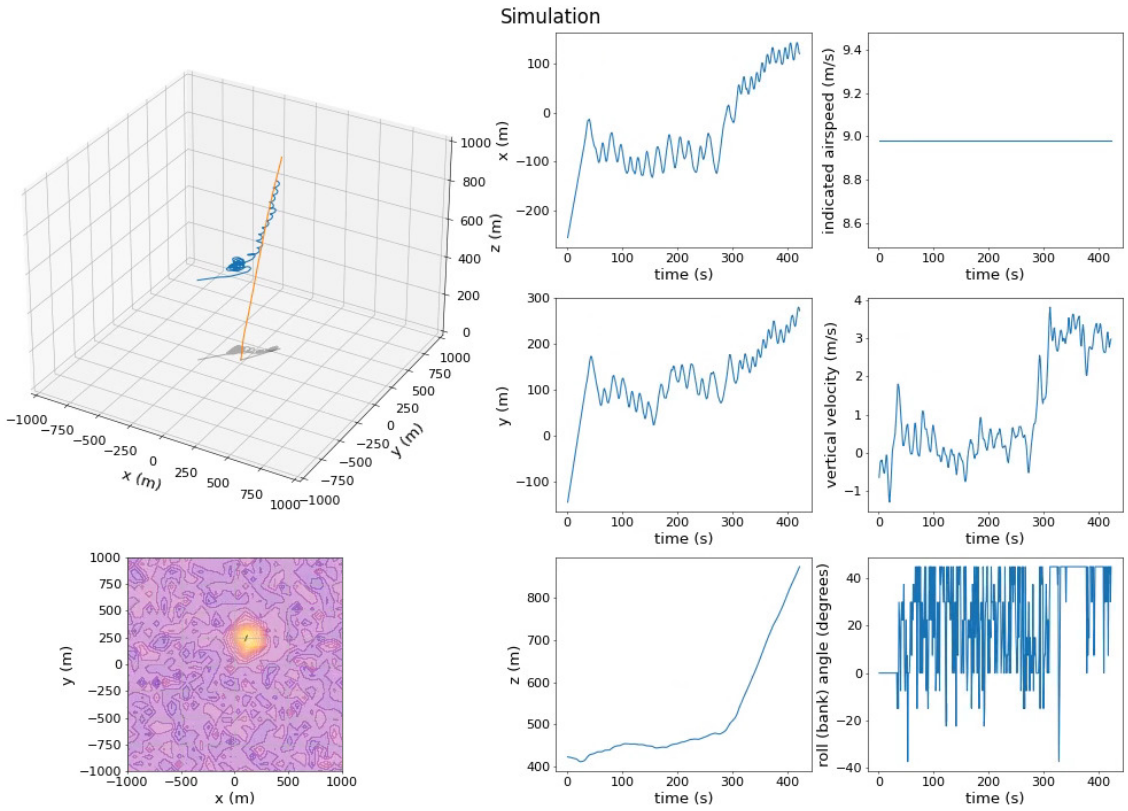



Figure 2.2: Snapshot of the visualization of the simulator

The Agent consists of a Policy, which is based on a modified Transformer architecture (section 1.2.8), and the PPO algorithm (section 1.2.5) was used for the training. During training, the PPO algorithm modifies the parameters of the model based on the rewards received from the Environment.

When creating the model, the following assumptions were taken into account:

- Discrete time is used ($t = 1, 2, 3, \dots$).
- The state space is *continuous* with a finite number of dimensions.
- This state space is *partially observable* using *continuous observation space* with a finite number of dimensions.
- The action space is discrete.
- The task is episodic.

## 2.1 Transformer Based Architecture

The architecture of the model used for the Agent's policy is illustrated by Figure 2.3.

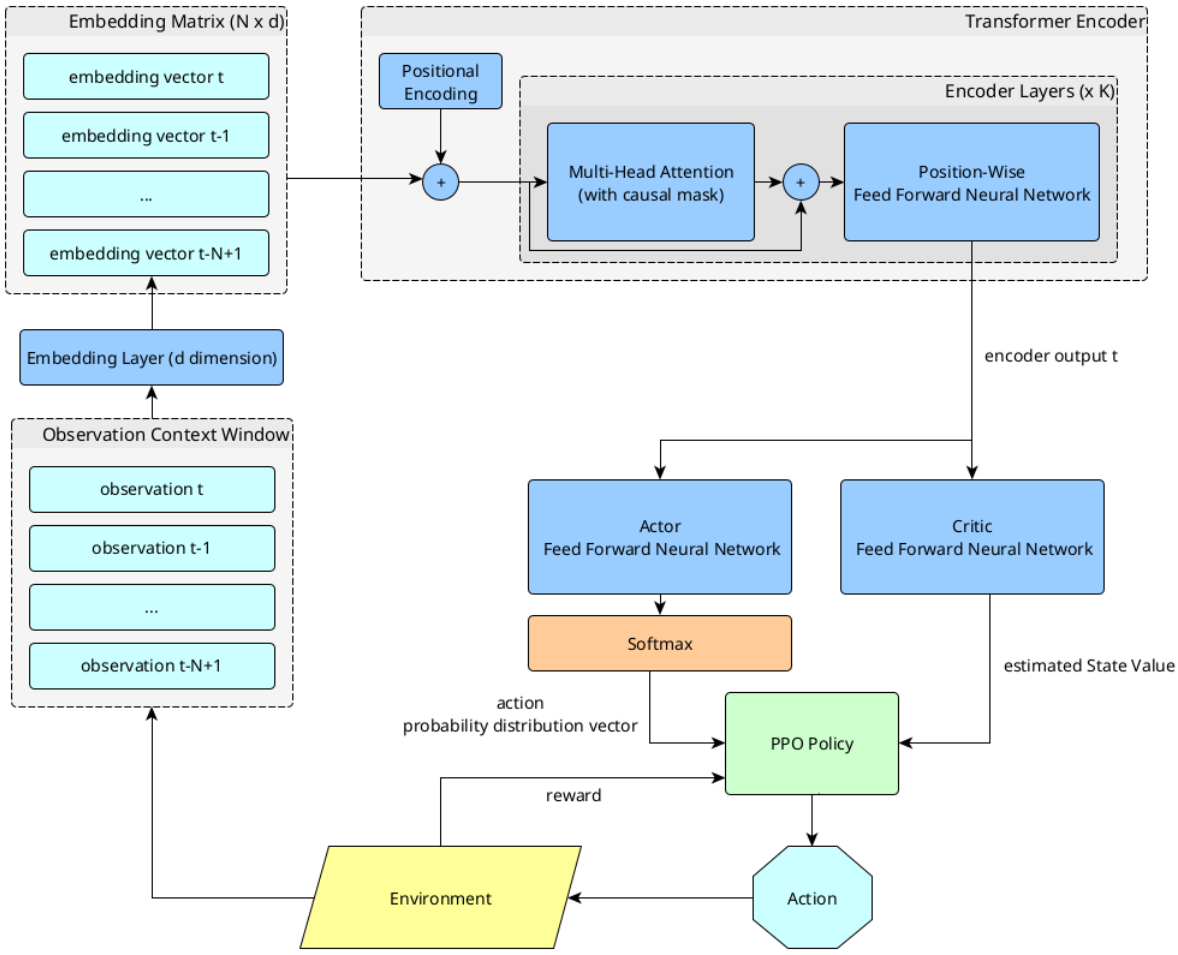



Figure 2.3: Model

In the bottom part of the figure, observations from the Environment are collected into an $N$-length context window. This represents the Agent's memory and aims to weakly but restore the Markov Property of the state space (section 1.2.6).

From these observation vectors, the *Embedding Layer* creates $d$ dimensional embedding vectors. Unlike NLP tasks, these embedding vectors are not created from words but from vectors containing the observations. This is essentially a linear layer that maps observations into a $d$ dimensional vector space. As part of the training process, the embedding layer also learns, so it produces a representation that is semantically meaningful for the downstream layers and helps the learning process.

The modified Transformer Encoder receives these embedding vectors as a sequence. Only the Encoder part of the original Transformer architecture is used with the purpose of capturing complex relationships between the elements of the memory sequence using the Attention mechanism and compressing it into a compact representation that will serve as input for the action selection. The Encoder consists $K$ Encoder Layers stacked after each other. The original Encoder Layer was modified similarly to the approach described in the "Transformers are meta-reinforcement learners" [40] paper. In that, a modified initialization (T-Fixup) [41] was used for stabilizing the Transformer for RL, allowing layer normalization to be omitted, so the encoder layer's core essentially consists only of the Multi-Head Attention with causal mask and the following position-wise Feed Forward Neural Network. Before the Encoder Layer, a sinusoid Positional Encoding vector is added same as described in the original Transformer paper [32].

The T-Fixup[41] initialization applied for the modified architecture as follows:

- Use Gaussian initialization $\mathcal{N}(0, d^{-\frac{1}{2}})$ for the input embeddings and scale by $(9K)^{-\frac{1}{4}}$, where $d$ represents the embedding dimension, $K$ represents the number of encoder layers stacked on each other.
- Initialize all other weight matrices in the Encoder using Xavier initialization [47].
- Scale *value* and output projection weight matrices in Attention blocks and in Encoder's position-wise Feed Forward Neural Networks (FFNN) by $0.67K^{-\frac{1}{4}}$.

The output of the stacked Encoder Layer consists of $N$ embedding vectors, one for each time step in the input sequence. But after the Encoder Layers, only the vector corresponding to the current time step $t$, is used. This vector contains information from previous time steps as well, as a result of the attention mechanism.

The PPO (see 1.2.5 for details) is an Actor-Critic RL algorithm. It trains both the Actor, which determines the action to be selected given an observation, and the Critic, which approximates the State Value used as a baseline. Both the Actor and Critic take the same embedding vector for time step $t$ as input for their Feed Forward Neural Networks (FFNN) from the common upstream Transformer Encoder layer.

As we use discrete actions, the output of the Actor, after the Softmax, is a discrete probability distribution that specifies the probability for each action of selecting them from the available action space. The output of the Critic is a floating-point number that represents the approximated State Value corresponding to the state representation encoded by the Encoder at time step $t$.

During the training, the PPO algorithm teaches both the FFNNs belonging to the Actor and Critic, as well as the shared stacked Transformer Encoder model and the Embedding Layer.

## 2.2 Glider Environment

### 2.2.1 Wind velocity field

In the thesis, the goal of the glider is to catch a thermal and use it to ascend to the highest possible altitude. Modeling a thermal that closely resembles reality would require a lot of computational resources, so a simplified model is used that still approximates the main pattern observable in reality.

To achieve this, synthetic turbulence and synthetic horizontal wind is added to the simple Gaussian model used as the basis for thermal generation. This produces turbulent, tilted thermals. The parameters of these thermals are randomly modified at the beginning of each episode, this helps the policy to generalize.

#### Thermal model

The vertical air velocity's horizontal distribution in a thermal is given by the bivariate Gaussian function:

$$w(x, y) = w_{max} \cdot \exp\left(-\left(\frac{(x - x_0)^2}{2\sigma_x^2} + \frac{(y - y_0)^2}{2\sigma_y^2}\right)\right) \tag{2.1}$$

where

- $w_{max}$ is the maximum lift at the center of the thermal.
- $x_0, y_0$ are the coordinates of the center of the thermal.

- $\sigma_x, \sigma_y$ are the standard deviations of the Gaussian. The same value is used for $\sigma_x$ and $\sigma_y$ ($\sigma_{xy}$). Also, a constant value of $k = 1.5$ is used to calculate the standard deviation for the desired thermal radius. For example, to obtain a thermal with a radius of $80m$, $\sigma_{xy} = 80/k$ is used. This ensures that most of the bell volume is contained within the specified radius.

The radius of the thermal also varies with altitude, and this variation is also given by a Gaussian distribution. The equation for this variation is shown below:

$$\sigma_{xy}(z) = \sigma_{(xy)_{max}} \cdot \exp\left(-\frac{(z - z_0)^2}{2 \cdot \sigma_z^2}\right) \tag{2.2}$$

where

- $\sigma_{(xy)_{max}}$ is the maximum radius.
- $z_0$ is the altitude where the radius is maximum.
- $\sigma_z$ is the standard deviation of the radius Gaussian distribution along the altitude axis ($z$).

The value of $\sigma_{xy}$ is then used in Equation 2.1 to obtain the radius of the thermal at a specific altitude.

The generated thermal using the Equation 2.1 and 2.2 can be seen on Figure 2.4.

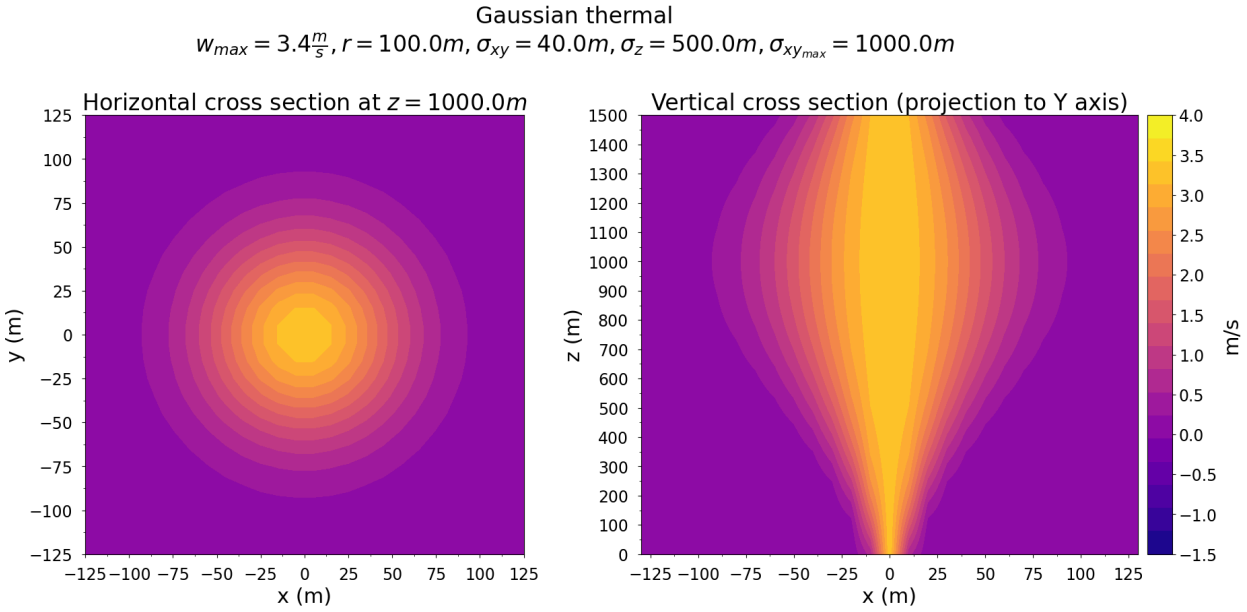



Figure 2.4: Basic Gaussian thermal

The horizontal components of the thermal wind velocity are zero:

$$\begin{aligned} u(x, y, z) &= 0 \\ v(x, y, z) &= 0 \end{aligned} \tag{2.3}$$

**Turbulence**

Generating turbulence is similar to what is described in the paper [48], with the difference that additive turbulence is used instead of multiplicative one. To do this, three uniform 3D grids are created that cover the space used for the simulation and the 3 dimensions of the *velocity field vector* $(u, v, w)$. For example, if the simulation space is $3000m \times 3000m \times 1500m$ and a resolution of $101 \times 101 \times 51$ is used, the grid points will be spaced $30m$ apart.

Then uniformly random numbers are generated between $[-1.0, 1.0]$ for these grid points. Then a 3-dimensional Gaussian filter is applied to smooth them along the points (with periodic boundary conditions). For example, for a smoothing radius of $30m$, $\sigma_{t_G} = 1$ is used for the space and resolution mentioned above.

The smoothed values are then normalized again to the $[-1.0, 1.0]$ range and multiplied by a multiplier $\eta_t > 0$ to determine the strength of the noise. This gives us the three 3D noise grids: $\xi_{t_u}, \xi_{t_v}, \xi_{t_w}$. The values between the grid points are obtained by weighted averaging based on the distances to the 8 surrounding grid points.

Using the generated noise field and the underlying Gaussian thermal, the turbulent wind velocities can be obtained using the formulas in Equation 2.4. Here, the vertical velocity $(w)$ is additive, and horizontal velocities are also additive but weighted by the vertical velocity at the given point. Note that $u(x, y, z)$ and $v(x, y, z)$ are both equal to zero.

$$\begin{aligned} u_t(x, y, z) &= u(x, y, z) + w(x, y, z) \cdot \xi_{t_u}(x, y, z) \\ v_t(x, y, z) &= v(x, y, z) + w(x, y, z) \cdot \xi_{t_v}(x, y, z) \\ w_t(x, y, z) &= w(x, y, z) + \xi_{t_w}(x, y, z) \end{aligned} \tag{2.4}$$

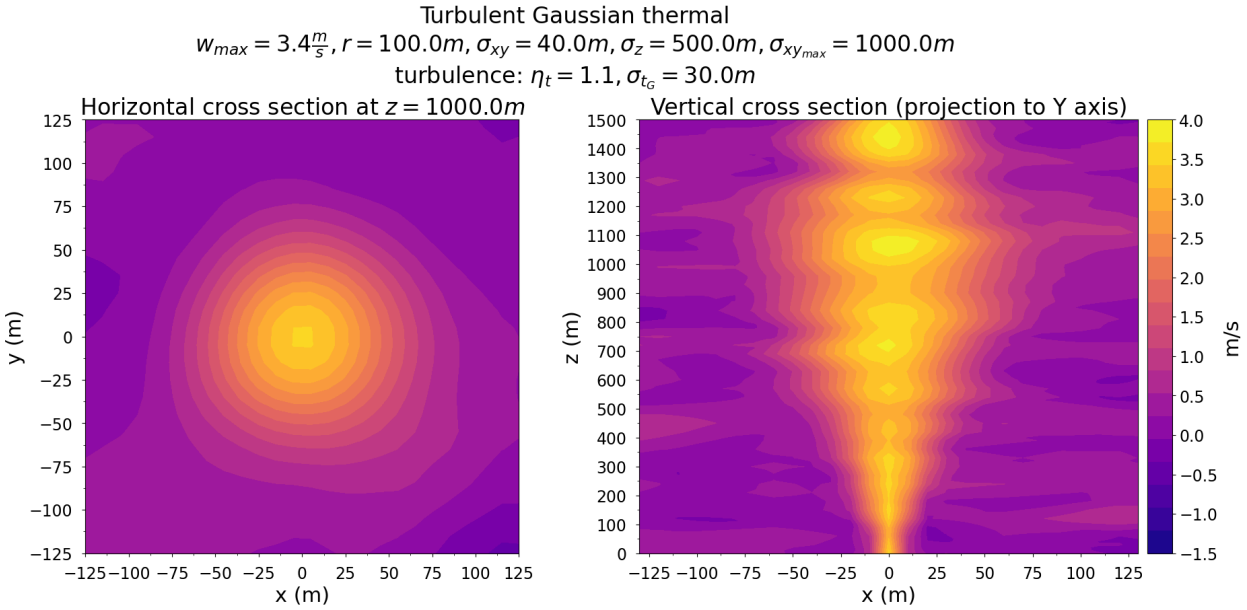



Figure 2.5: Gaussian thermal with turbulence

### Horizontal wind

For generating skewed thermals, a two-dimensional altitude-dependent horizontal wind is added to the turbulent thermal.

The altitude-dependent wind velocity distribution is given by the empirical formula named Wind profile power law [49]. The formula returns larger wind velocities for higher altitudes, which mimics real velocity distribution.

$$p(z) = p_r \left( \frac{z}{z_r} \right)^{\alpha} \tag{2.5}$$

where

- $p(z)$: The wind velocity at altitude $z$ $(m/s)$.
- $z_r$: The reference altitude, $z_r = 2m$ is used.
- $p_r$: The wind velocity at the reference altitude $(m/s)$.
- $\alpha$: Constant, $\alpha = 0.143$ is used.

The direction of the wind does not depend on the Coriolis force. A uniformly random direction is selected $(\theta)$, and the wind velocity $(p)$ is decomposed into two

horizontal coordinates. Then, similarly to the turbulence (see Section 2.2.1), a 2-dimensional noise $(\xi_{h_u}, \xi_{h_v})$ is added with a multiplier of $\eta_h$ and $\sigma_{h_G}$ as Gaussian filter sigma.

$$\begin{aligned} u_h(z) &= p(z) \cdot \cos(\theta) + \xi_{h_u}(z) \\ v_h(z) &= p(z) \cdot \sin(\theta) + \xi_{h_v}(z) \end{aligned} \tag{2.6}$$

Next, the horizontal displacement of the thermal core at a given altitude is determined by integration. As a simplification, the horizontal displacement only depends on the maximum vertical velocity at the thermal core:

$$\begin{aligned} x_0(z) &= \int_{i=0}^{z} \frac{u_h(i)}{w_{max}} di \\ y_0(z) &= \int_{i=0}^{z} \frac{v_h(i)}{w_{max}} di \end{aligned} \tag{2.7}$$

A turbulent thermal with added wind is shown in Figure 2.6.

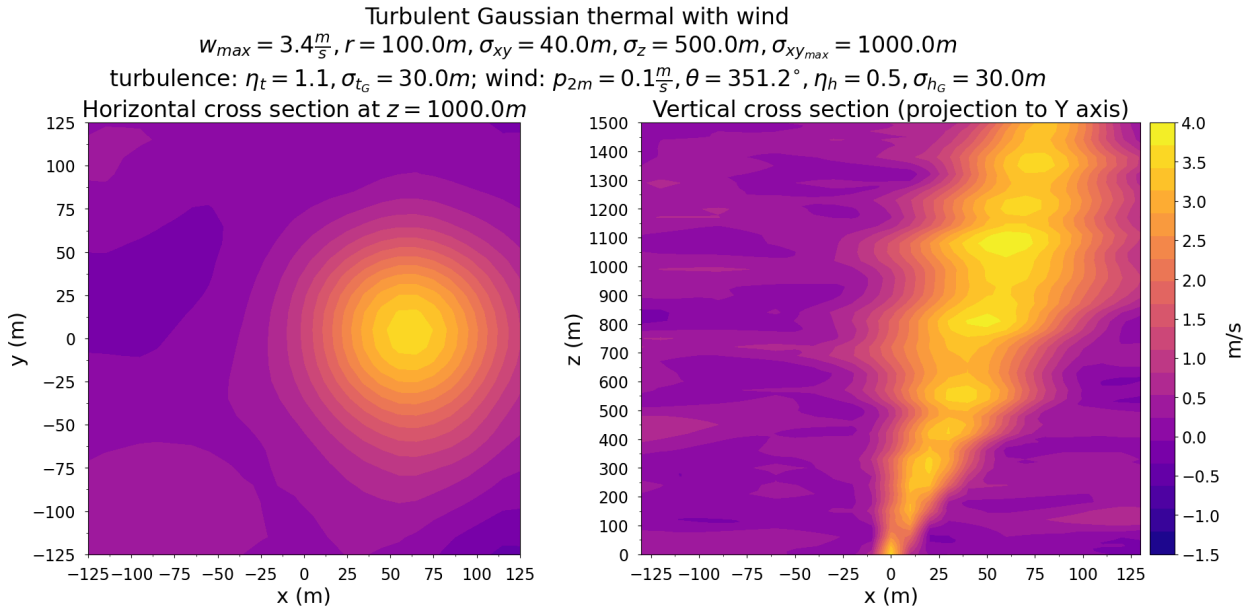



Figure 2.6: Gaussian thermal with turbulence and wind

### 2.2.2 Observation space

The glider receives its observation at uniform time intervals, for example, every second ($dt = 1s$). The observation space for the policy is a sequence of these observations using a fixed-size sliding context window. The size of this context window determines the memory capacity of the glider. Increasing the maximum sequence

length does not increase the number of model parameters due to the use of the Transformer architecture, but it increases the computational capacity during both the training and inference phases.

At the beginning of an episode, when the sequence has not yet filled the sliding window, the remaining sequence elements are filled with zeros so that the tensors passed to the model would have the same dimension. This is necessary because during training, batches may contain sequences of different lengths. However, the model does not consider the zero sequence elements due to the Causal Mask, which prevents it from using the observation at time $t + n$ at time $t$, including any completely zero elements. The output used is the Encoder's output corresponding to the $t$ observation, so the zero sequence elements do not interfere with the output.

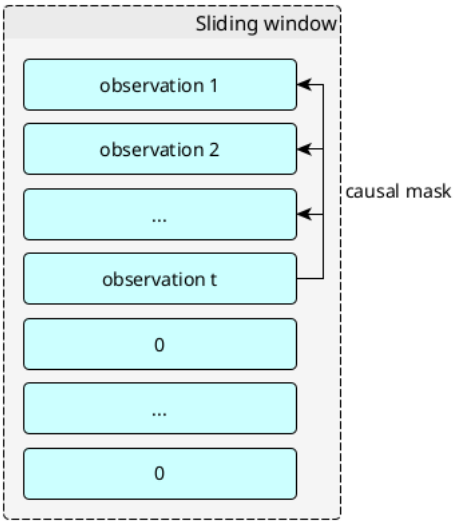



Figure 2.7: Sliding window with zero elements and causal mask

These observations consist of the following data:

- *Relative position*: The $[x, y, z]^{\mathrm{T}}$ position of the glider in meters relative to the first element of the glider's context window. The position corresponding to the first context window element will always be zero $[0, 0, 0]^{\mathrm{T}}$.
- *Roll*: The Earth-relative roll angle of the glider, measured in radians within the range $[-45°, 45°] = [-0.7853, 0.7853]$.
- *Velocity vector*: Three-dimensional velocity vector of the glider in $\frac{m}{s}$: $[v_x, v_y, v_z]^{\mathrm{T}}$.

So the observation is an $N \times d$ dimensional matrix, where $N$ is the maximum length of the context window, and $d = 7$ is the dimension of the observation vector corresponding to each sequence element.

### 2.2.3 Action space

The glider's control policy returns the discretized roll angle for the glider, which is chosen from the range $[-45°, 45°]$, using $N_a = 13$ discrete actions. Thus, at each time step, the action can take on one of the integer values $a_t = 0, 1, 2, \ldots, 12$. The prescribed roll angle for control is obtained as follows: $\phi_{deg} = -45° + (90°/(N_a - 1)) \cdot a_t$. This results in a division of 7.5° increments.

### 2.2.4 Reward

At each time step, the glider receives, as a reward, the current Earth-relative vertical velocity ($v_z$) measured in $\frac{m}{s}$. The vertical velocity is decreased by a penalty component $r_p$, which gives a larger penalty if the action difference from the previous action is larger. It motivates the agent to do a "finer" movement instead of abrupt bank angle changes. If the current altitude is larger than the previous maximum in the current episode ($z_{episode}$), then the difference will be added ($r_{alt}$). This provides a bonus for reaching a new highest altitude.

The reward for time step $t$ is given by Equation 2.8. Here a constant of 3 is used as a normalizer for smaller gradients.

$$r_t = \frac{v_z - r_p + r_{alt}}{3} \tag{2.8}$$

The penalty is given by Equation 2.9:

$$r_p = \frac{p_{max}}{N_a - 1} \cdot |a_t - a_{t-1}| \tag{2.9}$$

where

- $p_{max}$ represents the penalty of the maximum possible bank angle change.
- $N_a$ represents the count of discrete actions.
- $a_t$ means the discrete action index at time $t$.

The altitude bonus is given by Equation 2.10:

$$r_{alt} = \begin{cases} z - z_{episode} & if\ z > z_{episode} \\ 0 & otherwise \end{cases} \tag{2.10}$$

where

- $z$ represents the current altitude in meters.
- $z_{episode}$ represents the highest altitude in the current episode in meters.

### 2.2.5 Initial conditions

Each episode randomizes the initial conditions of the thermal and the glider. The following parameters of the thermal are set:

- The maximum radius ($\sigma_{xy_{max}}$) and vertical distribution ($\sigma_z$) of the thermal.
- The vertical velocity in the thermal's core ($w_{max}$).
- Turbulence parameters ($\eta_t, \sigma_{t_G}$).
- Wind parameters ($p_r, \theta, \eta_h, \sigma_{h_G}$).

The thermal is always located at the horizontal position $[0, 0]^T$, but since the glider only receives relative position information and the glider's thermal-relative initial position and orientation changes in every episode, it cannot overfit to the thermal's position.

The glider's initial conditions are the following:

- Initial altitude of the glider.
- The selection of the horizontal position is illustrated in Figure 2.8. The glider's position and direction are chosen in such a way that if the glider were to glide without changing direction from the beginning of the episode, it would encounter the thermal. The procedure does not take into account that the determined core position at the glider's initial position and the arrival position is not the same in windy conditions, however since the distance from the thermal is relatively small in the experiments, this is not a significant issue.

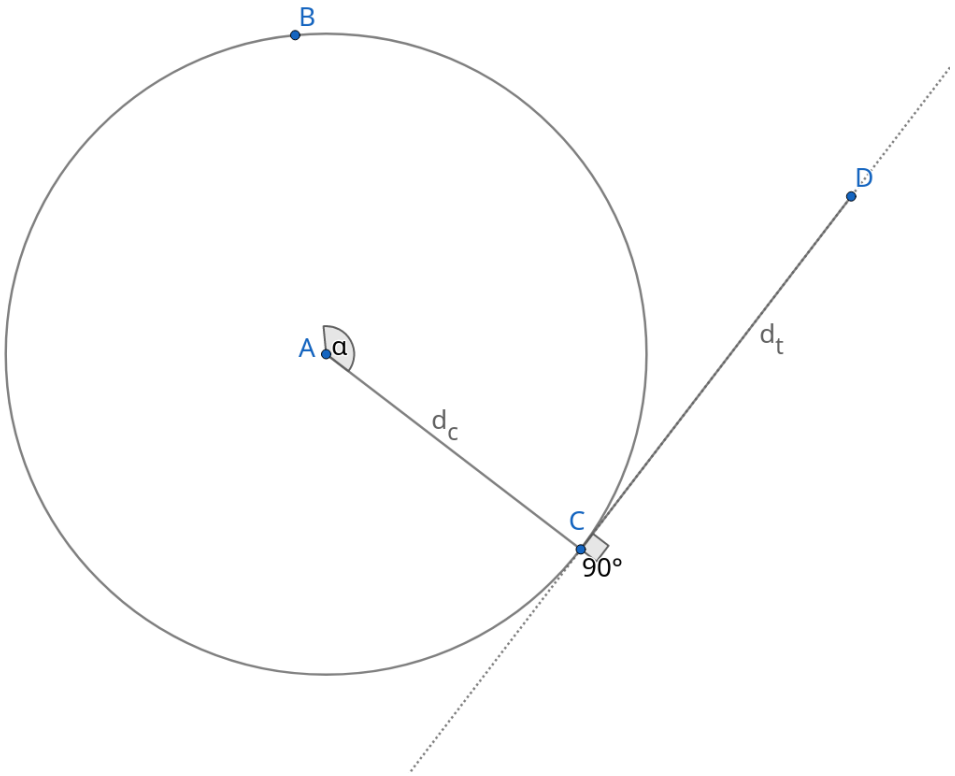



Figure 2.8: To select the horizontal position, first, the thermal's core position ($A$) at the glider's initial altitude is determined. Then, a random segment of length $d_c$ is drawn at an angle $\alpha$ from the core position, and from the end of this segment, another segment of length $d_t$ is drawn perpendicular to in a random direction from the two possible directions. The endpoint ($D$) of the $d_t$ segment determines the glider's initial position. The glider always looks towards the thermal in the direction collinear to the $d_t$ segment

### 2.2.6 Termination and truncation parameters

Episode termination is achieved when one of the following conditions is met:

- The glider reaches the maximum altitude ($z \geq z_{max}$).
- The glider reaches the minimum altitude corresponding to the ground level ($z \leq z_{min}$).

To prevent episodes from lasting too long, the following truncation parameters have been introduced:

- There is a maximum limit on the duration of an episode. This limit is specified in seconds ($t_{max}$).

- There is a maximum limit on how long the glider can fly without using the lift. Lift-free flight occurs when the vertical velocity is negative ($v_z < 0$). This limit is also specified in seconds ($t_{sink_{max}}$).
- There is a maximum limit on how far the glider can stay from the core. This distance is specified in meters ($d_{core_{max}}$).
- Additionally, truncation occurs if the glider exists the simulation area.

## 2.3 Simplified experiment method - Cart Pole

Before training in the glider environment, a simplified, lower-dimension problem was used for training based on the Cart Pole environment [50].

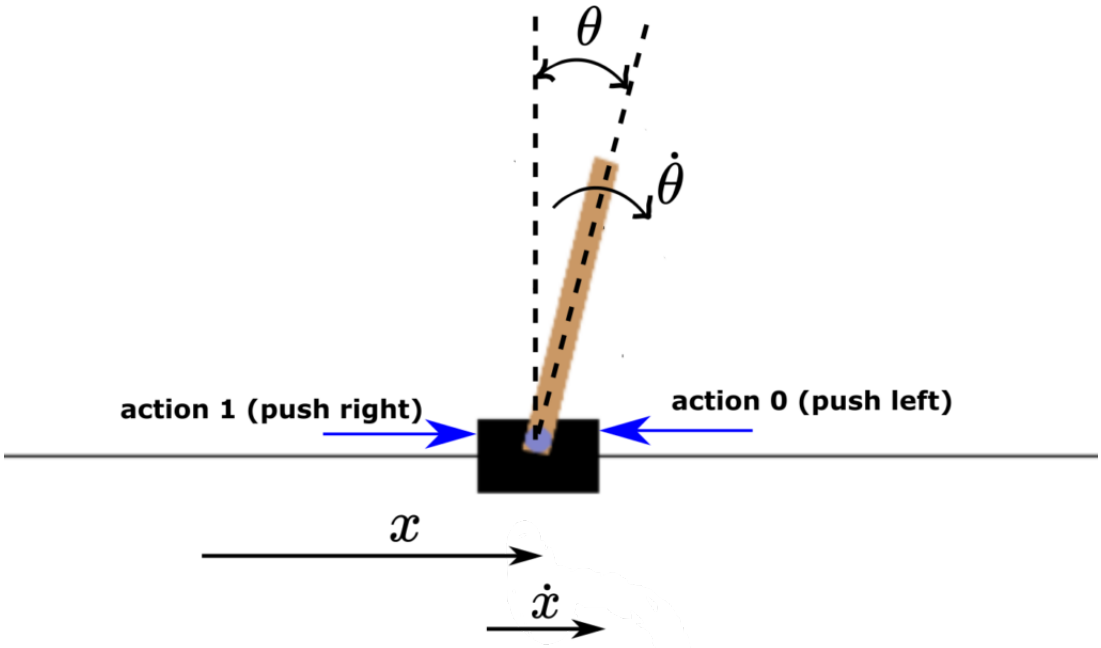



Figure 2.9: Observation and action space of the Cart Pole environment [51]

The Cart Pole environment's goal is to balance a pole attached to a cart by only moving the cart left or right. The original environment's observation space contained the cart's position, the cart's velocity, the pole's angle, and the pole's angular velocity. The original implementation gives a reward of 1 for every step, the termination occurs when the pole's angle reaches ±12° or the cart leaves the ±2.4 position range.

To make the environment similar to the thermal soaring problem, a position-specific reward was added, the position was removed from the observation space, and also the reward was added to the observation space. This modified environment was used to train the Transformer based network described in Section 2.1.

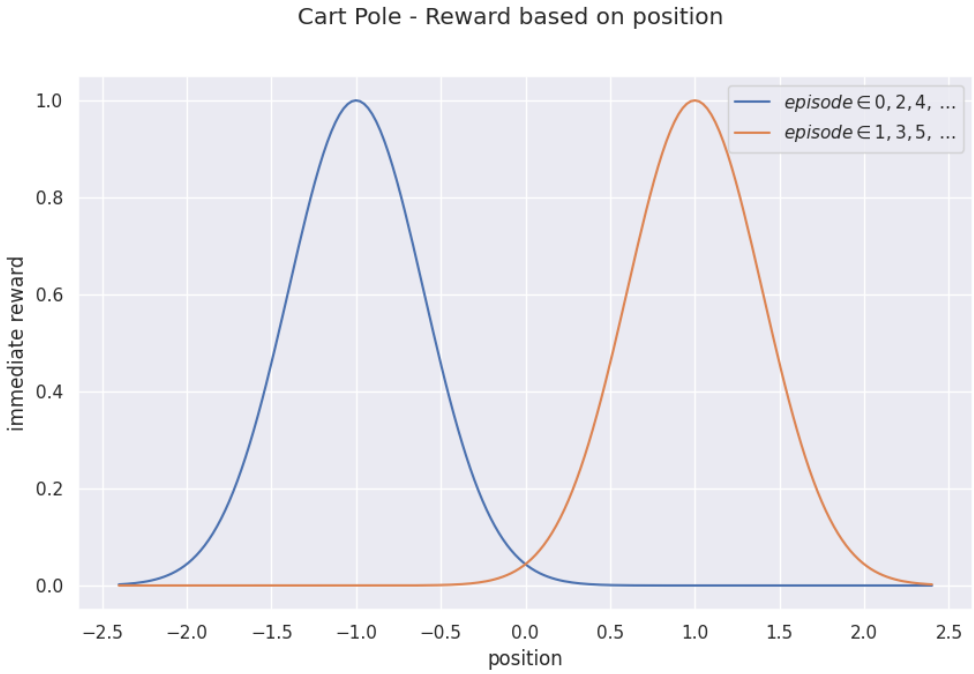



Figure 2.10: Modified position-dependent reward for the Cart Pole environment

The position-dependent reward is a Gaussian curve that makes the highest value at the target position ($\pm 1$). The policy in the modified Cart Pole environment receives this value as part of the observation space. In each episode, the target position alternates, as shown in Figure 2.10, to prevent the policy from overfitting to the target position. The Cart Pole receives the maximum reward for the episode if it goes to the target position as soon as possible and remains there while balancing the pole.

Without using memory, the policy only sees the current value of the curve, so it does not know which direction to start moving and proceed. While moving towards the target position, it must still need to balance the card, or the episode will terminate.

The glider also needs to find the thermal core and circle around it, making the modified Cart Pole environment a lower-dimensional analogue of the thermal soar-

ing problem. By refining the underlying model and training process using this environment, the training could be faster using a less computationally demanding environment.

## 2.4 Training pipeline

The training process and its related hyperparameters are shown in Figure 2.11.

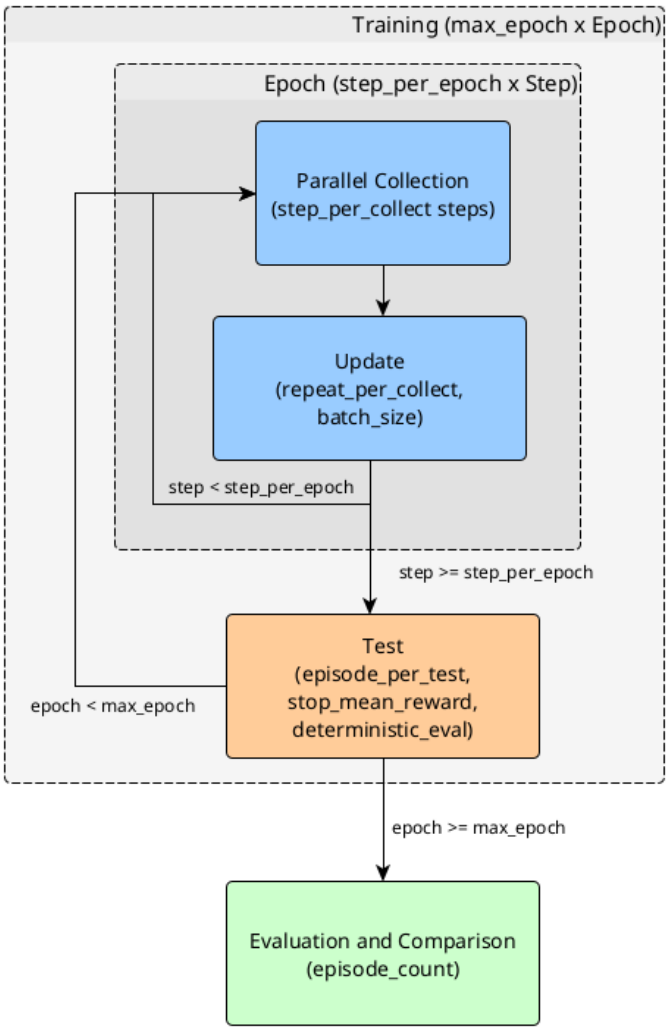



Figure 2.11: Workflow and hyperparameters of the training process

- The first step of the process is the *Parallel Collection*, where `step_per_collect` steps are collected using the current version of the Policy in a parallel manner for utilizing multi-core processors. During this phase, the actions are sampled from the action probability distribution returned by the Policy.

- The collected observations, actions, and rewards are then used to update the Policy in the *Update* phase with gradient ascent using the PPO algorithm (see Section 1.2.5). The update is performed in mini-batches of size `batch_size` to better utilize GPU and is repeated `repeat_per_collect` times. More repeats can help the algorithm learn more aspects from the collected steps. One batch item consists of a memory context window of the history, this is passed as input to the model (see Section 2.2.2).

- At the end of each Update process, the old steps collected by the previous version of the Policy are deleted, and the new steps collected by the updated version of the Policy are used for the next update. If the number of steps reaches `step_per_epoch`, an epoch is completed.

- At the end of each epoch, a *Test* is performed, which runs `episode_per_test` episodes. The mean reward obtained during the test is used for early stopping. If it reaches a specified `stop_mean_reward` value, the training is terminated. If `deterministic_eval` is used for the testing, the action with the highest probability is chosen instead of sampling from the action probability distribution. This gives a more stable and deterministic metric for the comparison of rewards from different training runs.

- If the training is not stopped by early stopping, it ends after `max_epoch` epochs.

- Then the model parameters corresponding with the highest $reward_{mean} - reward_{std}$ value obtained during Testing is loaded into the model for the *Evaluation and Comparison* phase. It runs `episode_count` episodes to determine the final result. The evaluation phase can compare multiple policies by giving the environments the same random seed, so the policies are compared with the same initial conditions.

# Chapter 3

# Experiments and results

I first used the simpler Cart Pole environment for stabilizing the training process and experimenting with the effects of the hyperparameters on the training process. Then, I transferred the knowledge gained here to the Glider Environment.

## 3.1 Hyperparameters

During the experiments, I configured the following hyperparameters:

**Model's parameter**

- Observation parameters:
  - `max_sequence_length`: The maximum length of the memory context window. (see Section 2.1)
- Transformer Encoder parameters (see Section 1.2.7 and 1.2.8):
  - `embedding_dim`: The dimensionality of the embedding vector. Vectors of this dimensionality will be passed between the Transformer Encoder layers.
  - `attention_head_num`: How many attention head is used.
  - `attention_internal_dim`: The dimensionality used in the attention heads for `key` and `value` vectors.
  - `transformer_ffnn_hidden_dim`: Parameter count for the Transformer Encoder's position-wise Feed Forward Neural Network.

  - `transformer_ffnn_dropout_rate`: Dropout rate for the Transformer Encoder's position-wise Feed Forward Neural Network.

- Actor and Critic parameters (see Section 2.1):
  - `actor_hidden_sizes`: Hidden sizes for the Actor's Feed Forward Neural Network. It predicts the Action Distribution.
  - `critic_hidden_sizes`: Hidden sizes for the Critic's Feed Forward Neural Network. It predicts the Value function's value.

**PPO algorithm's parameters**

- `discount_factor`: The discount factor for the discounted future reward ($\gamma$) (see Equation 1.12 in Section 1.2.4).
- `gae_lambda`: The exponential weighting factor ($\lambda$) for General Advantage Estimation (see Equations 1.32 and 1.33 in Section 1.2.5).
- `eps_clip`: The $\epsilon$ coefficient for the clipped surrogate function (see Equation 1.34 in Section 1.2.5).
- `value_clip`: Whether clipping is used for the Critic's Value Function's objective with the same `eps_clip` $\epsilon$ value.
- `dual_clip`: The $c$ coefficient for dual clipping [52].
- `ent_coef`: The Entropy Loss coefficient, which controls the weight of the Entropy Loss term in the PPO's objective function (see $c_2$ coefficient in Equation 1.36 in Section 1.2.5).
- `vf_coef`: The Value Function coefficient, which controls the weight of the Value Function Loss term in the PPO's objective function (see $c_1$ coefficient in Equation 1.36 in Section 1.2.5).
- `recompute_advantage`: Whether recompute Advantage Estimation after each repeat in the Update phase (see `repeat_per_collect` later in Training hyperparameters).

**Training hyperparameters (see Section 2.4 for details)**

- Collection / Update phase:
  - `step_per_collect`: Number of steps collected before each policy update.
  - `repeat_per_collect`: The number of repeats of the policy's parameter update after each Collection.
  - `batch_size`: Batch size for the policy Update.
  - `learning_rate`: The learning rate of the optimizer.
  - `step_per_epoch`: The number of steps used for the Collection and Update phase before testing.
- Test phase:
  - `episode_per_test`: Number of episodes used for determining test reward.
  - `stop_mean_reward`: Early stopping threshold for $reward_{mean} - reward_{std}$ value in Test phase.
  - `deterministic_eval`: Whether using deterministic policy in testing ($argmax$ instead of action distribution).
- `max_epoch`: Maximum number of epochs used for the whole training.

## 3.2 Other parameters

Adam optimizer was used for the training with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

## 3.3 Hardware

For the training, one computer used with the following hardware configuration:

- CPU: 1× `Intel® Core™ i9-12900F 16-Core, 24-Thread`
- GPU: 1× `ASUS Dual GeForce RTX™ 3060 V2 OC Edition 12GB`
- Memory: $32GB$

In the *Training* phase, 24 parallel environments were utilized on the CPU for the Collection phase. Then, in the Update phase, the collected data from the parallel environments were processed on the GPU.

## 3.4 Software

The Glider environment was implemented, and the Cart Pole environment was extended using the Gymnasium [53] (formerly OpenAI Gym [54]) framework. The training pipeline is implemented using the Tianshou RL framework [55]. It was utilized using a custom-developed configuration, training, and evaluation pipeline. The Transformer based architecture was implemented using vanilla PyTorch [56] and with the help of Tianshou's pre-created network modules. For the learning algorithm, Tianshou's PPO implementation was used. All of the experiments and their hyperparameters are logged using the Neptune experiment logging framework [57]. For hyperparameter optimization, Optuna [58] was used.

Artifacts of the project, such as simulation videos, will be made available on GitLab [59].

## 3.5 Cart Pole with Target Position

In this experiment, the cart had to move to either the $+1$ or $-1$ position using its memory, without directly observing the position, only the position-dependent reward function which was part of the observation space (see Section 2.3).

The following hyperparameters used for training:

| **Parameter** | **Parameter Value** |
|---|---|
| max_sequence_length | 10 |
| embedding_dim | 32 |
| attention_head_num | 1 |
| attention_internal_dim | 16 |
| transformer_ffnn_hidden_dim | 32 |
| transformer_ffnn_dropout_rate | 0.0 |
| actor_hidden_sizes | $[64, 64]$ |
| critic_hidden_sizes | $[64, 64]$ |
| discount_factor | 0.999 |
| gae_lambda | 0.9 |
| eps_clip | 0.2 |
| value_clip | *true* |
| dual_clip | 5.0 |
| ent_coef | 0.15 |
| vf_coef | 0.5 |
| recompute_advantage | $false$ |
| step_per_collect | 18000 |
| repeat_per_collect | 5 |
| batch_size | 64 |
| learning_rate | 0.001 |
| step_per_epoch | 36000 |
| episode_per_test | 20 |
| stop_mean_reward | 1400 |
| deterministic_eval | *true* |
| max_epoch | 100 |

Table 3.1: Hyperparameters used for Cart Pole training

The resulting model has a total of 19,731 parameters.

### 3.5.1 Cart Pole Training Stability and Duration

Multiple trainings were conducted with the same hyperparameters to examine the stability and required time for learning the policy.

Figure 3.1 shows the reward during the Training and also for intermediate Testings steps. This is averaged across 4 different training sessions.

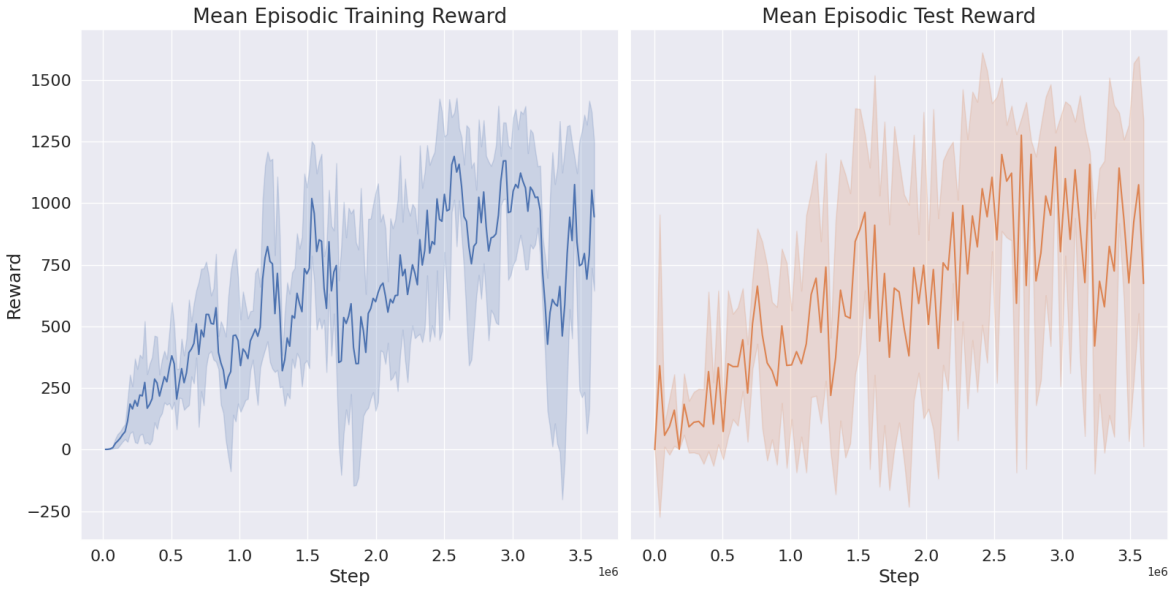



Figure 3.1: Mean and standard deviation of Episodic Mean Reward in Training and Test phases averaged along 4 different Cart Pole training. The Training plot (left) shows how the policy evolved after each policy update by the increasing episodic mean reward during the collection phase. The Test plot (right) shows the episodic mean reward achieved during the intermediate deterministic testing.

As the diagram shows however there is variance between the different training runs in the mean episodic training and test rewards, the reward increases in both plots. There are decreases too when because of the exploration, new types of solutions are tried because of the entropy coefficient (`ent_coef`).

The variance between the different training runs is larger in the Test reward ($std_{test_{avg}} = 399.4$) than the Training reward ($std_{train_{avg}} = 272.5$), because it uses `deterministic_eval`, which causes that if the action probability distribution is spread out, it has a larger probability of selecting a good action than in the deterministic selection where always the action with the largest probability is selected.

Table 3.2 shows the mean values of Training and Test metrics averaged along the different runs.

| **Metric** | **Value** |
|---|---|
| Best Test Reward | 1,380 ± 40 |
| Best Test Reward Std. (along episodes) | 60 ± 60 |
| Duration (s) | 2,700 ± 100 |
| Time in collector (s) | 660 ± 30 |
| Time in model update (s) | 1,610 ± 60 |
| Test time (s) | 470 ± 40 |
| Training steps | 3,500,000 ± 100,000 |
| Training episodes | 7,000 ± 1,000 |
| Training speed (step/s) | 1,560 ± 30 |
| Test steps | 2,200,000 ± 100,000 |
| Test episodes | 1,990 ± 70 |
| Test speed (step/s) | 4,600 ± 100 |

Table 3.2: Training and Test metrics averaged along 4 different training runs.

### 3.5.2 Cart Pole Evaluation Results

After the training, the results of a 100-episode evaluation can be shown at 3.2

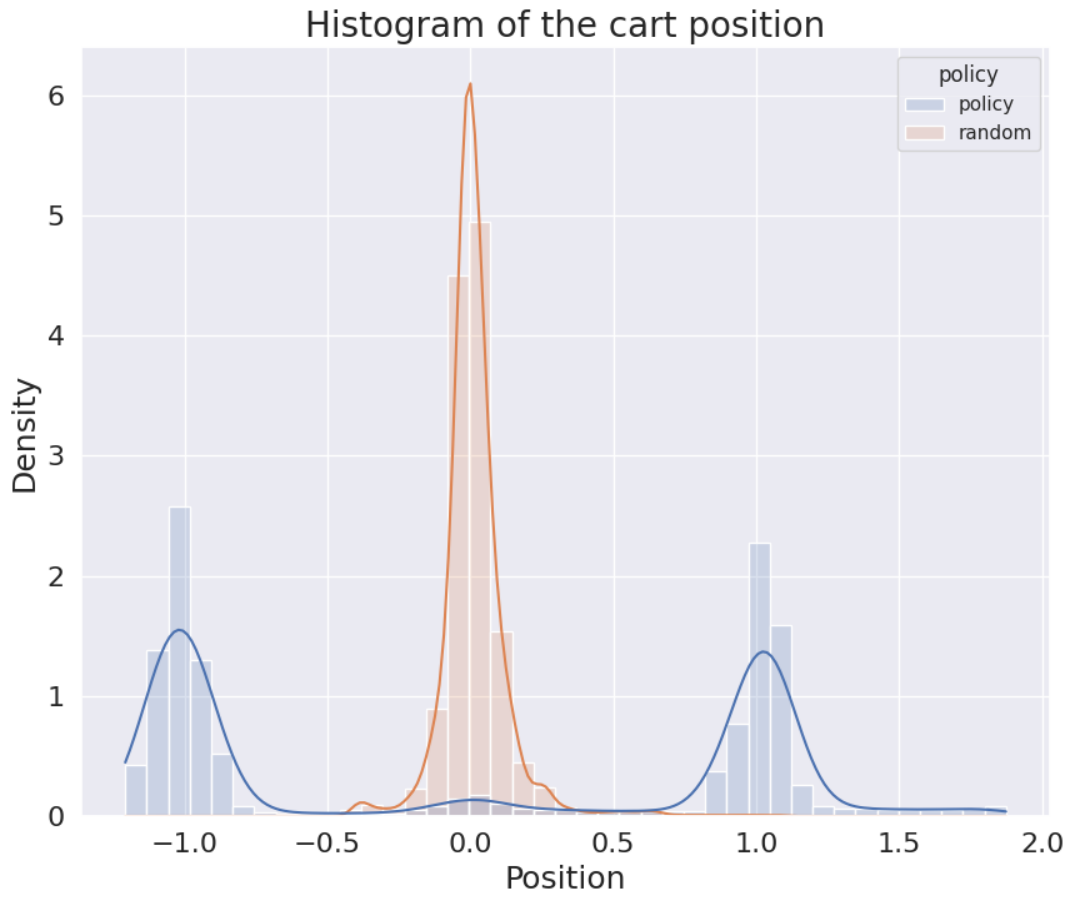



Figure 3.2: The distribution of the cart *Position* is shown by the result of the 100 episode evaluation. It is visible that the density is highest at around the target positions ($\pm 1$), indicating that the agent spent the most time there. The distribution is similar to the reward curve in Figure 2.10 (JS-divergence 0.138). The graph also shows the result of the random policy with orange, which was around 0, as the pole quickly fell.

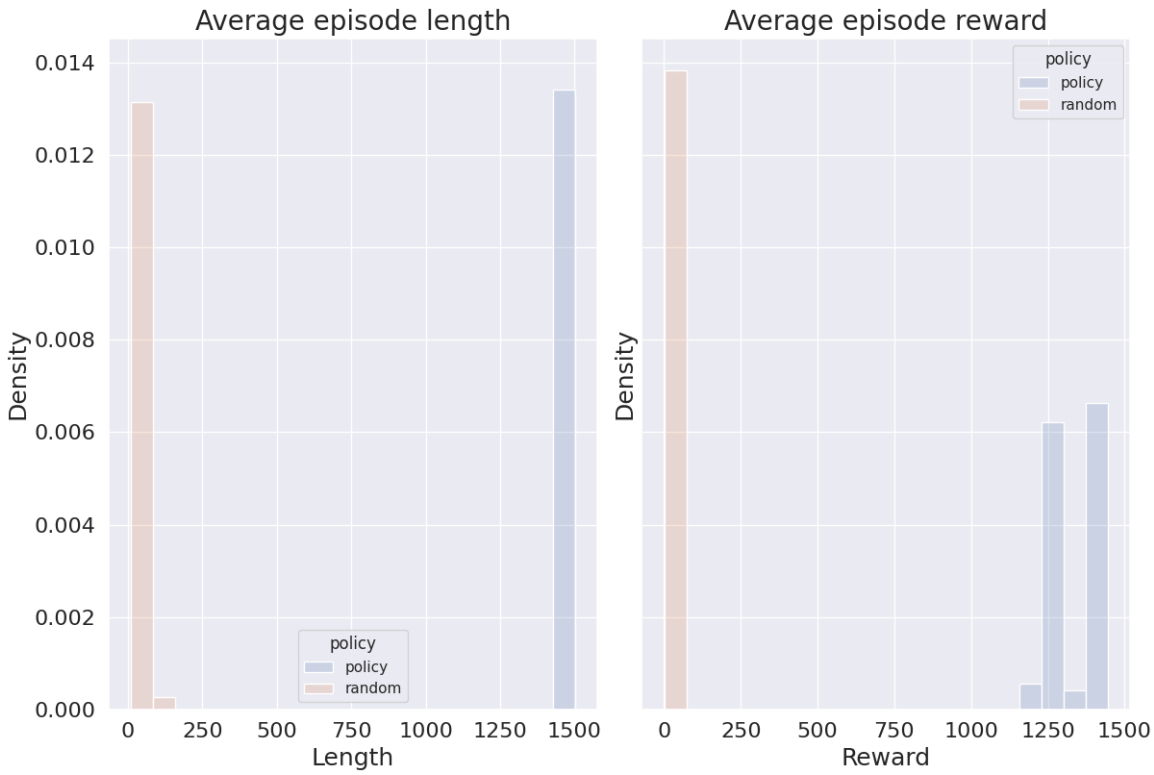



Figure 3.3: The distribution of the episode *Length* (left) and episode *Reward* (right) of the 100 episode evaluation. The trained Policy reached the maximum length of $1500 \pm 0$ in all of the 100 episodes (didn't fall) and the reward was $1,340 \pm 70$. The Random Policy's length was $30 \pm 20$ with a reward of $1.1 \pm 0.6$.

### 3.5.3 Knowledge Gained from the Cart Pole Environment

During the Cart Pole's training, it was observed that the Transformer-based network training was highly sensitive to hyperparameters. Automatic hyperparameter optimization was used using the Optuna [58] hyperparameter optimization framework with 146 different training runs. Although the number of training runs was not sufficient to find the optimal combination of the hyperparameters, and also not all of the hyperparameters passed to Optuna for optimization, the results obtained could provide valuable insights into which hyperparameters were worth tuning. Figure 3.4 shows the hyperparameter importance calculated by Optuna.

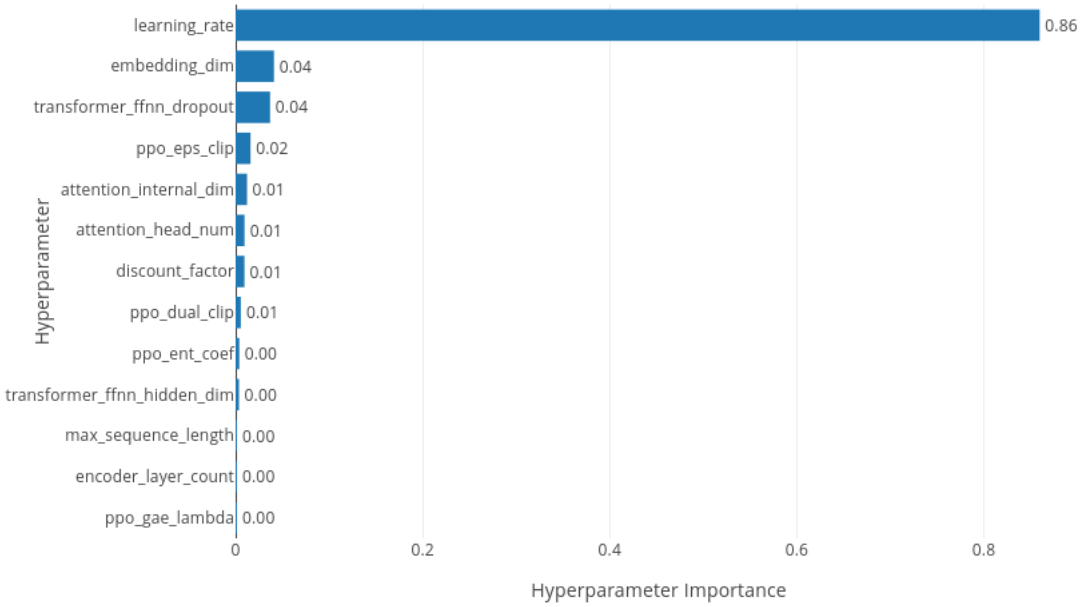



Figure 3.4: Hyperparameter importance calculated by Optuna using 146 training runs.

Using data acquired from Optuna and empirically, the following observations were made about the influence of different hyperparameters:

- Selection of learning rate and clipping parameters to avoid too large gradients were crucial (`eps_clip`, `value_clip`, `dual_clip`, `learning_rate`).
- Embedding dimension (`embedding_dim`): Between the Transformer Encoder layers, vectors of this dimension travel for each sequence element, and finally the vector that encodes the time step $t$ is also of this dimension. This contains the low-dimensional representation of the whole sequence weighted by attention. Therefore, if this dimension is too small, it cannot encode the sequence well.
- Entropy coefficient (`ent_coef`): If the entropy coefficient was too small, the policy explored very little and found a suboptimal local minimum. However, if it was too large, it continuously explored and did not settle in any one place.
- Maximum sequence length (`max_sequence_length`): When the sequence length was increased, it required a more complex model and larger embedding size to provide the required capacity of the network.
- Batch size (`batch_size`): When too low batch size was used, the gradient update couldn't receive representative information for the stable gradient update.

- Discount factor (`discount_factor`): When the discount factor was set too high (close to 1) for a task that provides immediate rewards, rather than just sparse rewards (like the Cart Pole environment), it could hinder the policy's ability to learn effectively because it overly prioritized future rewards.

## 3.6 Glider Environment

Using the insights and knowledge gained from the Cart Pole environment, it was easier to fine-tune the training of the Glider Environment. In the following chapter, I will present the experiments conducted in the glider simulator and their corresponding results.

### 3.6.1 Parameters of the Glider

The parameters of the glider used in simulation can be shown in Table 3.3 (see Section 1.2.3 for details).

| **Metric** | **Value** |
|---|---|
| Mass ($m$) | $3kg$ |
| Wing area ($S$) | $0.6m^2$ |
| Air density ($\rho$) | $1.225\frac{kg}{m^3}$ |
| Lift coefficient ($CL$) | 1.0 |
| Drag coefficient ($CD$) | 0.08684 |
| Gravitational acceleration ($g$) | $9.81\frac{m}{s^2}$ |

Table 3.3: Physical and aerodynamic parameters of the simulated glider

### 3.6.2 Termination and Truncation Parameters

The parameters of the simulation can be seen in Table 3.4 (see Section for details 2.2.6).

| Metric | Value |
|---|---|
| Success altitude ($z_{max}$) | $1000m$ |
| Minimum altitude ($z_{min}$) | $10m$ |
| Episode time limit ($t_{max}$) | $20min.$ |
| Maximum time without lift ($t_{sink_{max}}$) | $3min.$ |
| Maximum distance from core ($d_{core_{max}}$) | $700m$ |
| Simulation box $((x, y, z)_{low}, (x, y, z)_{high})$ | $(-1000m, -1000m, 0m)$, $(1000m, 1000m, 1000m)$ |

Table 3.4: Termination and truncation parameters of the simulation

### 3.6.3 Memory Context Window Length and Time Resolution

With a maximum sequence length of 30 and a time resolution of $1Hz$, the Agent could use information from a context window spanning the past 30 seconds.

### 3.6.4 Hyperparameters

The following hyperparameters used for the training:

| Parameter | Parameter Value |
|---|---|
| max_sequence_length | 30 |
| embedding_dim | 64 |
| attention_head_num | 4 |
| attention_internal_dim | 16 |
| transformer_ffnn_hidden_dim | 64 |
| transformer_ffnn_dropout_rate | 0.0 |
| actor_hidden_sizes | [64, 64] |
| critic_hidden_sizes | [64, 64] |
| discount_factor | 0.99 |
| gae_lambda | 0.9 |
| eps_clip | 0.2 |
| value_clip | $false$ |
| dual_clip | 5.0 |
| ent_coef | 0.01 |
| vf_coef | 0.5 |
| recompute_advantage | $true$ |
| step_per_collect | 22800 |
| repeat_per_collect | 5 |
| batch_size | 256 |
| learning_rate | 0.0003 |
| step_per_epoch | 45600 |
| episode_per_test | 10 |
| stop_mean_reward | – |
| deterministic_eval | $false$ |
| max_epoch | 200 |

Table 3.5: Hyperparameters used for Glider training

The resulting model has a total of 116,878 parameters.

Here, a more complex network was used than for the Cart Pole environment. The parameter `deterministic_eval` was set to $false$ due to the stochastic nature of the environment. In order to stabilize the learning process, a larger `batch_size` of 256 and a smaller `discount_factor` of 0.99 were used for the environment. The value function clipping specified as `value_clip` was disabled because it was slowing down the training, presumably due to the larger reward values which caused larger gradients.

### 3.6.5 Glider Training Stability and Duration

Similarly to the Cart Pole environment, multiple trainings were conducted with the same hyperparameters.

Figure 3.5 shows the reward during the Training and also for intermediate Testing steps. This is averaged across 4 different training sessions. The mean of the Training reward variance was $std_{train_{avg}} = 48.7$, and the mean of the Test reward variance was $std_{test_{avg}} = 62.6$.

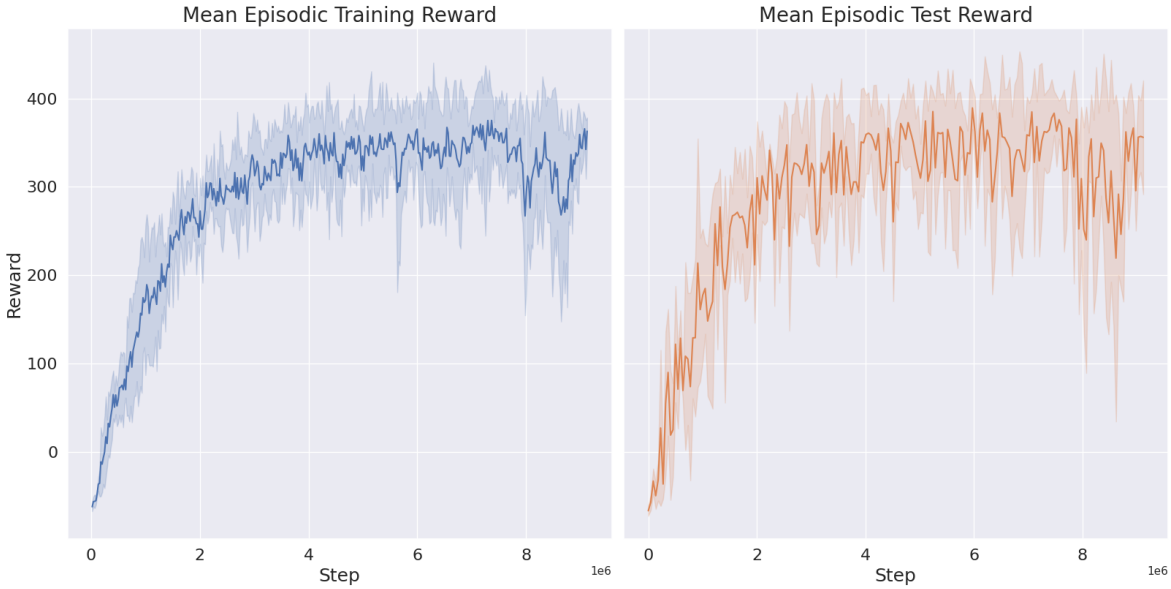



Figure 3.5: Mean and standard deviation of Episodic Mean Reward in Training and Test phases averaged along 4 different *Glider* training. The Training plot (left) shows how the policy evolved after each policy update by the increasing episodic mean reward during the collection phase. The Test plot (right) shows the episodic mean reward achieved during the intermediate stochastic policy testing.

Table 3.6 shows the mean values of Training and Test metrics averaged along the different runs.

| | Value |
|---|---|
| Best Test Reward | 405 ± 4 |
| Best Test Reward Std. (along episodes) | 13 ± 3 |
| Duration (s) | 21,400 ± 300 |
| Time in collector (s) | 13,500 ± 300 |
| Time in model update (s) | 6,710 ± 30 |
| Test time (s) | 1,170 ± 50 |
| Training steps | 9,120,000 ± 0 |
| Training episodes | 34,000 ± 1,000 |
| Training speed (step/s) | 451 ± 7 |
| Test steps | 550,000 ± 20,000 |
| Test episodes | 2,010 ± 0 |
| Test speed (step/s) | 468 ± 3 |

Table 3.6: Training and Test metrics averaged along 4 different Glider training runs.

### 3.6.6 Glider Experiments

#### Selection of the best parameters

After training, the parameter set with the highest test $mean - std$ reward value was selected from the training runs.

$$\theta = \arg \max_{\theta_{i,j} \;\; i \in [0...4], j \in test \;\; in \;\; run_i} (test\_reward_{mean}(\theta_{i,j}) - test\_reward_{std}(\theta_{i,j})) \quad (3.1)$$

#### Compared policies

The parameter set was loaded and evaluated in four different weather conditions using 100-episode evaluations. Three different policies were compared during the evaluation. The initial conditions were randomized using the same seed for the different policies, so they observed the same 100 episodes. However, the episodes themselves were different from each other.

The evaluated policies were the following:

- Trained policy: The policy obtained from the training process.
- Random policy: A policy that selects actions uniformly at random.
- Reichmann rules [60]: A policy used by glider pilots helping them centering the thermal. This is based on the following simple rules:

- If the vertical velocity increases, decrease the bank angle (change the bank angle towards the left).
- If the vertical velocity decreases, increase the bank angle (change the bank angle towards the right).
- If the climb rate remains the same, do not change the bank angle.

To check these conditions, the vertical velocity of the glider ($v_z$) at the current time ($t$) is compared to the mean of the previous vertical velocities in the previous `max_sequence_length-1` ($N-1$) length context window. Any control was only applied if the vertical velocity was greater than $-0.1\frac{m}{s}$, and any difference was considered a change only if the magnitude of the change was greater than $\epsilon = 0.1$. These parameters are empirically selected to give better control for the Reichmann policy.

$$\left| v_{z_t} - \frac{1}{N-1} \sum_{i=t-N+1}^{t-1} v_{z_i} \right| > \epsilon \tag{3.2}$$

**Weather initial conditions**

There were four different conditions for the experiments:

- *Experiment 1 (Thermal)*: Moderate thermal, no turbulence, no wind.
- *Experiment 2 (Thermal with turbulence)*: Moderate thermal, light turbulence, no wind.
- *Experiment 3 (Thermal with wind)*: Moderate thermal, no turbulence, light wind.
- *Experiment 4 (Thermal with turbulence and wind)*: Moderate thermal, light turbulence, light wind.

You can find the initial conditions for the 100-episode evaluation in Table 3.7 and Table 3.8, along with the corresponding figures that depict the thermals.

The RL Policy was trained using the weather conditions corresponding to *Experiment 4 (Thermal with turbulence and wind)*.

**Parameters of *Experiment 1 (Thermal)* and *Experiment 2 (Thermal with turbulence)***

| | Experiment 1 | Experiment 2 |
|---|---|---|
| Max thermal radius $[m]$ $(\sigma_{xy_{max}} \cdot 1.5)$ | 150 ± 10 | 150 ± 10 |
| Altitude of maximum radius $[m]$ $(z_0)$ | 1,000 ± 20 | 1,000 ± 20 |
| Std. of radius Gaussian $[m]$ $(\sigma_z)$ | 1,000 ± 40 | 1,000 ± 40 |
| Thermal max. vertical velocity $[\frac{m}{s}]$ $(w_{max})$ | 3.9 ± 0.4 | 4.0 ± 0.4 |
| Initial altitude of the glider $[m]$ | 400 ± 20 | 400 ± 20 |
| Turbulence noise multiplier $(\eta_t)$ | - | 0.5 ± 0.1 |
| Turbulence grid spacing $[m]$ | - | 20 ± 0 |
| Turbulence noise Gaussian filter std. $[m]$ $(\sigma_{t_G})$ | - | 10 ± 2 |
| Horizontal wind at $2m$ $[\frac{m}{s}]$ $(p_r)$ | - | - |
| Horizontal wind noise multiplier $(\eta_h)$ | - | - |
| Horizontal wind grid spacing $[m]$ | - | - |
| Horizontal wind noise Gaussian filter Std. $[m]$ $(\sigma_{h_G})$ | - | - |

Table 3.7: Initial conditions of the 100-episode evaluation for *Experiment 1 (Thermal)* and *Experiment 2 (Thermal with turbulence)*

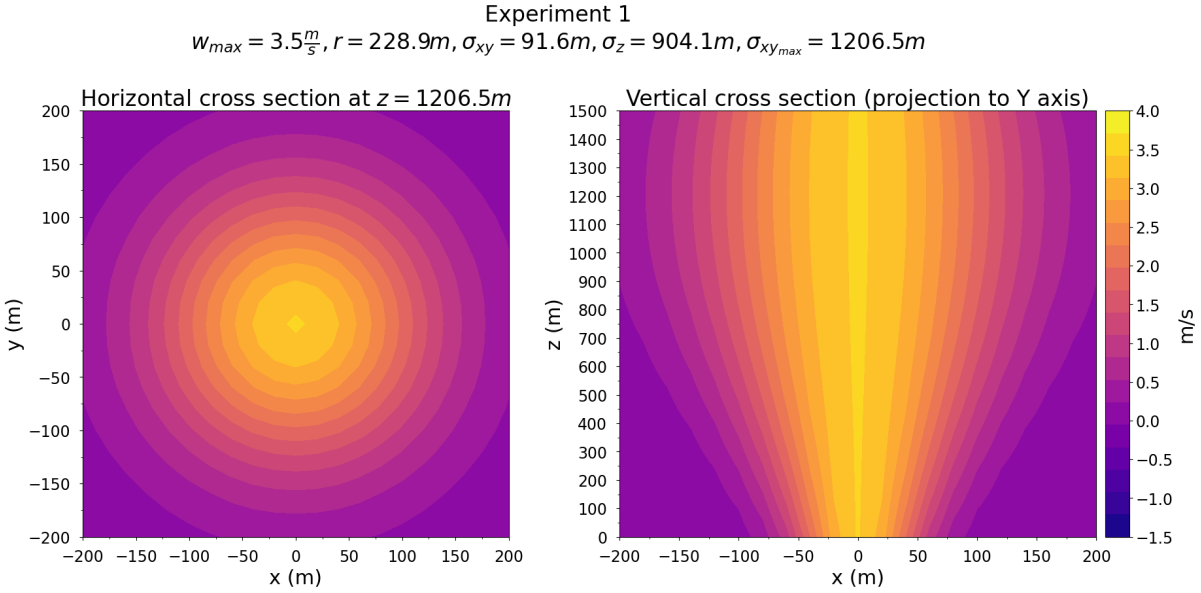



Figure 3.6: Sample thermal for *Experiment 1 (Thermal)*

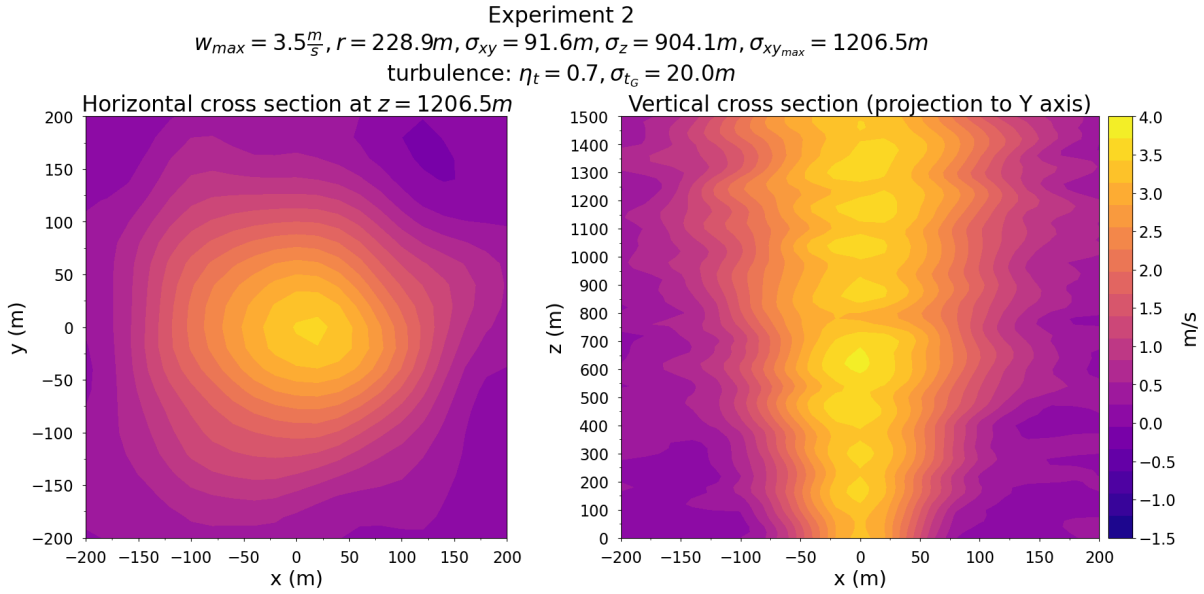



Figure 3.7: Sample thermal for *Experiment 2 (Thermal with turbulence)*

**Parameters of *Experiment 3 (Thermal with wind)* and *Experiment 4 (Thermal with turbulence and wind)***

| | Experiment 3 | Experiment 4 |
|---|---|---|
| Max thermal radius $[m]$ $(\sigma_{xy_{max}} \cdot 1.5)$ | 150 ± 10 | 150 ± 10 |
| Altitude of maximum radius $[m]$ $(z_0)$ | 1,000 ± 20 | 1,000 ± 20 |
| Std. of radius Gaussian $[m]$ $(\sigma_z)$ | 1,000 ± 40 | 1,000 ± 40 |
| Thermal max. vertical velocity $[\frac{m}{s}]$ $(w_{max})$ | 4.0 ± 0.4 | 4.0 ± 0.4 |
| Initial altitude of the glider $[m]$ | 400 ± 20 | 400 ± 20 |
| Turbulence noise multiplier $(\eta_t)$ | - | 0.5 ± 0.1 |
| Turbulence grid spacing $[m]$ | - | 20 ± 0 |
| Turbulence noise Gaussian filter std. $[m]$ $(\sigma_{t_G})$ | - | 10 ± 3 |
| Horizontal wind at $2m$ $[\frac{m}{s}]$ $(p_r)$ | 0.10 ± 0.02 | 0.10 ± 0.02 |
| Horizontal wind noise multiplier $(\eta_h)$ | 0.5 ± 0.2 | 0.5 ± 0.2 |
| Horizontal wind grid spacing $[m]$ | 20 ± 0 | 20 ± 0 |
| Horizontal wind noise Gaussian filter Std. $[m]$ $(\sigma_{h_G})$ | 10 ± 4 | 10 ± 4 |

Table 3.8: Initial conditions of the 100-episode evaluation for *Experiment 3 (Thermal with wind)* and *Experiment 4 (Thermal with turbulence and wind)*

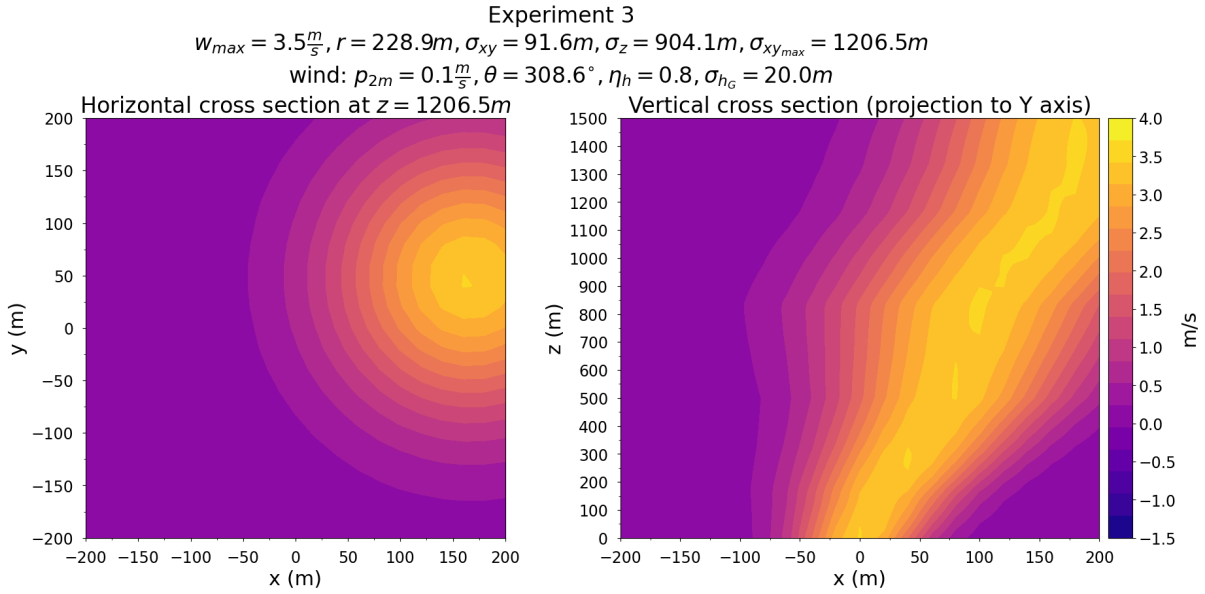



Figure 3.8: Sample thermal for *Experiment 3 (Thermal with wind)*

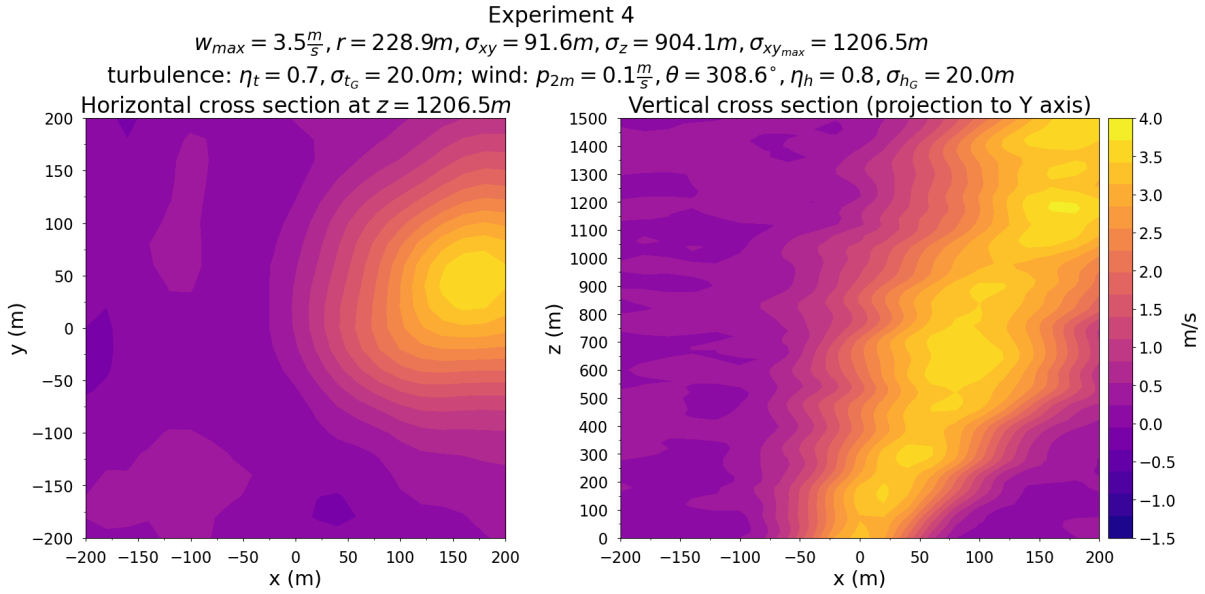



Figure 3.9: Sample thermal for *Experiment 4 (Thermal with turbulence and wind)*

### 3.6.7 Glider Experiment Results

The results of the four different experiments can be seen in Table 3.9-3.9.

In these tables:

- *Success rate* represents what percentage of the episodes reached the maximum (success) altitude ($z_{max}$).

- *Episode reward*, *Episode length*, *Maximum time without lift*, *Minimum altitude*, *Average altitude*, *Maximum altitude* are collected for each individual episode. These values are then combined and summarized for a set of 100 different episodes.
- *Average distance from core*, *Average vertical velocity*, *Average horizontal wind velocity* are calculated by considering all the steps taken in the 100 episodes.

**Results of *Experiment 1 (Thermal)***

The results of the experiment can be seen in Table 3.9.

| | Policy | Reichmann | Random |
|---|---|---|---|
| Success rate | 93.0% | 49.0% | 0.0% |
| Episode reward | 400 ± 100 | 200 ± 200 | -60 ± 30 |
| Episode length $[s]$ | 280 ± 60 | 1,000 ± 200 | 200 ± 100 |
| Maximum time without lift $[s]$ | 40 ± 30 | 40 ± 40 | 160 ± 40 |
| Average distance from core $[m]$ | 100 ± 100 | 150 ± 30 | 400 ± 100 |
| Average vertical velocity $[\frac{m}{s}]$ | 2 ± 1 | 0.3 ± 0.4 | -0.6 ± 0.6 |
| Minimum altitude $[m]$ | 370 ± 30 | 360 ± 50 | 280 ± 50 |
| Average altitude $[m]$ | 610 ± 80 | 500 ± 100 | 340 ± 30 |
| Maximum altitude $[m]$ | 1,000 ± 200 | 800 ± 300 | 410 ± 30 |
| Average horizontal wind velocity $[\frac{m}{s}]$ | 0 ± 0 | 0 ± 0 | 0 ± 0 |

Table 3.9: *Experiment 1 (Thermal)* result

It can be seen that in the absence of wind and turbulence, the trained Policy achieves a 93% *Success rate*. Although the Reichmann rules only achieve a 49% *Success rate*, it yields significantly higher *Episode reward* than the Random policy (t-test, $N = 100$, $p < 10^{-27}$).

The distribution of *Average distance from core* and *Altitude* can be seen in Figure 3.10.

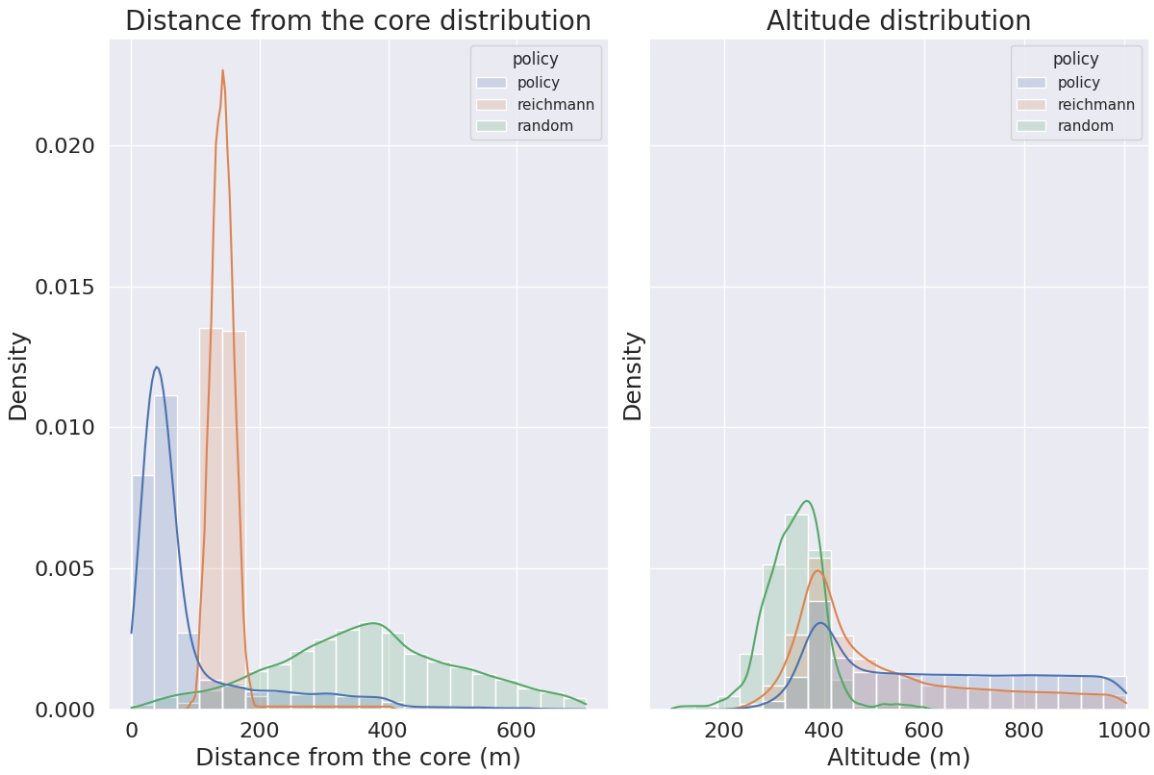



Figure 3.10: *Average distance from core* and *Altitude* distribution for *Experiment 1 (Thermal)*

As can be seen in the left distribution in Figure 3.10, the trained Policy managed to stay closest to the thermal core. Although the Reichmann rules did not generally come as close (t-test p-value $< 10^{-300}$), it significantly outperformed the random policy (t-test p-value $< 10^{-100}$).

### Results of *Experiment 2 (Thermal with turbulence)*

The results of the experiment can be seen in Table 3.10.

| | Policy | Reichmann | Random |
|---|---|---|---|
| Success rate | 90.0% | 29.0% | 0.0% |
| Episode reward | 400 ± 100 | 200 ± 200 | -70 ± 30 |
| Episode length $[s]$ | 280 ± 60 | 900 ± 300 | 200 ± 100 |
| Maximum time without lift $[s]$ | 50 ± 50 | 80 ± 70 | 160 ± 40 |
| Average distance from core $[m]$ | 100 ± 100 | 150 ± 40 | 400 ± 100 |
| Average vertical velocity $[\frac{m}{s}]$ | 2 ± 1 | 0.3 ± 0.5 | -0.6 ± 0.6 |
| Minimum altitude $[m]$ | 360 ± 40 | 360 ± 50 | 270 ± 50 |
| Average altitude $[m]$ | 600 ± 100 | 500 ± 100 | 340 ± 30 |
| Maximum altitude $[m]$ | 900 ± 200 | 700 ± 300 | 400 ± 30 |
| Average horizontal wind velocity $[\frac{m}{s}]$ | 0.3 ± 0.3 | 0.1 ± 0.1 | 0.0 ± 0.1 |

Table 3.10: *Experiment 2 (Thermal with turbulence)* result

When turbulence was added to the thermal, the Reichmann rules achieved a lower *Success rate*. This is consistent with the findings examined in the paper [48]. But, the *Episode reward* was similar to that of Experiment 1, and the *Episode length* remained relatively high, suggesting that the Reichmann rules spent a significant amount of time near the thermals (otherwise the episode would have been terminated), but did not reach the maximum altitude as frequently as in Experiment 1.

The trained Policy's success rate decreased slightly from 93% to 90%, but the *Episode reward* remained similar to the Experiment 1's.

The distribution of *Average distance from core* and *Altitude* can be seen in Figure 3.11.

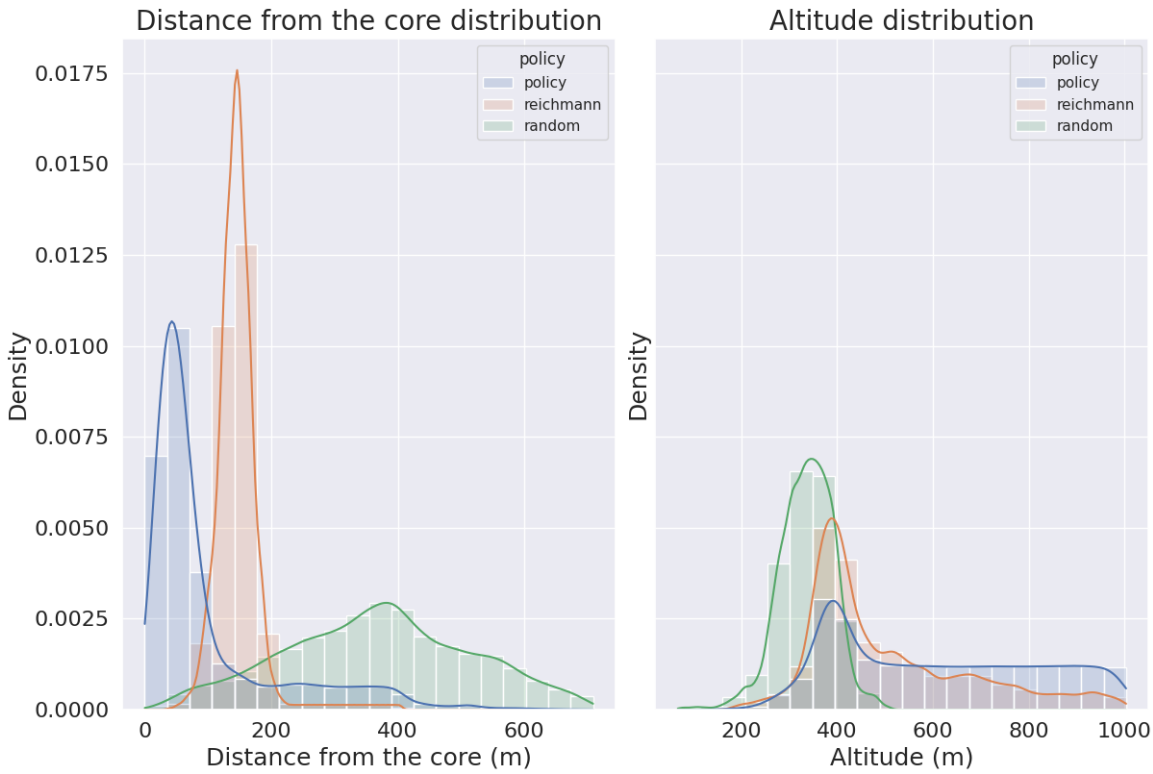



Figure 3.11: *Average distance from core* and *Altitude* distribution for *Experiment 2 (Thermal with turbulence)*

Both the distances from the core and altitude distributions are very similar to Experiment 1's distributions.

### Results of *Experiment 3 (Thermal with wind)*

The results of the experiment can be seen in Table 3.11.

| | Policy | Reichmann | Random |
|---|---|---|---|
| Success rate | 92.0% | 25.0% | 0.0% |
| Episode reward | 400 ± 100 | 100 ± 200 | -60 ± 30 |
| Episode length $[s]$ | 280 ± 70 | 500 ± 200 | 200 ± 100 |
| Maximum time without lift $[s]$ | 50 ± 40 | 140 ± 70 | 150 ± 40 |
| Average distance from core $[m]$ | 100 ± 100 | 160 ± 60 | 400 ± 100 |
| Average vertical velocity $[\frac{m}{s}]$ | 2 ± 1 | 0.2 ± 0.8 | -0.6 ± 0.6 |
| Minimum altitude $[m]$ | 370 ± 40 | 340 ± 50 | 290 ± 50 |
| Average altitude $[m]$ | 600 ± 90 | 500 ± 100 | 350 ± 40 |
| Maximum altitude $[m]$ | 1,000 ± 200 | 600 ± 300 | 410 ± 40 |
| Average horizontal wind velocity $[\frac{m}{s}]$ | 0.4 ± 0.2 | 0.3 ± 0.2 | 0.4 ± 0.2 |

Table 3.11: *Experiment 3 (Thermal with wind)* result

The added wind without turbulence has a less negative impact on the trained Policy compared to the Reichmann rules. The Policy's *Success rate* decreased to 92%, but its *Episode reward* remained similar to Experiment 1. On the other hand, Reichmann rules' success rate dropped to 25%, and its reward decreased compared to Experiment 2 ($200 \pm 200 \rightarrow 100 \pm 200$).

The distribution of *Average distance from core* and *Altitude* can be seen in Figure 3.12.

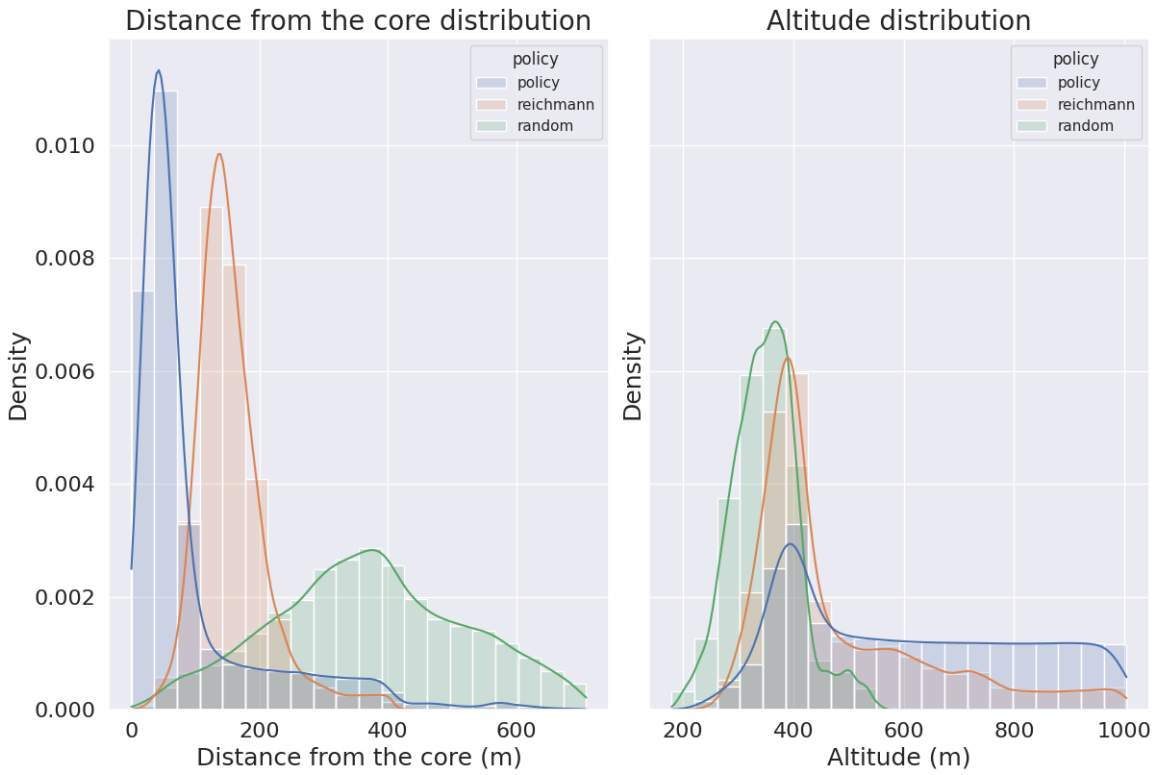



Figure 3.12: *Average distance from core* and *Altitude* distribution for *Experiment 3 (Thermal with wind)*

### Results of *Experiment 4 (Thermal with wind and turbulence)*

The results of the experiment can be seen in Table 3.12.

| | Policy | Reichmann | Random |
|---|---|---|---|
| Success rate | 90.0% | 26.0% | 0.0% |
| Episode reward | 300 ± 100 | 100 ± 200 | -60 ± 40 |
| Episode length $[s]$ | 290 ± 70 | 500 ± 300 | 200 ± 100 |
| Maximum time without lift $[s]$ | 50 ± 50 | 140 ± 70 | 150 ± 40 |
| Average distance from core $[m]$ | 100 ± 100 | 160 ± 50 | 400 ± 200 |
| Average vertical velocity $[\frac{m}{s}]$ | 2 ± 1 | 0.2 ± 0.8 | -0.6 ± 0.7 |
| Minimum altitude $[m]$ | 360 ± 40 | 330 ± 50 | 280 ± 50 |
| Average altitude $[m]$ | 600 ± 90 | 500 ± 100 | 350 ± 50 |
| Maximum altitude $[m]$ | 900 ± 200 | 600 ± 300 | 410 ± 60 |
| Average horizontal wind velocity $[\frac{m}{s}]$ | 0.5 ± 0.4 | 0.3 ± 0.2 | 0.4 ± 0.2 |

Table 3.12: *Experiment 4 (Thermal with wind and turbulence)* result

When both turbulence and wind were added, the trained Policy's success rate decreased from 93% to 90% compared to *Experiment 1*, and also the *Episode reward* decreased from 400±100 to 300±100. For the Reichmann rules the added turbulence besides the added wind didn't decrease the performance significantly.

The distribution of *Average distance from core* and *Altitude* can be seen in Figure 3.13.

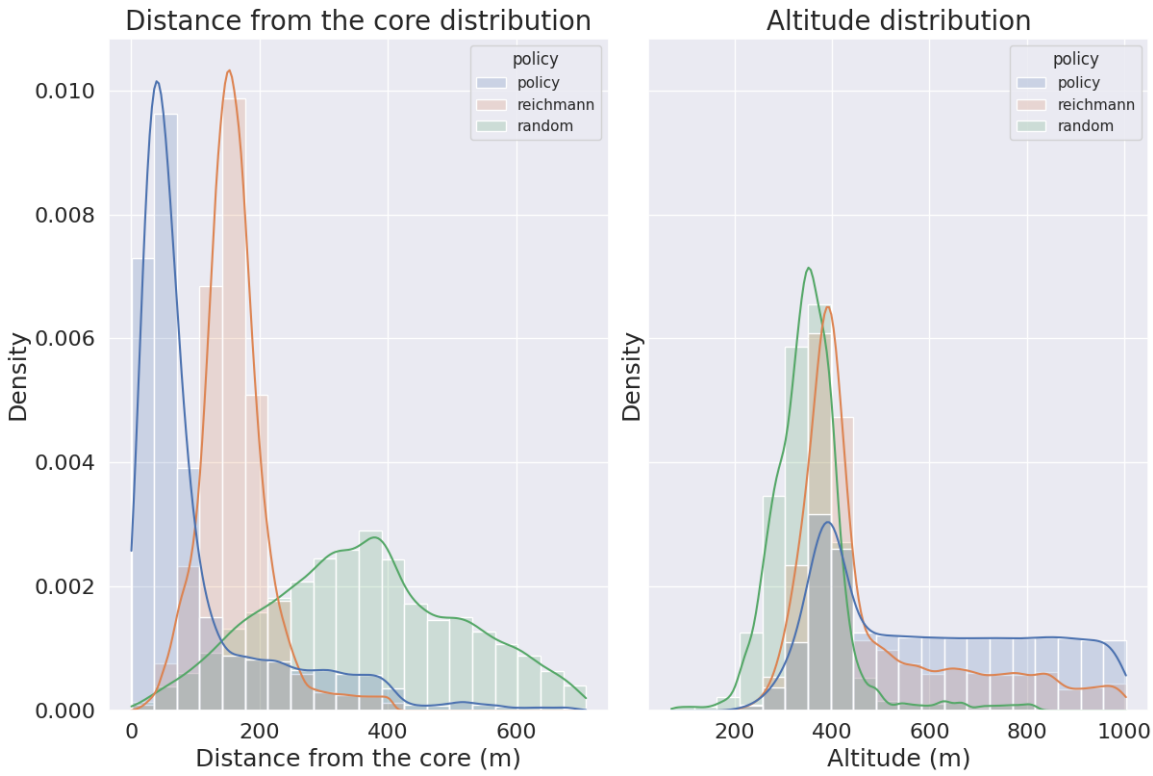



Figure 3.13: *Average distance from core* and *Altitude* distribution for *Experiment 4 (Thermal with wind and turbulence)*

### 3.6.8 Qualitative Results

Based on the videos of the simulations, it was observed empirically that despite the glider couldn't receive clear lift readings due to the turbulent air, the Policy learned to fly straight forward until it found a noticeable lift.

When it initially missed the thermal, it started to explore, and when it rediscovered it, the glider was generally able to catch the thermal and ascend to the top (see Figure 3.14).

Additionally, if it drifted away from the core of the thermal, it was able to correct its circling trajectory towards the core (see Figure 3.15).

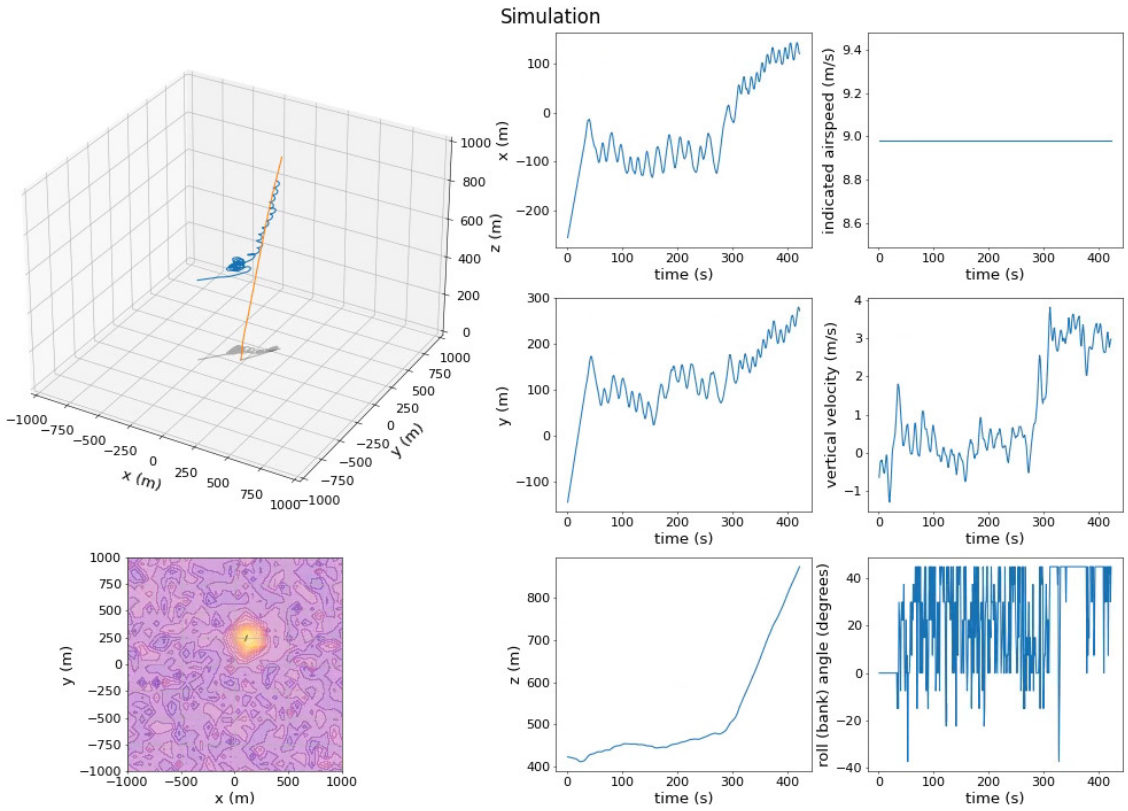



Figure 3.14: Exploration after initially missing the thermal

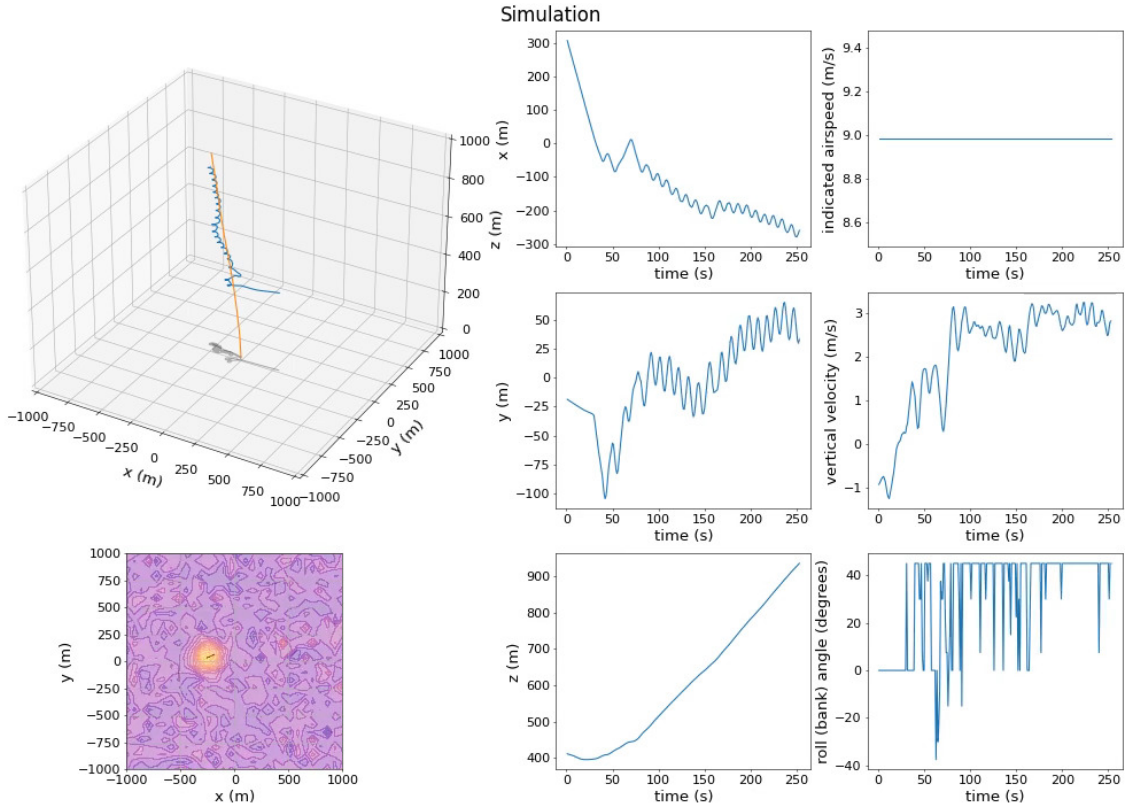



Figure 3.15: Correction towards the core after drifting away

# Chapter 4

# Conclusion/Future work

While the Transformer-based architecture has shown promising qualitative and quantitative results in turbulent and windy thermals in a simulated environment, further work is needed to test its real-world applicability by integrating it with autopilot software. A key area for future research involves exploring ways to reduce the current 1Hz sampling rate and increase the 30-second context window length to improve its control and long-term performance.

Moreover, further investigation is needed to compare the results with other thermaling agents. Currently, there is no known benchmark that compares glider thermaling and cross-country flying, making it unfeasible at the moment. It would be beneficial to enhance the developed simulator with more detailed aerodynamics and realistic turbulent models, thereby reducing the sim-to-real gap, and publish the environment so that it could be used as a soaring training environment and benchmark in the future.

A future research goal is to incorporate bioinspiration [61] and meta-learning [62] into the learning algorithm, teaching the agent the flight strategies of birds based on their soaring behaviors. Furthermore, utilizing multiple modalities for decision-making, such as cumulus clouds or observing other birds, could be explored. Additionally investigating how multiple UAVs can synchronize their flight with each other and with birds both in thermaling and during cross-country flight while maximizing the amount of information obtained from the environment would be valuable.

# Acknowledgements

I want to thank my Supervisors, and also Pedro Lacerda for their good questions, insightful ideas, and inspiration. My heartfelt appreciation also goes to my family for their constant support and enduring patience.

# Bibliography


[1] Catherine Brahic. "The Perlan Project: flying on the thinnest air". In: *New Scientist* 213.2846 (2012), pp. 34–37.

[2] URL: https://perlanproject.org/.

[3] Steve Longland. *Gliding*. en. Ramsbury, England: Crowood Press, Aug. 2012.

[4] John M Wallace and Peter V Hobbs. *Atmospheric science: an introductory survey*. Vol. 92. Elsevier, 2006.

[5] *Lapse Rates*. URL: https://revisionworld.com/a2-level-level-revision/geography/challenge-atmosphere/atmospheric-process/moisture-atmosphere.

[6] Bernard Eckey. *Advanced soaring made easy*. Future Aviation, 2009.

[7] AG Williams and JM Hacker. "The composite shape and structure of coherent eddies in the convective boundary layer". In: *Boundary-Layer Meteorology* 61 (1992), pp. 213–245.

[8] RS Scorer and FH Ludlam. "Bubble theory of penetrative convection". In: *Quarterly Journal of the Royal Meteorological Society* 79.339 (1953), pp. 94–103.

[9] Eric J. Kim and Ruben E. Perez. "Neuroevolutionary Control for Autonomous Soaring". In: *Aerospace* 8.9 (2021). ISSN: 2226-4310. DOI: 10.3390/aerospace8090267. URL: https://www.mdpi.com/2226-4310/8/9/267.

[10] Abdulghani Mohamed et al. "Opportunistic soaring by birds suggests new opportunities for atmospheric energy harvesting by flying robots". In: *Journal of the Royal Society Interface* 19.196 (2022), p. 20220671.

[11] *Frames of Reference*. URL: https://jsbsim-team.github.io/jsbsim-reference-manual/mypages/user-manual-frames-of-reference/.

[12] Gautam Reddy et al. "Glider soaring via reinforcement learning in the field". In: *Nature* 562 (Oct. 2018). DOI: 10.1038/s41586-018-0533-0.

[13] András Zábó. "Determination of flight parameters of soaring birds based on their trajectories using realistic thermal simulations". MSc thesis. Eötvös Lóránd University, Faculty of Science, 2022.

[14] Yüksel Eraslan. "A Training Sailplane Design". PhD thesis. Nov. 2018.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[16] Gergely Neu and Csaba Szepesvari. *Apprenticeship Learning using Inverse Reinforcement Learning and Gradient Methods*. 2012. arXiv: 1206.5264 [cs.LG].

[17] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in neural information processing systems* 29 (2016).

[18] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[19] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[20] Sergey Levine. *Deep Reinforcement Learning*. 2022. URL: http://rail.eecs.berkeley.edu/deeprlcourse/ (visited on 04/09/2023).

[21] Sanyam Kapoor. *Policy gradients in a Nutshell*. 2018. URL: https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d (visited on 04/09/2023).

[22] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8 (1992), pp. 229–256.

[23] Faisal Ahmed. *Baseline for policy gradients that all deep learning enthusists must know*. 2020. URL: https://www.analyticsvidhya.com/blog/2020/11/baseline-for-policy-gradients/ (visited on 04/09/2023).

[24] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[25] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[26] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79 –86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

[27] Yan Duan et al. "Benchmarking deep reinforcement learning for continuous control". In: *International conference on machine learning*. PMLR. 2016, pp. 1329–1338.

[28] Claude Elwood Shannon. "A mathematical theory of communication". In: *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.

[29] *Turn Performance*. URL: https://code7700.com/turn_performance.htm.

[30] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[31] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL].

[32] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[33] Yatian Shen and Xuan-Jing Huang. "Attention-based convolutional neural network for semantic relation extraction". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. 2016, pp. 2526–2536.

[34] *The Illustrated GPT-2 (Visualizing Transformer Language Models)*. URL: https://jalammar.github.io/illustrated-gpt2/.

[35] *Tutorial 5: Transformers and Multi-Head Attention*. URL: https://lightning.ai/docs/pytorch/latest/notebooks/course_UvA-DL/05-transformers-and-MH-attention.html.

[36] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[37] Wenzhe Li et al. *A Survey on Transformers in Reinforcement Learning*. 2023. DOI: 10.48550/ARXIV.2301.03044. URL: https://arxiv.org/abs/2301.03044.

[38] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[39] Emilio Parisotto et al. "Stabilizing transformers for reinforcement learning". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 7487–7498.

[40] Luckeciano C Melo. "Transformers are meta-reinforcement learners". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 15340–15359.

[41] Xiao Shi Huang et al. "Improving transformer optimization through better initialization". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4475–4483.

[42] Michael Allen and Victor Lin. "Guidance and Control of an Autonomous Soaring Vehicle with Flight Test Results". In: *45th AIAA Aerospace Sciences Meeting and Exhibit*. DOI: 10.2514/6.2007-867. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2007-867. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2007-867.

[43] Samuel Tabor, Iain Guilliard, and Andrey Kolobov. "ArduSoar: An Open-Source Thermalling Controller for Resource-Constrained Autopilots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 6255–6262. DOI: 10.1109/IROS.2018.8593510.

[44] Iain Guilliard et al. *Autonomous Thermalling as a Partially Observable Markov Decision Process (Extended Version)*. 2018. DOI: 10.48550/ARXIV.1805.09875. URL: https://arxiv.org/abs/1805.09875.

[45] Gautam Reddy et al. "Learning to soar in turbulent environments". In: *Proceedings of the National Academy of Sciences* 113 (Aug. 2016), p. 201606075. DOI: 10.1073/pnas.1606075113.

[46] Stefan Notter, Fabian Schimpf, and Walter Fichter. "Hierarchical Reinforcement Learning Approach Towards Autonomous Cross-Country Soaring". In: Jan. 2021. DOI: 10.2514/6.2021-2010.

[47] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[48] Zsuzsa Ákos et al. "Thermal soaring flight of birds and unmanned aerial vehicles". In: *Bioinspiration & Biomimetics* 5.4 (Nov. 2010), p. 045003. DOI: 10.1088/1748-3182/5/4/045003. URL: https://dx.doi.org/10.1088/1748-3182/5/4/045003.

[49] James F Manwell, Jon G McGowan, and Anthony L Rogers. *Wind energy explained: theory, design and application*. John Wiley & Sons, 2010. Chap. 2, pp. 46–47.

[50] *Gymnasium documentation*. URL: https://gymnasium.farama.org/environments/classic_control/cart_pole/.

[51] *CART pole control environment in Openai Gym (gymnasium)- introduction to openai gym*. 2023. URL: https://aleksandarhaber.com/cart-pole-control-environment-in-openai-gym-gymnasium-introduction-to-openai-gym/.

[52] Deheng Ye et al. "Mastering complex control in moba games with deep reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 6672–6679.

[53] Farama Foundation. *Gymnasium*. 2022. URL: https://gymnasium.farama.org/.

[54] Greg Brockman et al. *OpenAI Gym*. 2016. DOI: 10.48550/ARXIV.1606.01540. URL: https://arxiv.org/abs/1606.01540.

[55] Jiayi Weng et al. "Tianshou: A Highly Modularized Deep Reinforcement Learning Library". In: *Journal of Machine Learning Research* 23.267 (2022), pp. 1–6. URL: http://jmlr.org/papers/v23/21-1127.html.

[56] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[57] neptune.ai. *neptune.ai: experiment tracking and model registry*. 2022. URL: `https://neptune.ai`.

[58] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[59] Zoltán Szarvas. *GlideRL ThermalingAgent*. `https://gitlab.com/avisense/GlideRL-ThermalingAgent`. 2023.

[60] Helmut Reichmann. *Cross Country Soaring*. Soaring Society of Amer, 1993.

[61] Zsuzsa Ákos, Máté Nagy, and Tamás Vicsek. "Comparing bird and human soaring strategies". In: *Proceedings of the National Academy of Sciences* 105.11 (Mar. 2008), pp. 4139–4143. DOI: `10.1073/pnas.0707711105`. URL: `https://doi.org/10.1073/pnas.0707711105`.

[62] Jacob Beck et al. "A Survey of Meta-Reinforcement Learning". In: *arXiv preprint arXiv:2301.08028* (2023).