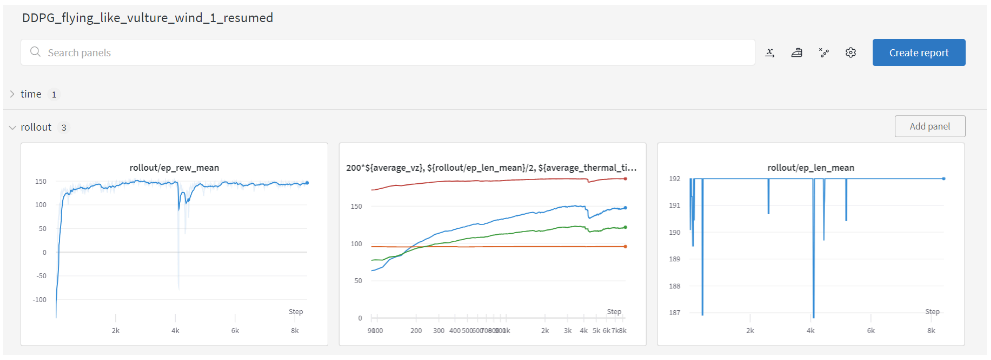
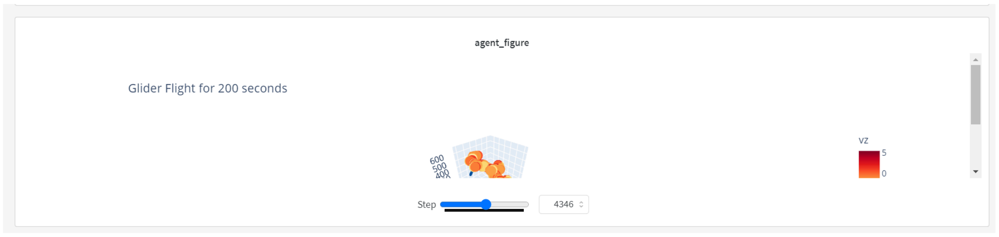
RL gliding project - documentation and instructions

Author: Yoav Flato, yoav.flato@mail.huji.ac.il (and ChatGPT)
Git repo: https://github.com/MicroFlightLab/

Important Note:

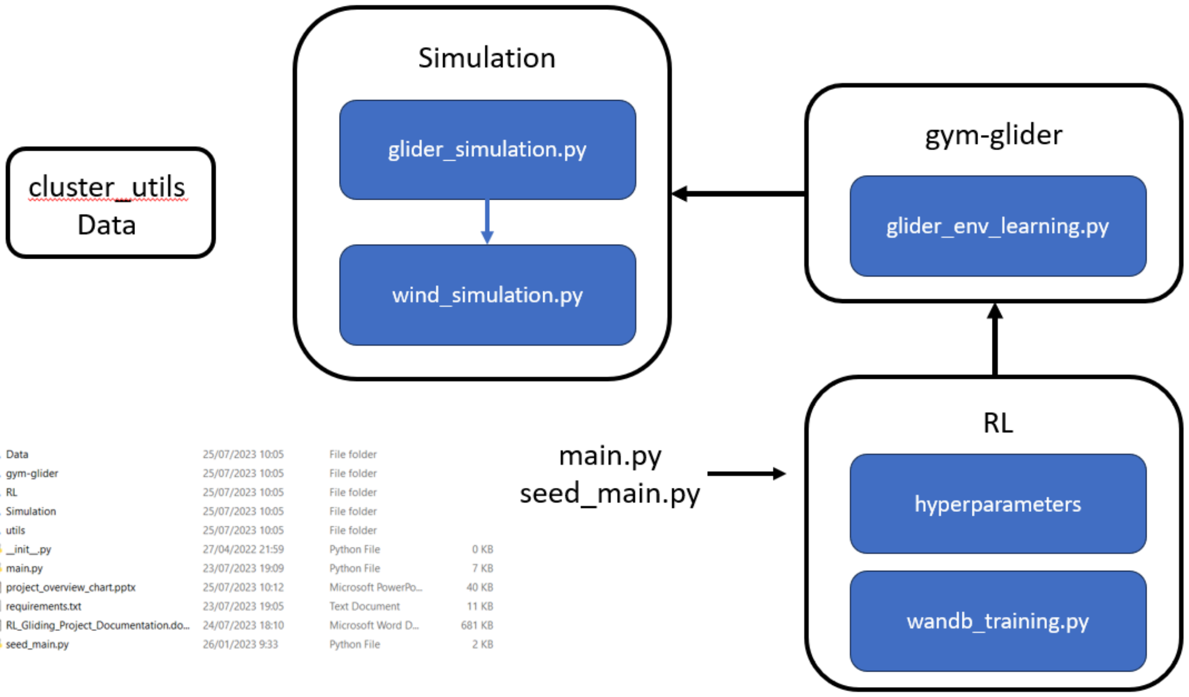
I strongly encourage you to thoroughly review the code and experiment with it. It might take a week or two to get in, but it's worth it. Don't hesitate to seek my assistance if needed. The purpose of this documentation is to provide you with a one-click solution to access the dashboard outlined below.

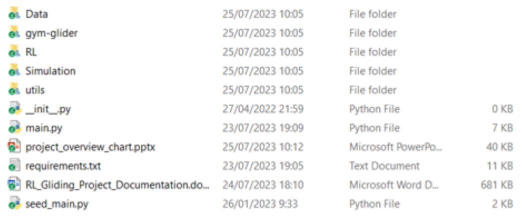

DDPG_flying_like_vulture_wind_1_resumed
Search panels
Create report
time 1
rollout 3
Add panel
rollout/ep_rew_mean
200*${average_vz}, ${rollout/ep_len_mean}/2, ${average_thermal_ti...
rollout/ep_len_mean
Step



Project overview

The project is composed from different directories as described below:





Data	25/07/2023 10:05	File folder	
gym-glider	25/07/2023 10:05	File folder	
RL	25/07/2023 10:05	File folder	
Simulation	25/07/2023 10:05	File folder	
utils	25/07/2023 10:05	File folder	
__init__.py	27/04/2022 21:59	Python File	0 KB
main.py	23/07/2023 19:09	Python File	7 KB
project_overview_chart.pptx	25/07/2023 10:12	Microsoft PowerPo...	40 KB
requirements.txt	23/07/2023 19:05	Text Document	11 KB
RL_Gliding_Project_Documentation.do...	24/07/2023 18:10	Microsoft Word D...	681 KB
seed_main.py	26/01/2023 9:33	Python File	2 KB

We will go over each part and explain it.

Simulation

Within this directory, you will find two crucial modules responsible for constructing the simulation framework. The first module, "wind_simulation," is dedicated to modeling the wind field. The second one, "glider_simulation," incorporates the glider model.

Advise: I advise running the notebook named "run_simulation.ipynb" to gain a better understanding of the framework.

Please note that it is essential to familiarize yourself with this simulation framework, especially if you plan to introduce additional options for actions and states, as you will need to make appropriate modifications.

gym-glider

This section focuses on the gym environment, which serves as a library to model the glider as a reinforcement learning problem. Notably, both the state and action spaces are continuous within our gym environment.

This approach offers significant advantages as we can now leverage a wide range of pre-built RL algorithms from libraries such as stable baselines 3.

The gym-glider acts as a wrapper for the simulation environment, and understanding the interface between gym-glider and the simulation is crucial. The gym-glider receives information from the environment, constructs a state, and determines the reward through the "get_curr_state_for_learning" function, while it interacts through a separate set of functions.

To initialize an environment, it's essential to configure the hyperparameters file (a sample can be found in the RL directory, and further details are available in the "How to train an agent" section).

Advise: I recommend running the notebook named "run_gym_rl.ipynb" to deepen your understanding of this concept.

RL

The RL directory comprises the training framework responsible for logging and saving all results to the wandb platform (an account is required). While you have the option to construct your own learning framework relying solely on our gym-glider simulation, I strongly recommend using this framework for training. It offers valuable insights into the learning process, aids in debugging, provides agent insights, and records the run history.

Within the RL directory, there are three essential files that you should familiarize yourself with:

1. The hyperparameters file represents a learning process. Modifying the values in this file will affect the glider environment that is created. Here, you can influence both the environment and glider characteristics, as well as the state, action, and reward in the RL setup. Be sure to read this file and refer to the explanation in the "How to train an agent" section.
2. The "wandb_training.py" manages the complete learning process and communicates with wandb. When you run the main.py file, this script is executed.
3. The "wandb_utils.py" includes numerous examples of how to use the wandb API to retrieve information logged during the process. This API proves useful for analyzing results, and I recommend exploring and understanding it. One noteworthy function is "get_df_and_model_dict_by_run_id", which serves as an excellent example of API utilization.

By familiarizing yourself with these files, you'll gain a better grasp of the learning process and effectively utilize the training framework.

Here is an example of how to use the "get_df_and_model_dict_by_run_id" function.

```
params = {"project_name": "ChangingWindConditions",
          "run_id": "3b3r1cat",
          "user_name": "yoavflato",
          "model_full_path_in_wandb": "model_DDPG_5900000.zip",
          "update_hyperparameters": {"horizontal_wind": [3, 0, 0],
"horizontal_wind_settings": None},
          "episodes": 1}
res = wandb_utils.get_df_and_model_dict_by_run_id(**params)
```

This code will to load the model named "model_DDPG_5900000.zip," which was previously saved during a learning process after 5900000 timesteps. The output of this function includes the loaded model, an environment with varying wind settings (using the hyperparameter dictionary that was saved in wandb and utilized during the training process), and updated hyperparameters for the new environment.

Moreover, the function provides both the model and the dataframe containing the trajectory resulting from running the model on the environment.

Note: Inside the RL directory there is a subdirectory called “analyze_results”which includes neuron activation patterns investigation code.

Main files

Both files are used to run wandb_training.py in some mode. Further explanation in the “how to train an agent” part.

Cluster utils and data

Cluster utils is an example for code that will help to connect and handle the cluster.

The data contains analysis code on the data.

Further explanations can be found in the sections below.

Installations

To ensure everything runs smoothly, you must download all the required libraries using the following command:

```
pip install -r requirements.txt
```

I personally used Python version 3.8 on my computer, but rest assured, it should also work on Python 3.9.

Additionally, you'll need to install our custom-made gym-environment by executing the following command in the bash:

```
pip install -e gym-glider
```

Please be aware that there might be installation issues, and if you encounter any problems, feel free to contact us for assistance.

How to train an agent

After finishing the installation, we can train an agent.

Our framework is working with weights and biases (wandb), if want to train and get detailed dashboard you might want to open wandb account (it's free).

You can also use the gym-glider library directly (note: to create such a gym environment you need to pass also the hyperparameter file).

To run single training process, we will need to run the "main.py" file with 2 arguments specifying the algorithm name (currently there are 4 algorithms that our framework support: DDPG, A2C, SAC, PPO – to add more need to add another algorithm to the dictionary in the file "training_utils.py").

Example for how run the training process:

```
Python main.py DDPG check
```

Note there are 3 types of run, regular, seed, sweep. We will first go over regular learning process.

Regular training process

This training process is achieved by running the line: `Python main.py DDPG check --hyperparmeter 0`

(review the code to see all other options that are possible in run, by adding --)

Note: there are couple of hyperparameters file to enable different runs on the same time.

This command will run the function the "wandb_train_env" function of in wandb_training.py file with configuration detailed in the hyperparameter file.

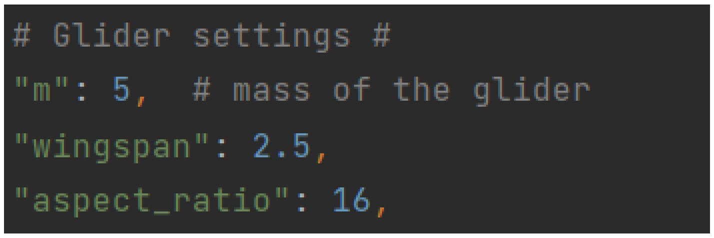
Before running this command you need to make sure that the hyperpaprameters file Is set to the parameters you want. We will now go over this file to make sure it's clear for you. From this file you will control the learning process.

Each change in the file will change some parameter in the run.

The hyperparameter file contains:

Glider settings

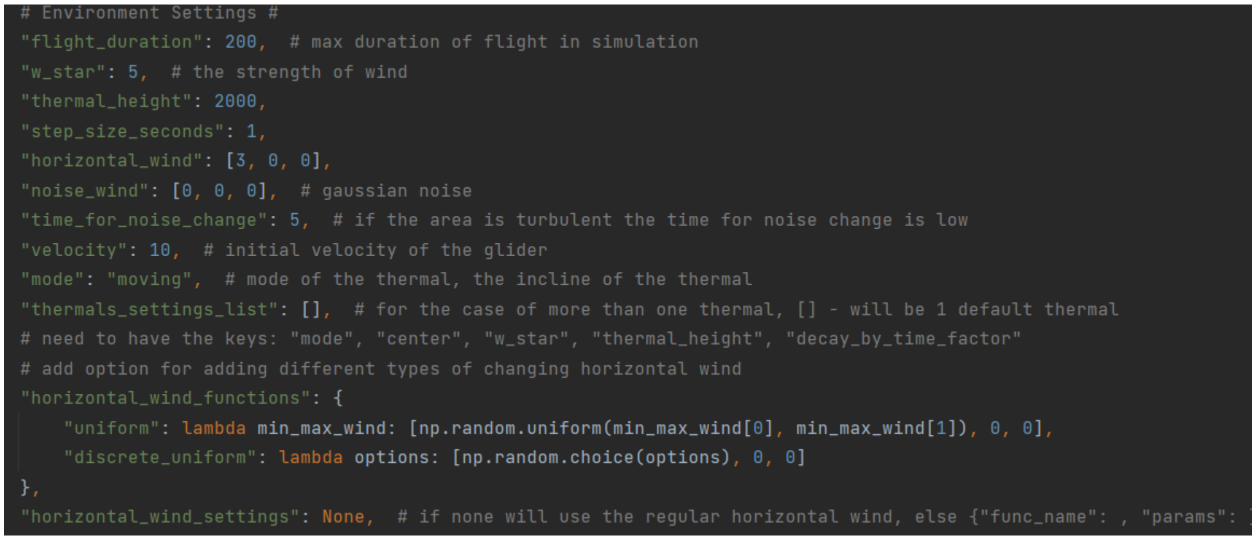
glider characteristics for the simulation:



```
# Glider settings #
"m": 5,  # mass of the glider
"wingspan": 2.5,
"aspect_ratio": 16,
```

Environment settings

parameters that determine the winds, the thermal, some initial parameters of the glider in the environment (there can be a run with changing winds by using horizontal wind settings).

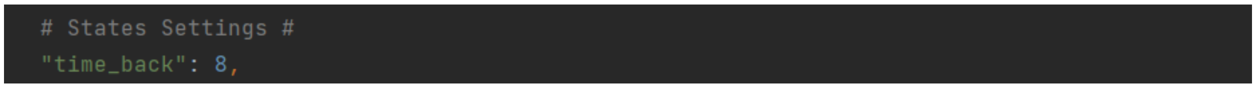


```
# Environment Settings #
"flight_duration": 200,  # max duration of flight in simulation
"w_star": 5,  # the strength of wind
"thermal_height": 2000,
"step_size_seconds": 1,
"horizontal_wind": [3, 0, 0],
"noise_wind": [0, 0, 0],  # gaussian noise
"time_for_noise_change": 5,  # if the area is turbulent the time for noise change is low
"velocity": 10,  # initial velocity of the glider
"mode": "moving",  # mode of the thermal, the incline of the thermal
"thermals_settings_list": [],  # for the case of more than one thermal, [] - will be 1 default thermal
# need to have the keys: "mode", "center", "w_star", "thermal_height", "decay_by_time_factor"
# add option for adding different types of changing horizontal wind
"horizontal_wind_functions": {
    "uniform": lambda min_max_wind: [np.random.uniform(min_max_wind[0], min_max_wind[1]), 0, 0],
    "discrete_uniform": lambda options: [np.random.choice(options), 0, 0]
},
"horizontal_wind_settings": None,  # if none will use the regular horizontal wind, else {"func_name": , "params":
```

States settings

This section is of utmost importance! It encompasses the declaration of all RL states, giving you the freedom to choose and customize each state according to your preferences.

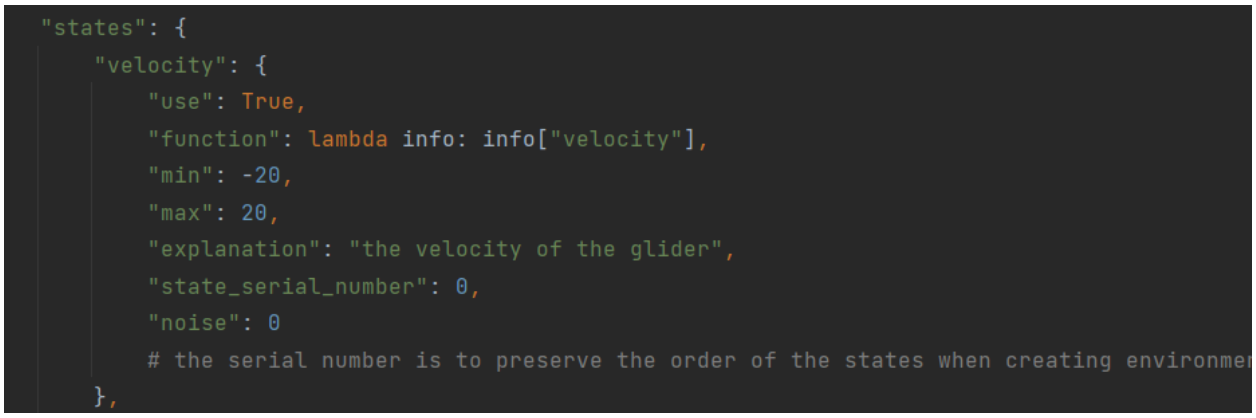
The history size is controlled by the "time_back" parameter. For instance, setting a history size of 3 implies that each state will incorporate the last 3 timesteps of selected parameters, such as velocity or bank_angle.



```
# States Settings #
"time_back": 8,
```

The state comprises various parameters, such as velocity and bank_angle, for instance.

Each state is represented as an element within the states dictionary.



```
"states": {
    "velocity": {
        "use": True,
        "function": lambda info: info["velocity"],
        "min": -20,
        "max": 20,
        "explanation": "the velocity of the glider",
        "state_serial_number": 0,
        "noise": 0
        # the serial number is to preserve the order of the states when creating environme
    },
```

The key value determines the parameter name, while the "use" value determines whether the parameter should be included in the state. This is a crucial distinction to make.

The "function" defines how to calculate the parameter. During each simulation step, a set of parameters is returned (refer to the "get_curr_state_for_learning" function in glider_simulation.py). By applying the lambda function to these parameters, the corresponding state value is obtained.

The "min" and "max" attributes are employed for parameter value normalization during the run.

The "explanation" field provides insight into the purpose of each state.

The "state_serial_number" is essential for the proper functioning of the program and should not be altered.

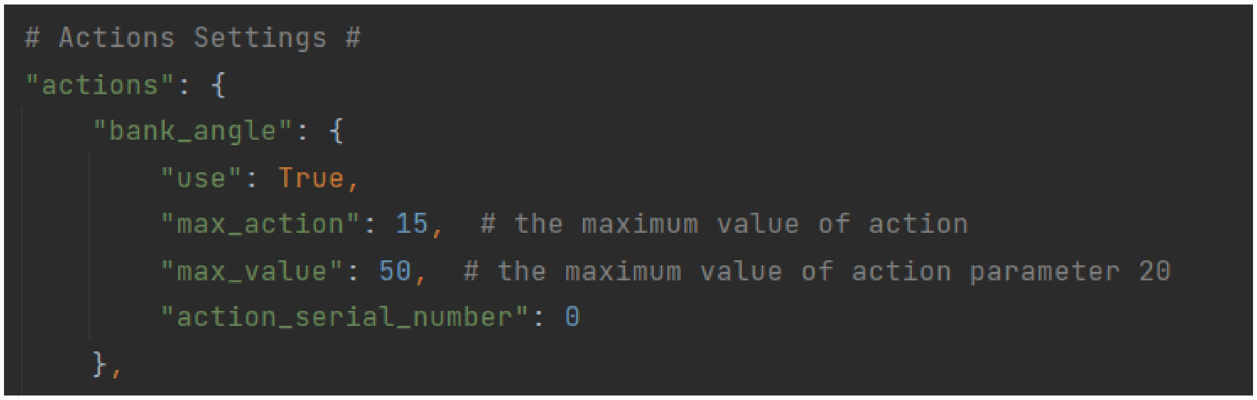
The "noise" is used to introduce Gaussian noise to the state during the run, which can aid in training more robust agents.

<u>Note:</u> If you wish to add another state, simply add a new element to the dictionary, ensuring it has a unique "state_serial_number."

Actions settings

The actions exhibit less flexibility and are relatively more hard-coded. However, by adjusting the "use" parameter, you can decide whether to utilize a particular action or not.

There are four actions in total: three angle actions and one wingspan action. The wingspan action pertains to altering the wingspan and aspect ratio within the simulation.

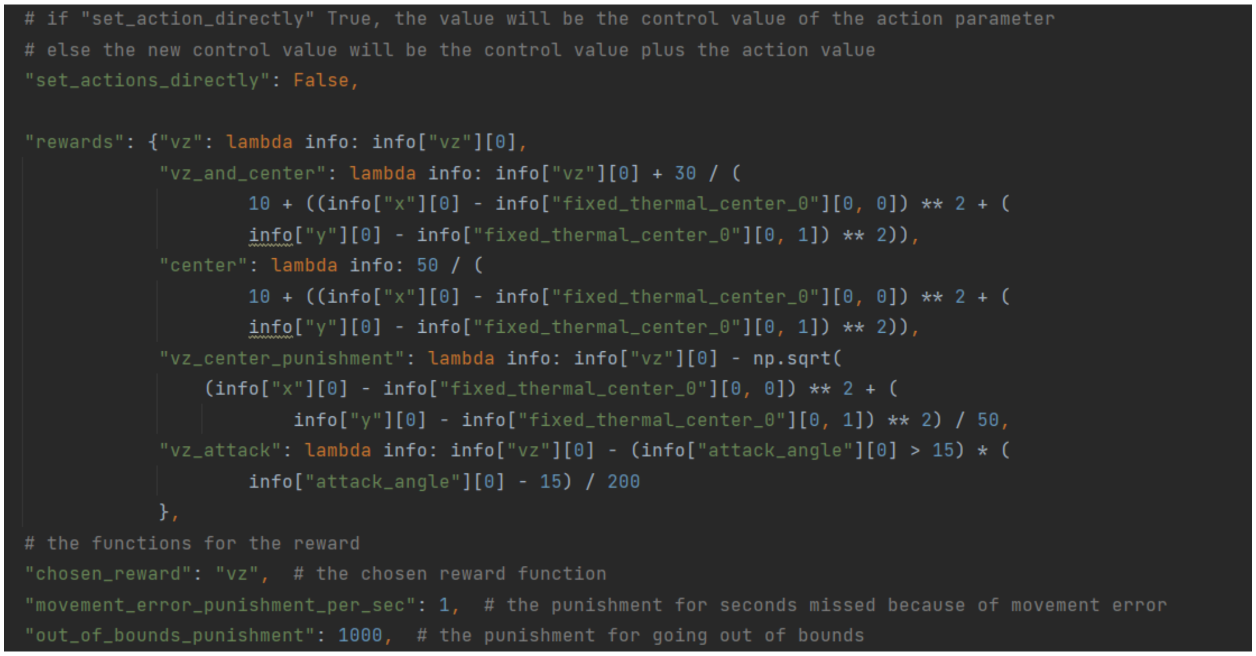


```
# Actions Settings #
"actions": {
    "bank_angle": {
        "use": True,
        "max_action": 15,  # the maximum value of action
        "max_value": 50,  # the maximum value of action parameter 20
        "action_serial_number": 0
    },
```

Reward settings

To customize the reward, modify the "chosen_reward" in the hyperparameter file. Additionally, you can introduce new rewards by adding them to the "rewards" dictionary.

Furthermore, be aware that there are penalties in place to address instabilities.



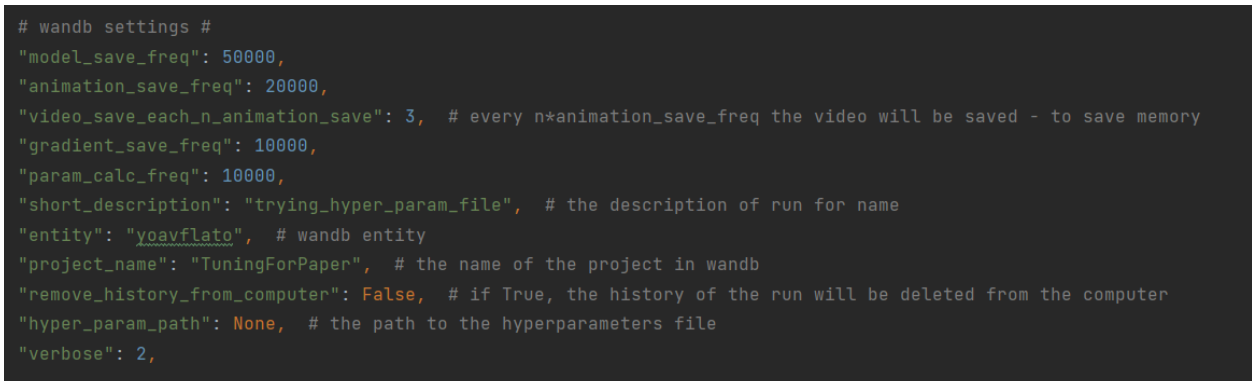
```
# if "set_action_directly" True, the value will be the control value of the action parameter
# else the new control value will be the control value plus the action value
"set_actions_directly": False,

"rewards": {"vz": lambda info: info["vz"][0],
            "vz_and_center": lambda info: info["vz"][0] + 30 / (
                    10 + ((info["x"][0] - info["fixed_thermal_center_0"][0, 0]) ** 2 + (
                    info["y"][0] - info["fixed_thermal_center_0"][0, 1]) ** 2)),
            "center": lambda info: 50 / (
                    10 + ((info["x"][0] - info["fixed_thermal_center_0"][0, 0]) ** 2 + (
                    info["y"][0] - info["fixed_thermal_center_0"][0, 1]) ** 2)),
            "vz_center_punishment": lambda info: info["vz"][0] - np.sqrt(
                (info["x"][0] - info["fixed_thermal_center_0"][0, 0]) ** 2 + (
                        info["y"][0] - info["fixed_thermal_center_0"][0, 1]) ** 2) / 50,
            "vz_attack": lambda info: info["vz"][0] - (info["attack_angle"][0] > 15) * (
                    info["attack_angle"][0] - 15) / 200
            },
# the functions for the reward
"chosen_reward": "vz",  # the chosen reward function
"movement_error_punishment_per_sec": 1,  # the punishment for seconds missed because of movement error
"out_of_bounds_punishment": 1000,  # the punishment for going out of bounds
```

Wandb settings

Wandb is a logging platform used to track values and charts throughout a process or training phase.

The essential steps involve modifying the "entity" (representing the user in wandb) and "project_name" to ensure the run is uploaded to your desired project.

Additionally, you have the flexibility to adjust settings for the frequency of file and chart saving.

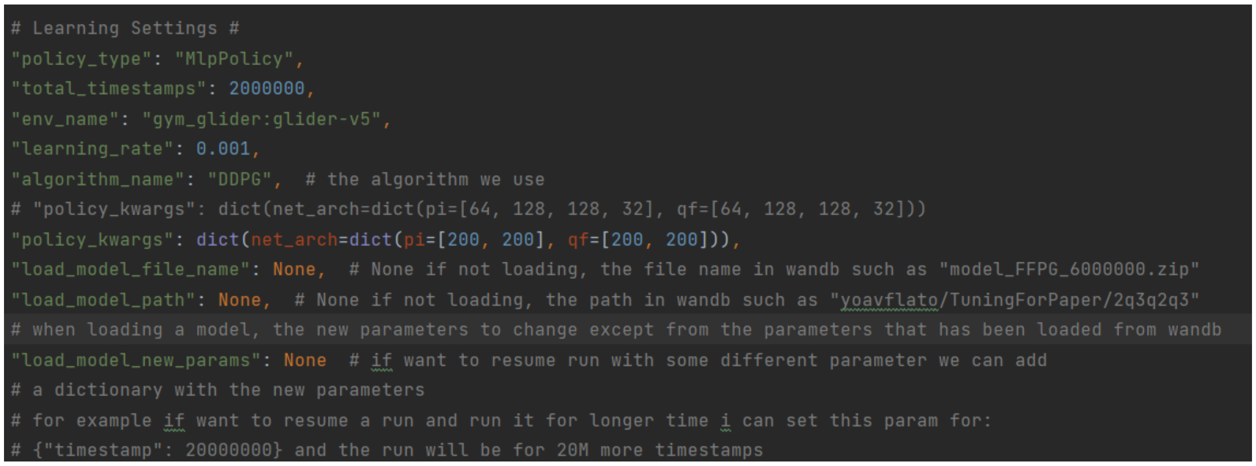


```
# wandb settings #
"model_save_freq": 50000,
"animation_save_freq": 20000,
"video_save_each_n_animation_save": 3,  # every n*animation_save_freq the video will be saved - to save memory
"gradient_save_freq": 10000,
"param_calc_freq": 10000,
"short_description": "trying_hyper_param_file",  # the description of run for name
"entity": "yoavflato",  # wandb entity
"project_name": "TuningForPaper",  # the name of the project in wandb
"remove_history_from_computer": False,  # if True, the history of the run will be deleted from the computer
"hyper_param_path": None,  # the path to the hyperparameters file
"verbose": 2,
```

Learning process settings

You can manage the learning process's running time using the "total_timesteps" parameter. Additionally, you can specify the type and shape of the neural network by adjusting "policy_kwargs" and "policy_type."

Furthermore, the option to resume a run is available. To resume a specific run with customized parameters, modify "load_model_file_name," "load_model_path," and "load_model_new_params" accordingly.



```
# Learning Settings #
"policy_type": "MlpPolicy",
"total_timestamps": 2000000,
"env_name": "gym_glider:glider-v5",
"learning_rate": 0.001,
"algorithm_name": "DDPG",  # the algorithm we use
# "policy_kwargs": dict(net_arch=dict(pi=[64, 128, 128, 32], qf=[64, 128, 128, 32]))
"policy_kwargs": dict(net_arch=dict(pi=[200, 200], qf=[200, 200])),
"load_model_file_name": None,  # None if not loading, the file name in wandb such as "model_FFPG_6000000.zip"
"load_model_path": None,  # None if not loading, the path in wandb such as "yoavflato/TuningForPaper/2q3q2q3"
# when loading a model, the new parameters to change except from the parameters that has been loaded from wandb
"load_model_new_params": None  # if want to resume run with some different parameter we can add
# a dictionary with the new parameters
# for example if want to resume a run and run it for longer time i can set this param for:
# {"timestamp": 20000000} and the run will be for 20M more timestamps
```

You need to change the relevant keys to make a run with the desired states, actions, reward, timestamp and all other parameters.

Seed learning process

In this mode, we initiate the training process as follows:

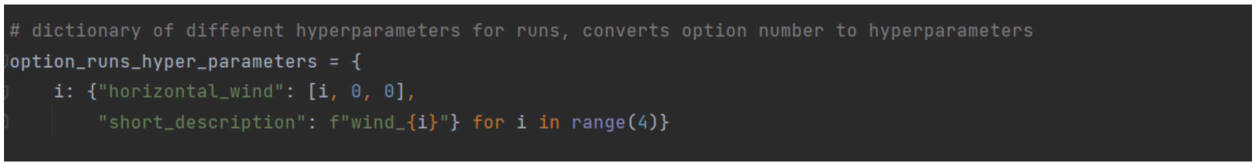
python seed_main.py seed_number

Here, the "seed_number" corresponds to a specific value that determines which hyperparameter to utilize (not to be confused with "--hyperparameter" usage).

The hyperparameter file contains a dictionary named "option_runs_hyper_parameters." The keys in this dictionary represent the seed numbers, and their respective values correspond to specific settings within the main "hyper_parameters_dict" that governs the entire run.

To illustrate this, consider the following important example:

Suppose we wish to conduct four runs with identical settings and training processes but varying horizontal winds. We would modify the "hyper_parameters_dict" and then adjust the "option_runs_hyper_parameters" to assign distinct seed_numbers (0/1/2/3) to each specific horizontal wind configuration.



```
# dictionary of different hyperparameters for runs, converts option number to hyperparameters
option_runs_hyper_parameters = {
    i: {"horizontal_wind": [i, 0, 0],
        "short_description": f"wind_{i}"} for i in range(4)}
```

If I will run Python seed_main.py 3, a training process with horizontal_wind=3 will be executed.

Now we can do a loop of batch commands that will run 4 distinct training process for each horizontal wind.

Sweep learning process

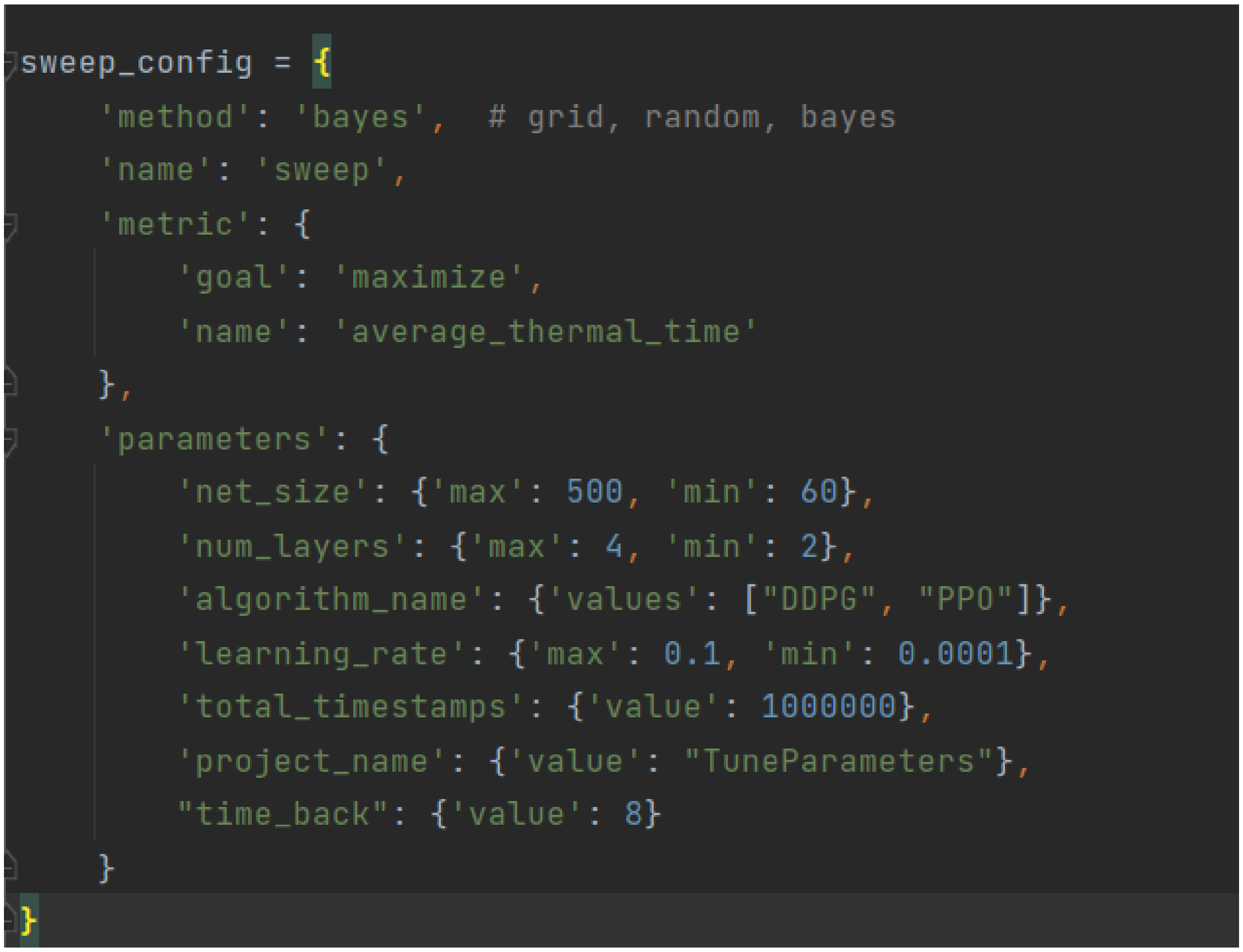
There is another option to do sweep over parameters for hyperparameter tuning.

To run this, you will need to do the following command:

```
Python seed_main.py sweep sweep_name
```

And change the “sweep_config” dictionary.

The goal is to maximize some variable that we log to wandb and the parameters are the parameters from "hyper_parameters_dict" dictionary.

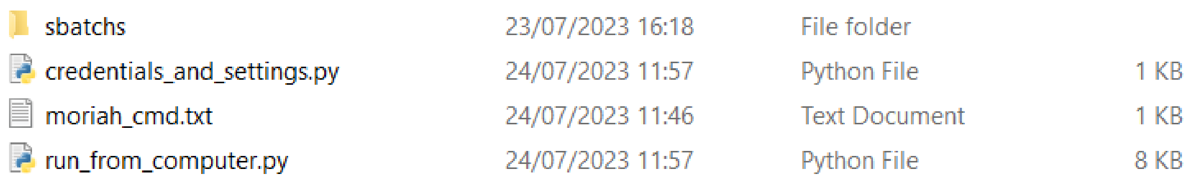


```
sweep_config = {
    'method': 'bayes',  # grid, random, bayes
    'name': 'sweep',
    'metric': {
        'goal': 'maximize',
        'name': 'average_thermal_time'
    },
    'parameters': {
        'net_size': {'max': 500, 'min': 60},
        'num_layers': {'max': 4, 'min': 2},
        'algorithm_name': {'values': ["DDPG", "PPO"]},
        'learning_rate': {'max': 0.1, 'min': 0.0001},
        'total_timestamps': {'value': 1000000},
        'project_name': {'value': "TuneParameters"},
        "time_back": {'value': 8}
    }
}
```

Working on the cluster - Relevant for HUJI students

Note: This part is only relevant for HUJI students.

For enhanced efficiency, it is recommended to execute this code on the cluster. To enable its functionality on your cluster account, necessary adaptations should be made within the cluster_utils directory.



sbatchs	23/07/2023 16:18	File folder	
credentials_and_settings.py	24/07/2023 11:57	Python File	1 KB
moriah_cmd.txt	24/07/2023 11:46	Text Document	1 KB
run_from_computer.py	24/07/2023 11:57	Python File	8 KB

To ensure smooth operation, modifications must be made to the sbatch files in the sbatchs directory and credentials.py. We highly advise reviewing the contents of run_from_computer.py, as it includes numerous automated commands tailored for cluster execution.

Installations and preparations

To begin, you should copy the project to the cluster and initiate a virtual environment that encompasses the required libraries. If using PyCharm, you can utilize PyCharm SSH deployment for this purpose.

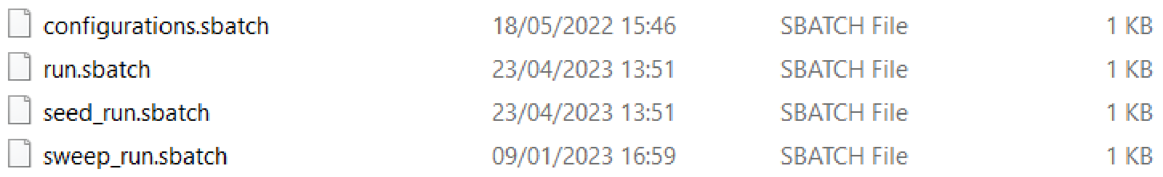
Additionally, ensure that the gym-glider library is installed by running the following command:

```
pip install -e gym-glider
```

Please note that the moriah_cmd.txt file contains useful commands for installation on the cluster.

Running a training process

To initiate a run on the cluster, you'll execute the corresponding sbatch file. There are three available modes: Sweep (sweep_run), seed (seed_run), and regular (run), each having its dedicated sbatch file.



configurations.sbatch	18/05/2022 15:46	SBATCH File	1 KB
run.sbatch	23/04/2023 13:51	SBATCH File	1 KB
seed_run.sbatch	23/04/2023 13:51	SBATCH File	1 KB
sweep_run.sbatch	09/01/2023 16:59	SBATCH File	1 KB

The Regular mode is used for a single run, Seed allows for simultaneous execution of multiple runs, and Sweep is employed for exploring various hyperparameters.

For comprehensive information on each mode, please refer to the "how to train an agent" section.

To start the process, execute the appropriate sbatch file and then patiently wait until you receive the completion notification via email.

You can run the commands on the cluster through mobaXterm.

Run on the cluster from your computer

To facilitate a convenient and seamless connection to the cluster, I have developed a user-friendly command-line program called "run_from_computer.py." This program establishes an SSH connection to the cluster, allowing you to execute commands with simple, concise inputs. To grasp its functionality, it's essential to review the code.

To run the program, execute the "run_from_computer.py" file and customize the credentials and settings according to your cluster connection preferences and file paths.

Note: Even if you opt not to utilize this program, I strongly encourage you to explore the code and familiarize yourself with the potential commands it offers.

Working with the data

I won't delve into extensive documentation for this part. However, within the Data directory, you'll find a file named "data_utls.py," which houses a class named "VulturesData." This class encompasses all the calculations performed on the data derived from the raw data (supplied by roi harel). You can gain a clear understanding of how the data manipulations are conducted by reviewing the code.

To generate the dataframes from roi harel datasets found here, simply execute the following command:

```
data = VulturesData(read_data=False, data_type="new")
```

The vultures raw data and pickles of my processed data can be found in this drive directory.