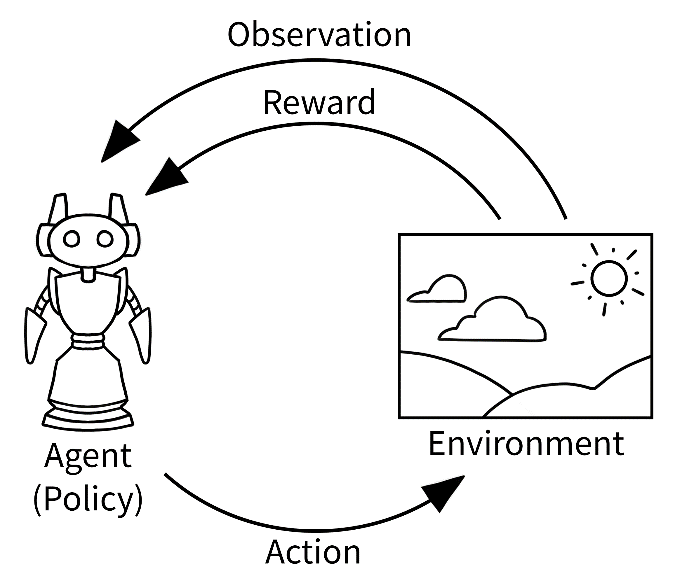
Autonomous Flight

Based on:

1. [zgoddard3](https://github.com/zgoddard3)/[jsbsim-gym](https://github.com/zgoddard3/jsbsim-gym)
   1. Gym-JSBSim provides reinforcement learning environments for the control of fixed-wing aircraft using the JSBSim flight dynamics model.
2. GYM 0.21.0
   1. The Gym interface is simple, pythonic, and capable of representing general RL problems



1. JSBSim Flight Dynamics Model v1.1.13
   1. JSBSim is a multi-platform, general purpose object-oriented Flight Dynamics Model (FDM) written in C++. The FDM is essentially the physics & math model that defines the movement of an aircraft, rocket, etc., under the forces and moments applied to it using the various control mechanisms and from the forces of nature.
2. JSBSim-ML v2.0
3. Python 3.9.13
4. PyCharm Community Edition 2022.3.3
5. Stable Baselines3
   1. Stable Baselines is a set of improved implementations of reinforcement learning algorithms based on OpenAI [Baselines](https://github.com/openai/baselines/).
6. PyTorch torch-2.0.0+cu117 ( for cuda support )
   1. PyTorch is a Python package that provides two high-level features:
      1. Tensor computation (like NumPy) with strong GPU acceleration
      2. Deep neural networks built on a tape-based autograd system
7. TensorBoard
   1. TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space.
8. PPO
   1. The Proximal Policy Optimization algorithm

State Space

"position/lat-gc-rad",  
"position/long-gc-rad",  
"position/h-sl-meters",  
"velocities/mach",  
"aero/alpha-rad",  
"aero/beta-rad",  
"velocities/p-rad\_sec",  
"velocities/q-rad\_sec",  
"velocities/r-rad\_sec",  
"attitude/phi-rad",  
"attitude/theta-rad",  
"attitude/psi-rad",

Action Space: (min=-1.0, max=1.0)

# Pass control inputs to JSBSim  
self.simulation.set\_property\_value("fcs/aileron-cmd-norm", roll\_cmd)  
self.simulation.set\_property\_value("fcs/elevator-cmd-norm", pitch\_cmd)  
self.simulation.set\_property\_value("fcs/rudder-cmd-norm", yaw\_cmd)  
self.simulation.set\_property\_value("fcs/throttle-cmd-norm", throttle)

# Pass control inputs to JSBSim  
self.simulation.set\_property\_value("fcs/aileron-cmd-norm", roll\_cmd)  
self.simulation.set\_property\_value("fcs/elevator-cmd-norm", pitch\_cmd)  
self.simulation.set\_property\_value("fcs/rudder-cmd-norm", yaw\_cmd)  
self.simulation.set\_property\_value("fcs/throttle-cmd-norm", throttle)

Run Command:

Cd C:\GitHub\gym-Jsbsim\jsbsim-gym-main

Python- C:\GitHub\gym-Jsbsim\jsbsim-gym-main\venv\Scripts\python.exe

Learn - C:\GitHub\gym-Jsbsim\jsbsim-gym-main\train\_PPO.py

Test - C:\GitHub\gym-Jsbsim\jsbsim-gym-main\test.py

TensorBoard - C:\GitHub\gym-Jsbsim\jsbsim-gym-main\venv\Scripts\tensorboard --logdir C:\\GitHub\\gym-Jsbsim\\jsbsim-gym-main\\logs

*Gym environment Description*

*### Description  
Gym environment using JSBSim to simulate an F-16 aerodynamics model with a  
simple point-to-point navigation task. The environment terminates when the  
agent enters a cylinder around the goal or crashes by flying lower than sea  
level. The goal is initialized at a random location in a cylinder around the  
agent's starting position.   
  
### Observation  
The observation is given as the position of the agent, velocity (mach, alpha,  
beta), angular rates, attitude, and position of the goal (concatenated in  
that order). Units are meters and radians.   
  
### Action Space  
Actions are given as normalized body rate commands and throttle command.   
These are passed into a low-level PID controller built into the JSBSim model  
itself. The rate commands should be normalized between [-1, 1] and the   
throttle command should be [0, 1].  
  
### Rewards  
A positive reward is given for reaching the goal and a negative reward is   
given for crashing. It is recommended to use the PositionReward wrapper   
below to eliminate the problem of sparse rewards.  
"""*

Policies:

Policies in Stable Baselines determine how an agent selects actions in an environment based on its observations.

**MlpPolicy**: This policy is a multi-layer perceptron (MLP) neural network that takes the observations as input and produces action probabilities or action values as output. It's a basic policy suitable for a wide range of environments.

Project main files:

The main files defining the environment and feature transformation are jsbsim\_gym/jsbsim\_gym.py and jsbsim\_gym/features.py. The files under jsbsim\_gym/visualization are auxiliary files for rendering the environment.

* jsbsim\_gym.py: This file defines the environment which wraps a JSBSim simulation which runs an F-16 aerodynamics model. The environment class defines a goal and reward function for the agent. Additional shaping rewards are also defined in a Gym wrapper in this file.
* features.py: This file defines a feature extractor for the JSBSim environment. This is the feature vector I found to be most beneficial for this task. Further details can be found in the comments in this file.
* train.py: This is a short script for training a SAC agent on the JSBSim environment. The hardcoded parameters should be sufficient to get decent results. The script takes about 12 hours to run on my desktop though time may vary depending on hardware.
* test.py: This script will run the trained agent for one episode while visualizing the environment. The visualization will automatically be saved to an MP4 video and GIF animation.

Good results in RL are generally dependent on finding appropriate hyperparameters. Recent algorithms (PPO, SAC, TD3) normally require little hyperparameter tuning, however, don’t expect the default ones to work on any environment.

A best practice when you apply RL to a new problem is to do automatic hyperparameter optimization.

REWARDS:

Touch point

Touch ground

Get fast

Save on fuel

Get closer

Small turns

Fly straight

----------------------------------------

| rollout/ | |

| ep\_len\_mean | 1.13e+03 |

| ep\_rew\_mean | 27 |

| time/ | |

| fps | 1361 |

| iterations | 156 |

| time\_elapsed | 234 |

| total\_timesteps | 319488 |

| train/ | |

| approx\_kl | 0.32502812 |

| clip\_fraction | 0.606 |

| clip\_range | 0.2 |

| entropy\_loss | -0.158 |

| explained\_variance | 0.976 |

| learning\_rate | 0.000894 |

| loss | -0.025 |

| n\_updates | 31690 |

| policy\_gradient\_loss | -0.00382 |

| std | 0.252 |

| value\_loss | 0.0554 |

rollout/

ep\_len\_mean: Mean episode length (averaged over 100 episodes)

ep\_rew\_mean: Mean episodic training reward (averaged over 100 episodes)

time/

fps: Number of frames per seconds (includes time taken by gradient update)

iterations: Number of iterations (data collection + policy update for A2C/PPO)

time\_elapsed: Time in seconds since the beginning of training

total\_timesteps: Total number of timesteps (steps in the environments)

train/

approx\_kl: approximate mean KL divergence between old and new policy (for PPO), it is an estimation of how much changes happened in the update

clip\_fraction: mean fraction of surrogate loss that was clipped for PPO.

clip\_range: Current value of the clipping factor for the surrogate loss of PPO

entropy\_loss: Mean value of the entropy

explained\_variance: Fraction of the return variance explained by the value [function](https://scikit-learn.org/stable/modules/model_evaluation.html#explained-variance-score), (ev=0 => might as well have predicted zero, ev=1 => perfect prediction, ev<0 => worse than just predicting zero)

learning\_rate: Current learning rate value

loss: Current total loss value

n\_updates: Number of gradient updates applied so far

policy\_gradient\_loss: Current value of the policy gradient loss (its value does not have much meaning)

value\_loss: Current value for the value function loss for on-policy algorithms, usually error between value function output and Monte-Carlo estimate (or TD(lambda) estimate)

std: Current standard deviation of the noise when using generalized State-Dependent Exploration (gSDE)

DOCS:

Physics engine - https://jsbsim-team.github.io/jsbsim-reference-manual/

Graphics Engine - https://www.flightgear.org/

Resources:

1. <https://github.com/zgoddard3/jsbsim-gym>
2. <https://github.com/hill-a/stable-baselines>
3. <https://github.com/JSBSim-Team/jsbsim>
4. <https://github.com/JSBSim-Team/jsbsim/tree/master/UnrealEngine>
5. <https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html>
6. <https://www.gymlibrary.dev/>
7. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>
8. <https://stable-baselines3.readthedocs.io/en/master/common/logger.html>

Videos:

1. https://www.youtube.com/watch?v=XbWhJdQgi7E