MIMo Environment Documentation

Setting up the MIMo Environment

The MIMo Environment is wrapped in OpenAI Gym standard which is de facto standard for representing Environments for training reinforcement Learning algorithms.

After cloning the Repository and installing it in local computer, the environment can be used in Python

```
import gymnasium as gym
import mimoEnv
env = gym.make("MIMoSelfBody-v0", render_mode="rgb_array")

#To Render the Environment in MuJoCo
env.reset()
env.render()
```

Observation Space

The Observation Space contains state vector that the Agent can be in at any given time step. MIMo environment has following observation space representation.

**Dict('achieved_goal': Box(-inf, inf, (37,), float64), 'desired_goal': Box(-inf, inf, (37,), float64), 'observation': Box(-inf, inf, (22
0,), float64), 'touch': Box(-inf, inf, (1890,), float32))**

The desired goal is one hot encoding representation for IDs of active geometries. The active geometries of MIMo Environment have following one hot encoding representation.

[22, 18, 21, 17, 20, 16, 2, 3, 4, 5, 12, 13, 11]

Where each ID in above array corresponds to following body parts.

['right_lower_leg', 'left_hand', 'right_upper_leg', 'left_hand', 'left_fingers', 'left_lower_arm', 'lower_body', 'upper_body', 'upper_body', 'upper_body', 'right_hand', 'right_fingers', 'right_hand']

Desired and Achieved Goal:

At each episode the desired goal is randomly sampled. For example the desired goal can be represented by array or following form.

[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

In the above array, the 3rd index is 1 which mean for given episode, the desired geometry is body part that corresponds to 3rd index, which is "upper body".

Once this goal is achieved, the episode terminates to receive reward of 500

Sample Observation Space

OrderedDict([('achieved_goal', array([-0.43625054, -0.07666568, -2.486077 , -1.42548809, -0.87853917,
0.55337786, 1.19718672, -0.44782168, -0.38249367, -0.38385445,
-0.64766705, -1.21438378, 1.30009414, 0.12324859, 1.37860532,
1.01369308, 1.11425214, 0.85435809, 0.92263914, 0.68341858,
-0.88965829, 0.20710376, 1.04853044, -1.36891252, 0.42770672,
0.26553167, 0.690526 , 2.39999194, 1.18116166, -1.8277683 ,
1.7146916 , 0.95912712, -0.14764002, 1.33542677, 0.55306803,
-1.21372438, -0.56289931])), ('desired_goal', array([0.47059163, 0.78650958, 0.10435072, -0.19956708, 0.89409657,
0.24221997, 0.82140956, 0.3196907 , -0.03916643, 1.20533585,
-1.03947732, 0.62389296, -1.19237693, 1.58855715, 0.08148209,
-1.90359604, 0.77183214, -0.17806088, -0.163523 , -2.04265812,
0.80481987, -1.03534359, -0.93603253, -0.52832596, -0.25270413,
1.37429024, -2.08702321, 0.21513068, -0.99229857, -0.90053623,
0.32011863, -0.21388811, 1.62519006, 0.44997 , 1.88643455,
-1.00046034, 0.0199473])), ('observation', array([1.67746330e+00, 4.22464478e-01, 5.67542979e-01, -1.04070495e+00,
5.68867280e-01, -1.95771361e-01, -9.92536077e-04, -3.62006970e-01,
8.76777359e-01, -2.77652809e-01, 6.25477060e-01, -2.58755894e-01,
-4.08064445e-01, 1.09949695e+00, -1.13527315e+00, -6.28936126e-01,
-5.68781978e-01, 1.19469861e+00, 7.97302027e-01, -2.38058207e+00,
3.80768163e-01, 6.50386102e-01, 3.48935373e-01, -1.44255896e-02,
1.99878701e+00, 1.70734744e+00, 2.21343072e+00, -1.99852598e+00,
1.62884651e-01, 1.13626633e+00, -1.96285930e+00, 1.80540377e+00,
-6.15609585e-01, 1.67020319e+00, -1.71652027e+00, 5.88358846e-01,
-1.66377321e+00, 1.17183931e+00, -3.95492253e-01, -1.37664172e-01,
1.89623717e+00, 3.99420418e-02, -9.91095655e-01, -6.12239598e-01,
5.22959273e-03, 4.35594954e-01, 5.18539708e-01, -4.68828002e-01,
7.33413867e-01, -7.24413792e-02, -5.23260316e-01, 1.17799937e+00,
1.04138355e+00, 1.97971233e-01, -3.44435328e-01, -1.93558968e+00,
1.62129378e+00, 3.25450573e-01, -2.42361671e+00, -2.37185320e-01,
1.07532810e+00, 5.31488399e-01, 7.08634990e-01, -1.15034125e+00,
-1.00517664e+00, 7.98645143e-01, -1.96936445e-01, 5.33356436e-01,
-1.43548160e-01, -1.66601454e+00, 2.03438907e-01, -3.74376068e-01,
-1.37888410e+00, 2.47907228e-01, -1.49816891e+00, 7.65580478e-01,
2.77065869e+00, 7.32555129e-01, 1.04366052e+00, -8.30922952e-01,
1.29089602e+00, -9.09935534e-01, -5.82995266e-01, 6.92566819e-01,
-2.81929426e-02, -7.73501098e-01, 6.68035809e-01, 8.15728949e-01,
6.03506851e-01, 1.62373666e+00, 1.43961771e+00, -2.96475550e-01,
2.62744427e-02, -7.19515395e-01, 2.26509526e-01, -1.49254422e+00,
8.22120620e-01, -9.66971073e-01, 8.15630722e-01, 7.50225870e-01,
-6.76421797e-01, -9.84370848e-01, -7.65888342e-01, 4.96121888e-01,
-1.31746171e-02, -5.41360670e-01, -1.12964791e+00, -1.50838645e+00,
2.07251107e+00, -5.13536898e-01, -2.36342055e+00, -8.79153065e-01,
2.61891860e+00, 6.01609755e-01, -1.22831606e+00, -1.36548190e+00,
-4.74372947e-01, -6.38343868e-01, 6.43582168e-01, -2.04875807e+00,
-1.34611382e+00, 1.62060078e+00, -4.64715964e-01, 3.65816798e-01,

```
   -1.63227002e+00,  3.02386727e-01,  1.12931503e+00,  5.59990079e-01,
    1.15484871e+00, -7.98928050e-01,  9.69187423e-01,  2.85075929e-01,
    1.26481671e-01,  6.26571552e-01, -5.72952818e-01, -6.80844314e-02,
    2.09696057e-01, -5.27729506e-02,  2.02254784e-01,  3.10137693e-02,
    4.12919284e-01,  3.36640674e-01,  1.37283023e+00, -1.19218941e-01,
    1.62994165e-01,  4.09458253e-01,  1.48418626e+00,  8.49293706e-01,
    3.73826648e-02,  5.12735542e-01,  1.01328343e+00,  5.39143976e-01,
    1.35700907e+00,  2.92393442e-01, -1.28218025e+00,  2.62514545e+00,
   -8.25421822e-01, -1.23038913e+00,  1.99932659e+00,  1.05444868e-01,
   -1.52074677e+00, -1.56666389e+00, -5.38690446e-01, -4.18430671e-01,
   -1.53663305e+00,  5.39352885e-02,  8.30303124e-01, -1.88077733e+00,
   -2.34471785e+00, -1.16689105e-01, -9.49216875e-01,  3.40113646e-02,
   -1.50415228e-01, -1.25230587e+00,  2.32774556e+00, -8.04896774e-01,
   -5.96349186e-01,  1.83153033e+00, -2.73992093e-01, -2.37908791e-01,
    2.04194146e-01,  1.50871772e-01,  9.70812856e-01, -2.15979795e-01,
   -4.50075934e-01, -1.81850557e+00, -1.91317381e+00,  5.03358974e-01,
   -1.62022717e+00,  8.81096788e-01,  1.13137665e+00,  6.02256347e-01,
    9.95053401e-01, -3.43470909e-01, -8.44540943e-03, -8.60361485e-01,
   -1.35354933e+00,  1.14421654e+00, -5.58217600e-01,  2.17260849e+00,
    5.98484999e-01, -1.96224361e-01,  1.90827368e-02,  1.17146679e-02,
   -1.37553458e+00, -4.82504168e-01, -1.46172588e+00,  2.15057846e+00,
   -8.86611492e-01,  1.45556785e-01, -4.66270741e-01,  4.07354763e-01,
    1.95209672e+00,  9.78116133e-01,  8.53123916e-01, -6.71687925e-01,
    6.47335766e-01, -2.15424038e-02, -4.00284770e-02, -1.15158880e-01]])), ('touch', array([
0.47887826, -0.17622866,  0.3799247 , .
..,  4.02211   ,
   -1.3237401 ,  1.0040638 ], dtype=float32))])
```

"env.sample_goal()" returns desired goal for given episode which is randomly selected for each episode

array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])

Action Space

The action space signifies the actions that agent can take at each time step. For example, in MIMo Environment, it can move multiple joints in multiple angles and orientations, so MIMo Environment has a high dimensional Action space.

Box([-1.   -0.6875 -1.   -1.   -1.   -1.   -0.87 -1.   ], [1. 1. 0.64 0.83 1. 0.57 1. 0.33], (8,), float32)

The "Box" represents that action space is continuous type rather that discrete. (8,) represents the shape of action space, or in other words the dimension of action space.

The randomly sample action, looks as follows

array([0.35703337, -0.6170899 , -0.28270736, 0.01912131, 0.7433121 ,-0.7549374 , 0.04262816, -0.6856727], dtype=float32)

Action at Time Step

The action is input to the environment at every time step. the environment outputs **observation space, reward achieved, terminated, truncated and info**

the env.step(action) function is essence of reinforcement learning environment. it takes action as an input. The action can be randomized as **action = env.action_space.sample()** or it can be selected based on some trained model such as **SAC or PPO**

Reward and Episode

The environment gives reward after each timestep, the time step is, loosely speaking, just one iteration of Loop in Python Programming. In essence, one iteration represents one time step, the environment gives reward after each time step.

If Desired body part is touched, then REWARD = 500 and Episode Terminated i.e. terminated in env.step(action) function is True

If Some Body Part Touched, but not desired body part, REWARD = -(Distance from Desired Body Part) and Episode keeps going

If No Body Part Touched at given Time Step, REWARD = -1

Termination and Truncation of Episode

The Episode ends if termination condition is achieved, i.e. the desired body part is touched as mentioned above. If desired body part is not touched after 500 Time Steps, the episode truncated, in other words, it is mandatory for agent to touch the desired body part in 500 Time steps for successful episode.

The Episode is successful if, the terminated = True is returned from step() function.

Return of Episode

The overall return (Reward) of Episode is Total Cumulative Reward of Each Time Step. The Reward of each time step is summed up to calculate overall reward for each episode.

Reinforcement Learning Algorithms

Off Policy and On Policy Algorithms

Reinforcement Learning Algorithms are categorized into two classes, namely on policy and off policy algorithms. In On Policy Algorithms, the Actions are generated based on current policy and only current policy is taken into account. There is no experience buffer to store past policy or experiences, the optimal policy is decided only based on actions taken using current policy.

In Off Policy Reinforcement Learning Algorithms, there is experience replay buffer, that is used to store past experiences. The experience based on past actions and policy is stored in replay buffer, and taken into account while deciding the optimal policy. Off Policy Algorithms are considered more sample efficient.

Soft Actor Critic (SAC)

This is an off policy algorithm, it has replay buffer, in SAC there is an actor network and critic network. The Input to Actor network is Current State of the environment and output is an action. The policy network (Or Actor) is responsible for generating action based on some policy and it also gets information from experience replay buffer.

The Input to Critic Network is State of the Environment, and the Action. The Critic Network Evaluates the action taken or generated by the actor (Policy Network) and tells the Q Value I.e. how good the action is given the current state of environment. The SAC Algorithm mostly used 2 Q Functions or Q Networks to ensemble (combining outputs of more than one neural networks to get more precise results).

Proximal Policy Optimization (PPO)

PPO is on Policy Algorithm, it does not use experience replay buffer and decides optimal policy only using the actions taken by utilizing current policy.

Stable Baselines 3

OpenAI Stable Baselines is one of the popular libraries with all state of the art algorithms combined in Reinforcement Learning.

```
pip3 install stable-baselines3
```

After Installing the library, the library can be imported in Python Code and any algorithm can be used

SAC Model Training

```
import gymnasium as gym
import mimoEnv
from stable_baselines3 import SAC

env = gym.make("MIMoSelfBody-v0", render_mode="rgb_array")
model = SAC("MlpPolicy", env, verbose=1, tensorboard_log="./tensorboard")

model.learn(1000000, log_interval=1)
model.save("sac_agent")
```

Model Inference

```
import gymnasium as gym
from stable_baselines3 import SAC
import mimoEnv

env = gym.make("MIMoSelfBody-v0", render_mode="human")

model = SAC("MultiInputPolicy", env, verbose=1)

model.load("sac_agent")
```

```python
obs, info = env.reset()
while True:
    action, _states = model.predict(obs, deterministic=True)
    obs, reward, terminated, truncated, info = env.step(action)
    print ("Reward: ", reward)
    env.render()
    if (terminated or truncated):
        break
```