**README - gym-cricket-robot**

The code has been tested on Ubuntu 18.04 with ROS Melodic and the following languages and libraries. in the following ReadMe there are some instructions on how to run and test the developed code.

**Folder’s structure**

The structure is built on the standard openAI gym type of folders:

├───res

│ ...

└───src

│ ddpg.py

│ main.py

│ out\_rew.txt

│ setup.py

│ test.py

│

├───gym\_cricket

│ │ \_\_init\_\_.py

│ │

│ ├───assests

│ │ │ cricket.py

│ │ │ cricketGoal.py

│ │ │ cricket\_abs.py

│ │ │ hebi\_cricket.py

│ │ │ hebi\_cricketGoal.py

│ │ │

│ │ ├───terrains

│ │ │ ├───flat

│ │ │ │ ...

│ │ │ └───slope

│ │ │ ...

│ │ └───urdfs

│ │ │ cricket\_robot.urdf

│ │ │ cricket\_robot.urdf.xacro

│ │ │

│ │ └───hebiCricket

│ │ ├───CAD

│ │ │ ...

│ │ ├───MATLAB Code

│ │ │ ...

│ │ └───ros\_packages

│ │ ├───hebi\_cpp\_api\_ros\_examples

│ │ ├───hebi\_description

│ │ └───hebi\_gazebo

│ └───envs

│ cricket\_env.py

│ \_\_init\_\_.py

│

├───neural\_network

│ actor\_nn.py

│ critic\_nn.py

│

├───utils

│ auxiliaryFuncs.py

│ buffer.py

│ evaluator.py

│ memory.py

│ OUNoise.py

│ random\_process.py

│ util.py

│

└───weights\_out

actor.pkl

critic.pkl

**Requirements**

**Languages:**

* python 3.6.4 or newer
* C++ (to compile HEBI robotics packages if needed)

**Libraries (not std)**

* gym
* pybullet
* numpy
* torch
* pywavefront
* matplotlib
* argparse
* setuptools
* pathlib

**How to run**

Run python main.py (the python command might change based on your version and the number of python versions installed).

**Flags:**

Check the code for the default values.

**Environment arguments**

* --mode support option: train/test
* --env open-ai gym environment
* --num\_apisode total training episodes
* --step\_episode simulation steps per episode
* --early\_stop change episode after [early\_stop] steps with a non-growing reward
* --cricket [hebi\_cricket, basic\_cricket] - cricket urdf model you want to load
* --terrain name of the terrain you want to load (to be implemented)

**Reward function**

* --w\_X weight X to compute difference between the robot and the optimal position.
* --w\_Y weight Y to compute difference between the robot and the optimal position.
* --w\_Z weight Z to compute difference between the robot and the optimal position.
* --w\_theta weight theta to compute difference between the robot and the optimal position.
* --w\_sigma weight sigma to compute difference between the robot and the optimal position.
* --disct\_factor discount factor for learning in the reward function
* --w\_joints weight to punish bad joints behaviors in the reward function

**Neural networks**

* --hidden1 hidden num of first fully connect layer
* --hidden2 hidden num of second fully connect layer
* --hidden3 hidden num of third fully connect layer
* --hidden4 hidden num of fourth fully connect layer
* --hidden5 hidden num of fifth fully connect layer
* --conv\_hidden1 hidden num of first convolutional layer
* --conv\_hidden2 hidden num of second convolutional layer
* --conv\_hidden3 hidden num of third convolutional layer
* --conv\_hidden4 hidden num of fourth convolutional layer
* --conv\_hidden5 hidden num of fifth convolutional layer
* --kernel\_size1 num of first kernel for cnn
* --kernel\_size2 num of second kernel for cnn
* --kernel\_size3 num of third kernel for cnn
* --kernel\_size4 num of fourth kernel for cnn

**Ddpg arguments**

* --bsize minibatch size
* --rate learning rate
* --prate policy net learning rate (only for DDPG)
* --warmup time without training but only filling the replay memory
* --discount discount factor
* --rmsize memory size
* --window\_length
* --tau moving average for target network
* --ou\_theta noise theta
* --ou\_sigma noise sigma
* --ou\_mu noise mu

**To be optimized**

The following flags are inserted but (some of them) not used in the code.

* --validate\_episodes how many episodes to perform during validate experiment
* --max\_episode\_length how many steps to perform a validate experiment
* --validate\_steps train iters each timestep
* --output linear decay of exploration policy
* --debug Resuming model path for testing
* --init\_w
* --train\_iter
* --epsilon
* --seed
* --resume

**How to change terrain**

The flat terrain is the flag terrain default value. To use another terrain you need to put the files required (\*.mtl, \*.obj, \*.urdf) inside a folder with the same name of the terrain, under the subfolder src/gym\_cricket/assests/terrains/

e.g., I want to use a new terrain named "slope"

* I create a new folder under src/gym\_cricket/assests/terrains/ named slope
* I put all the useful files inside that folder: at least *slope.mtl, slope.obj, slope.urdf* (and in case other useful files)
* I run the script with the correct flag python main.py --terrain="slope"

**How to customize the neural networks**

The default neural networks\* are the following:

* FC 400x300x150 🡪 it is possible to add layers **up to a total of 5 hidden layers** (by changing the code it is easy to add more)
* 3D-CNN 🡪 the default three dimensional CNN is very simple composed just by the input layer and a squared kernel 1x1x1. If you want to add more convolutional layer you also need to mind the respective **kernel sizes**. Again the maximum number of hidden layers is 5.

\* To understand how the neural networks work, give a look to the report with the explanation of the algorithm

# Classes explanation

Under *res* folder there are some useful resources as the report and the pseudocode.

Under *src* folder there is the code explained below:

* **ddpg.py** contains the core of the main algorithm.
* **main.py** contains the logic that connect the environment to the algorithm.
* **gym\_cricket/** this folder contains the parts of the code that communicate with the simulation environment.
  + **assests/** this folder containsthe *terrains*, the *urdf models*, and the classes that describe the robot itself and its characteristics inside the simulation.
    - **cricket\_abs.py** this is an abstract class containing the main function used by the simulation to obtain information about the robot. It contains the getter and the setter for the joint positions, the getters for the normal forces, the collisions etc.
    - **cricket.py** implements the abstract class ***cricket\_abs.*** This is a simplified version of the cricket robot. Useful for test and debug, because light for the simulation.
    - **hebi\_cricket.py** implements the abstract class ***cricket\_abs.*** The code describes and represent the robot developed by HEBI Robotics.
    - **cricketGoal.py** and **hebi\_cricketGoal.py** contain the final desired position for the cricket. Used by the reward function (see report)
    - **terrains/**

contains the folders for the different kinds of terrain

* + - **urdf/**

contains the *.urdf* and *.xacro* files for the simplified cricket robot, and the folder that incorporates the HEBI Robotics’ cricket robot.

* + **envs/**
    - **cricket\_env.py** contains the logic of the environment. Hence, the reward function, the logic behind the steps in the simulation, and control the state of the cricket robot.
* **neural\_networks/** see the ddpg algorithm for the actor-critic network
  + **actor\_nn.py**
  + **critic\_nn.py**
* **utils/**
  + this folder contains all the auxiliary classes and functions (as the buffer for the ddpg or function useful for CUDA).
* **weights\_out/** 
  + the folders contain the output of the algorithm. Hence, the actor and critic neural networks weights.

# Not implemented

The following are process or component not fully implemented in the current code. Everything already works without these tunings.

## Final goal read from an external file

Currently, the goal position (see the report) is represented in the code as a list of joint angles which represent the final desired joint angles. It is possible to implement a reading function that, from the main.py script, reads a list of values from an external file (.txt, .json, or similar) as it does now for the different kind of terrains.