**Action Space**

References:

1. [Genesis-world Read Docs Control Your Robot](https://genesis-world.readthedocs.io/en/latest/user_guide/getting_started/control_your_robot.html#joint-control)

2. [Genesis-Embodied-AI Examples/locomotion Git Repo](https://github.com/Genesis-Embodied-AI/Genesis/tree/main/examples/locomotion)

The Go2 robot has 12 joints. At every time instant, the Reinforcement Learning (RL) agent has to provide control signals to the mentioned 12 joints in order to actuate them to achieve the desired positions.

The following code will help to understand how to construct the action space for the Go2 robot in the gym environment.

As usual, at first, the scene is built in Genesis to integrate the robot xml file as follows.

# Importing libraries

import genesis as gs

#Initializes Genesis with the CPU backend.

gs.init(backend=gs.cpu)

#Create a Scene

scene = gs.Scene(show\_viewer=True)

#Adds a flat ground plane to the scene.

plane = scene.add\_entity(gs.morphs.Plane())

#Integrate the Go2 Robot xml.

robot = gs.morphs.MJCF(file="xml/Unitree\_Go2/go2.xml")

#Add an entity to the scene.

Go2 = scene.add\_entity(robot)

#Builds the scene.

scene.build()

When running the above code, a pop-up video terminal should appear showing a stand-still Go2 robot.

Once the robot xml is integrated, the joint IDs need to be defined. The joint names are defined as follows.

# Declaring the joint names in the desired order

joint\_names = [

'FL\_hip\_joint', 'FR\_hip\_joint', 'RL\_hip\_joint', 'RR\_hip\_joint',

'FL\_thigh\_joint', FR\_thigh\_joint', 'RL\_thigh\_joint', 'RR\_thigh\_joint',

'FL\_calf\_joint', 'FR\_calf\_joint', 'RL\_calf\_joint', 'RR\_calf\_joint', ]

\* The above-mentioned joint names can be found in the xml file.

Next, the Motor Degree of Freedom IDs, motor\_dof\_idx is defined as follows,

# Defining the Joint Name IDs

motors\_dof\_idx = [Go2.get\_joint(name).dof\_start for name in joint\_names]

# Printing the Joint Name IDs

motors\_dof\_idx

The above code should generate an output as follows.

[0, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]

Once the dof Idx is declared, the joints can be controlled in the simulation in the following ways

1. Hard Reset
2. Controlling joint velocity
3. Controlling position

Out of the three options mentioned above, the last one, i.e., setting the position, is the most realistic and closely mimics the true replication of a real robot.

In the third method, a target position, i.e., target\_dof\_pos, is applied to the robot Go2. The built-in PD controller of Genesis tracks the input target position. The command is as follows,

Go2.control\_dofs\_position(target\_dof\_pos, motors\_dof\_idx)

The target\_dof\_pos is a 12-element array that provides the target position of the joints in radians.

\* An important point needs to be mentioned here. To use the control\_dofs\_position method, the kp and kd gains of the dofs must be set as follows.

Go2.set\_dofs\_kp(np.array([12 element array containing the kp gains]), motors\_dof\_idx) Go2.set\_dofs\_kv(np.array([12 element array containing the kd gains]), motors\_dof\_idx)

Next, the controller’s input to the robot should be maintained within the minimum and maximum limits. So, it is essential to clip the input using upper and lower bounds. The upper and lower bounds can be found from the xml files as follows.

There exist the following classes of joints

1. class abduction joints (all 4 hips): FL\_hip\_joint, FR\_hip\_joint, RL\_hip\_joint, RR\_hip\_joint, Range= [-1.0472, 1.0472]

2. class front\_hip joints (2 front thighs): FL\_thigh\_joint, FR\_thigh\_joint, Range= [-1.5708 3.4907]

3. class back\_hip joints (2 rear thighs): RL\_thigh\_joint, RR\_thigh\_joint, Range= [-0.5236 4.5379]

4. class knee (all 4 calfs): FL\_calf\_joint, FR\_calf\_joint, RL\_calf\_joint, RR\_calf\_joint, Range= [-2.7227 -0.83776]

Finally the action space is defined as follows

# Joint position limits in radians (based on the Go2 xml)

action\_low = np.array([-1.0472, -1.0472, -1.0472, -1.0472, -1.5708, -1.5708, -0.5236, -0.5236, -2.7227, -2.7227, -2.7227, -2.7227], dtype=np.float32)

action\_high = np.array([1.0472, 1.0472, 1.0472, 1.0472, 3.4907, 3.4907, 4.5379, 4.5379, -0.83776, -0.83776, -0.83776, -0.83776], dtype=np.float32)

# Create the action space

action\_space = gym.spaces.Box(low=action\_low, high=action\_high, dtype=np.float32)