

A Appendix

A.1 Full Prompts

In this section, we provide all the prompts for training with LAPP.

Prompt for Flat Plane

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories and decide which one is better in each pair.

Your feedback of the comparisons will be used as a reward signal (for reinforcement learning) to train a quadruped robot (Unitree Go2) to walk forward at some speed given by the commands, and the velocity range of the speed command is $[0.0, 2.2]$ m/s.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot moving on a flat ground.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is $[0.0, 2.2]$ m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
- 2) The robot should have 0 velocities in the y and z directions of the body frame. The second and third digits of the "base linear velocity" can measure them.
- 3) The robot should keep its body torso near the height of 0.34 meter. The "base height" value can measure the robot torso height.
- 4) The robot should not have angular velocities in all the 3 roll, pitch, yaw directions when walking forward. The 3 values of the "base angular velocity" should be close to 0.
- 5) The robot should not have roll or pitch angles when walking forward. Since the linear and angular velocities of the robot are randomly initialized at each episode, the robot might have some yaw angle from start, but this yaw angle should not change when the robot is walking forward.
- 6) The robot is encouraged to take longer steps instead of small steps. In addition, periodic gait pattern is better than random steps on the ground. The "feet contacts" can be used to analyze the gait pattern of the robot.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.

- 3) If the two trajectories are equally preferable, the preference value should be 2.
 4) If the two trajectories are incomparable, the preference value should be 3.
 Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3].
 Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.
 Please provide preference values 0 and 1 as many as possible, which clearly indicate which one is better in a pair.
 Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.
 Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Stairs

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories and decide which one is better in each pair.

Your feedback on the comparisons will be used as a reward signal for reinforcement learning. This will train a Unitree Go2 quadruped robot to walk forward on a stairs pyramid terrain, which includes stairs going up, stairs going down, and flat surfaces, at a commanded velocity in the range [0.0, 2.2] m/s.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot moving on a flat ground.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is [0.0, 2.2] m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 5) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 6) "ground height": the z position (height) of the terrain ground right beneath the center of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 7) "feet heights": the four height values of the four feet. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].
- 8) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
- 2) The robot should have 0 velocities in the y direction of the body frame. The second digit of the "base linear velocity" can measure them.

- 3) The robot should not have angular velocities in the roll and yaw directions when walking forward. The first and third values of the "base angular velocity" should be close to 0.
- 4) The robot should keep its base height about 0.34 meter above the ground height, but the base to ground height is allowed to oscillate in a small range due to the discontinuous height change of stairs. Compare the "base height" and "ground height" values to measure this.
- 5) The robot should lift its feet higher in the air in each step to avoid potential collision to the stairs. Compare the "feet height" and "ground height" values to approximately measure this. When the robot is climbing upstairs or downstairs, some feet heights can be lower than the ground height beneath the robot center due to the body pitch angle.
- 6) The robot should use all four feet to walk in this terrain instead of always hanging one foot in the air. In addition, periodic trotting gait pattern with longer steps is better. The "feet contacts" can be used to analyze the gait pattern of the robot.
- 7) The robot should have 0 roll angle when walking forward. Pitch angle is allowed for climbing upstairs and downstairs. Since the linear and angular velocities of the robot are randomly initialized at each episode, the robot might have some yaw angle from start, but this yaw angle should not change when the robot is waling forward.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.
- 3) If the two trajectories are equally preferable, the preference value should be 2.
- 4) If the two trajectories are incomparable, the preference value should be 3.

PLease note that the robot should should use all four feet to walk. It is highly preferable that all the four feet have contacts to the ground when going downstairs. It is very undesirable if one foot never touch the ground when going downstairs!

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Obstacles

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train a quadruped robot to walk forward at some speed given by the commands, with speed range of [0.0, 2.2] m/s.

Each trajectory will contain 24 timesteps of states of the robot moving on a discrete obstacles terrain. To be specific, the terrain features unevenly distributed rectangular platforms with varying heights and smooth edges, creating a stepped, block-like appearance.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is [0.0, 2.2] m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.

3) "base angular velocity": the raw, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and raw, pitch, yaw 3 angular velocities around the x, y, z axes.

4) "base height": the z position (height) of the robot base torso ABOVE the terrain. The data shape is (24,), standing for the 24 steps of a trajectory.

5) "base angular orientation": the raw, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and raw, pitch, yaw 3 rotation angles around the x, y, z axes.

6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.

2) The robot should have no velocity in y axis of the base torso. The second digit of "base linear velocity" can measure.

3) The robot should keep its body torso near the height of 0.34 meter. "base height" can measure.

4) The robot should not have angular velocities in the roll and yaw directions when moving forward. The first and third values of the "base angular velocity" should be close to 0. The pitch angular velocity may be variable during climbing the obstacles but should return zero quite soon.

5) The robot should not have roll angle when moving forward. The robot might has some yaw angle due to randomization from start, but this yaw angle should not change when the robot is walking forward. Small pitch orientation is acceptable so as to adapt to the terrain.

6) The robot is encouraged to take a ****trotting**** gait to move forward. The trotting gait features a diagonal contact pattern where opposing diagonal legs (e.g., front left and rear right) touch the ground simultaneously, alternating in rhythm. The "feet contacts" can be used to analyze the gait pattern of the robot.

7) The robot is encouraged to take farther steps. "feet contacts" can help measure.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

1) If the trajectory 0 is better, the preference value should be 0.

2) If the trajectory 1 is better, the preference value should be 1.

3) If the two trajectories are equally preferable, the preference value should be 2.

4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

1) If both can move forward, the one with greater velocity in x axis is better.

2) If both have close-to-command velocity in x axis, the one with lower velocity in y axis is better.

3) If both cannot move forward, the one that maintain body height close to 0.34 meter is better.

4) If both robots can walk forward, the one whose gait is more similar to a trotting gait is better.

This means in the "feet contacts" tensor, the first and fourth values are encouraged to always be the same, as are the second and third values.

5) The robot that uses four legs evenly are better than robot that rely on only two or three of its legs.

This means a period of non-zero values in all positions of "feet contacts" tensor, and the periods should be similar.

6) The robot that takes longer steps are better. This means longer period is preferable.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

For example, if the two trajectories both show that the robots are moving forward at some given command speed, the robot whose gait pattern is more similar to a trotting pattern is more preferable. Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Obstacles

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train a quadruped robot to walk forward at some speed given by the commands, with speed range of $[0.0, 2.2]$ m/s.

Each trajectory will contain 24 timesteps of states of the robot moving on a pyramid slope terrain. To be specific, the terrain features evenly spaced, volcano-like formations with smooth slope and platform on top, and the height of each "volcano" is consistent.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is $[0.0, 2.2]$ m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso ABOVE the terrain. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base angular orientation": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
- 2) The robot should have no velocity in y axis of the base torso. The second digit of "base linear velocity" can measure.
- 3) The robot should keep its body torso near the height of 0.34 meter. "base height" can measure.
- 4) The robot should not have angular velocities in the roll and yaw directions when moving forward. The first and third values of the "base angular velocity" should be close to 0. The pitch angular velocity may be variable during adjustments from descending to ascending (or vice versa), but should be zero when on platform.
- 5) The robot should not have roll angle when moving forward. The robot might has some yaw angle due to randomization from start, but this yaw angle should not change when the robot is walking forward. Small pitch orientation is acceptable so as to adapt to the terrain.
- 6) The robot is encouraged to take a **trotting** gait to move forward. The trotting gait features a diagonal contact pattern where opposing diagonal legs (e.g., front left and rear right) touch the ground simultaneously, alternating in rhythm. The "feet contacts" can be used to analyze the gait pattern of the robot.

7) The robot is encouraged to take farther steps. "feet contacts" can help measure. The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.
- 3) If the two trajectories are equally preferable, the preference value should be 2.
- 4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

- 1) If both can move forward, the one with greater velocity in x axis is better.
- 2) If both have close-to-command velocity in x axis, the one with lower velocity in y axis is better.
- 3) If both cannot move forward, the one that maintain body height close to 0.34 meter is better.
- 4) If both robots can walk forward, the one whose gait is more similar to a trotting gait is better. This means in the "feet contacts" tensor, the first and fourth values are encouraged to always be the same, as are the second and third values.
- 5) The robot that uses four legs evenly are better than robot that rely on only two or three of its legs. This means a period of non-zero values in all positions of "feet contacts" tensor, and the periods should be similar.
- 6) The robot that takes longer steps are better. This means longer period is preferable. Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

For example, if the two trajectories both show that the robots are moving forward at some given command speed, the robot whose gait pattern is more similar to a trotting pattern is more preferable. Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Slope

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories. Your feedback of the comparisons will be used as reward signal to train a quadruped robot to walk forward at some speed given by the commands, with speed range of [0.0, 2.2] m/s. Each trajectory will contain 24 timesteps of states of the robot moving on a pyramid slope terrain. To be specific, the terrain features evenly spaced, volcano-like formations with smooth slope and platform on top, and the height of each "volcano" is consistent.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is [0.0, 2.2] m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.

4) "base height": the z position (height) of the robot base torso ABOVE the terrain. The data shape is (24,), standing for the 24 steps of a trajectory.

5) "base angular orientation": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.

6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.

2) The robot should have no velocity in y axis of the base torso. The second digit of "base linear velocity" can measure.

3) The robot should keep its body torso near the height of 0.34 meter. "base height" can measure.

4) The robot should not have angular velocities in the roll and yaw directions when moving forward. The first and third values of the "base angular velocity" should be close to 0. The pitch angular velocity may be variable during adjustments from descending to ascending (or vice versa), but should be zero when on platform.

5) The robot should not have roll angle when moving forward. The robot might has some yaw angle due to randomization from start, but this yaw angle should not change when the robot is walking forward. Small pitch orientation is acceptable so as to adapt to the terrain.

6) The robot is encouraged to take a ****trotting**** gait to move forward. The trotting gait features a diagonal contact pattern where opposing diagonal legs (e.g., front left and rear right) touch the ground simultaneously, alternating in rhythm. The "feet contacts" can be used to analyze the gait pattern of the robot.

7) The robot is encouraged to take farther steps. "feet contacts" can help measure.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

1) If the trajectory 0 is better, the preference value should be 0.

2) If the trajectory 1 is better, the preference value should be 1.

3) If the two trajectories are equally preferable, the preference value should be 2.

4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

1) If both can move forward, the one with greater velocity in x axis is better.

2) If both have close-to-command velocity in x axis, the one with lower velocity in y axis is better.

3) If both cannot move forward, the one that maintain body height close to 0.34 meter is better.

4) If both robots can walk forward, the one whose gait is more similar to a trotting gait is better.

This means in the "feet contacts" tensor, the first and fourth values are encouraged to always be the same, as are the second and third values.

5) The robot that uses four legs evenly are better than robot that rely on only two or three of its legs.

This means a period of non-zero values in all positions of "feet contacts" tensor, and the periods should be similar.

6) The robot that takes longer steps are better. This means longer period is preferable.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

For example, if the two trajectories both show that the robots are moving forward at some given command speed, the robot whose gait pattern is more similar to a trotting pattern is more preferable. Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Wave

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train a quadruped robot to walk forward at some speed given by the commands, with speed range of $[0.0, 2.2]$ m/s.

Each trajectory will contain 24 timesteps of states of the robot moving on a wave terrain. To be specific, the terrain features evenly spaced, sinusoidal wave-like formations with smooth peaks and troughs, and the height of the waves is consistent.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is $[0.0, 2.2]$ m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso ABOVE the terrain. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base angular orientation": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
- 2) The robot should have no velocity in y axis of the base torso. The second digit of "base linear velocity" can measure.
- 3) The robot should keep its body torso near the height of 0.34 meter. "base height" can measure.
- 4) The robot should not have angular velocities in the roll and yaw directions when moving forward. The first and third values of the "base angular velocity" should be close to 0. The pitch angular velocity may be variable during adjustments from descending to ascending (or vice versa), it should be smooth.
- 5) The robot should not have roll angle when moving forward. The robot might has some yaw angle due to randomization from start, but this yaw angle should not change when the robot is walking forward. Small pitch orientation is acceptable so as to adapt to the terrain.
- 6) The robot is encouraged to take a **trotting** gait to move forward. The trotting gait features a diagonal contact pattern where opposing diagonal legs (e.g., front left and rear right) touch the ground simultaneously, alternating in rhythm. The "feet contacts" can be used to analyze the gait pattern of the robot.
- 7) The robot is encouraged to take farther steps. "feet contacts" can help measure.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.
- 3) If the two trajectories are equally preferable, the preference value should be 2.
- 4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

- 1) If both can move forward, the one with greater velocity in x axis is better.
- 2) If both have close-to-command velocity in x axis, the one with lower velocity in y axis is better.
- 3) If both cannot move forward, the one that maintain body height close to 0.34 meter is better.
- 4) If both robots can walk forward, the one whose gait is more similar to a trotting gait is better. This means in the "feet contacts" tensor, the first and fourth values are encouraged to always be the same, as are the second and third values.
- 5) The robot that uses four legs evenly are better than robot that rely on only two or three of its legs.

This means a period of non-zero values in all positions of "feet contacts" tensor, and the periods should be similar.

- 6) The robot that takes longer steps are better. This means longer period is preferable.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

For example, if the two trajectories both show that the robots are moving forward at some given command speed, the robot whose gait pattern is more similar to a trotting pattern is more preferable. Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Bounding Gait

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories and decide which one is better in each pair.

Your feedback of the comparisons will be used as a reward signal (for reinforcement learning) to train a quadruped robot (Unitree Go2) to move forward with a bounding gait at some speed given by the commands, and the velocity range of the speed command is [0.0, 2.2] m/s.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot moving on a flat ground.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is [0.0, 2.2] m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.

4) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.

5) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.

6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.

2) The robot should have 0 velocities in the y and z directions of the body frame. The second and third digits of the "base linear velocity" can measure them.

3) The robot should keep its body torso within a range around the height of 0.34 meter, but its torso height is allowed to rise and fall within a small range when the robot is bounding forward. The "base height" value can measure the robot torso height.

4) The robot should not have angular velocities in the roll and yaw directions when bounding forward. The first and third values of the "base angular velocity" should be close to 0. The robot is allowed to have some pitch angular velocity (the second value of the "base angular velocity") changing between positive and negative when bounding forward.

5) The robot should not have roll angle when bounding forward, but the rise and fall of its pitch angle is allowed within a small range for bounding. Since the linear and angular velocities of the robot are randomly initialized at each episode, the robot might have some yaw angle from start, but this yaw angle should not change when the robot is waling forward.

6) The robot is encouraged to take a bounding gait to move forward. The "feet contacts" can be used to analyze the gait pattern of the robot. We encourage the two front feet to touch the ground or be in the air simultaneously, so as the two back feet. I.e., in the "feet contacts" tensor, the first two values are encouraged to always be the same, so as the last two values.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

1) If the trajectory 0 is better, the preference value should be 0.

2) If the trajectory 1 is better, the preference value should be 1.

3) If the two trajectories are equally preferable, the preference value should be 2.

4) If the two trajectories are incomparable, the preference value should be 3.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

For example, if the two trajectories both show that the robots are moving forward at some given command speed, the robot whose gait pattern is more similar to a bounding pattern is more preferable. Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for High Cadence

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories and decide which one is better in each pair.

Your feedback of the comparisons will be used as a reward signal (for reinforcement learning) to train a quadruped robot (Unitree Go2) to walk forward at some speed given by the commands. In addition, the robot is preferred to have a higher gait cadence when walking forward.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot moving on a flat ground.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is $[0.0, 2.2]$ m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
 - 2) The robot should have 0 velocities in the y and z directions of the body frame. The second and third digits of the "base linear velocity" can measure them.
 - 3) The robot should keep its body torso near the height of 0.34 meter. The "base height" value can measure the robot torso height.
 - 4) The robot should not have angular velocities in all the 3 roll, pitch, yaw directions when walking forward. The 3 values of the "base angular velocity" should be close to 0.
 - 5) The robot should not have roll or pitch angles when walking forward. Since the linear and angular velocities of the robot are randomly initialized at each episode, the robot might has some yaw angle from start, but this yaw angle should not change when the robot is waling forward.
 - 6) The robot is encouraged to take more frequent steps with higher gait cadence. The "feet contacts" can be used to analyze the gait pattern of the robot. Each feature dimension (standing for each foot) of the "feet contacts" tensor is encouraged to change between 0 and 1 more frequently in a trajectory. The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).
- 1) If the trajectory 0 is better, the preference value should be 0.
 - 2) If the trajectory 1 is better, the preference value should be 1.
 - 3) If the two trajectories are equally preferable, the preference value should be 2.
 - 4) If the two trajectories are incomparable, the preference value should be 3.

Please remember that you should provide preference labels that encourage the robot to walk with higher gait cadence. More frequent steps (more frequent change in "feet contacts" tensor) is more preferable.

Please give response with only one list of 5 preference values, e.g., $[0, 0, 1, 2, 3]$. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indicate which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Low Cadence

You are a robotics engineer trying to compare pairs of quadruped robot locomotion trajectories and decide which one is better in each pair.

Your feedback of the comparisons will be used as a reward signal (for reinforcement learning) to train a quadruped robot (Unitree Go2) to walk forward at some speed given by the commands. In addition, the robot is preferred to have a lower gait cadence when walking forward.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot moving on a flat ground.

The state includes:

- 1) "commands": the linear velocity command along x axis that the robot needs to follow. its length is 24, standing for the 24 steps of a trajectory. its value range at each step is $[0.0, 2.2]$ m/s. Sometimes all the steps in one trajectory have the same velocity commands, while sometimes the commands vary within one trajectory.
- 2) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 3) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot should follow the forward velocity command as close as possible. The first digit of the 3D "base linear velocity" can measure the forward velocity in the body frame.
- 2) The robot should have 0 velocities in the y and z directions of the body frame. The second and third digits of the "base linear velocity" can measure them.
- 3) The robot should keep its body torso near the height of 0.34 meter. The "base height" value can measure the robot torso height.
- 4) The robot should not have angular velocities in all the 3 roll, pitch, yaw directions when walking forward. The 3 values of the "base angular velocity" should be close to 0.
- 5) The robot should not have roll or pitch angles when walking forward. Since the linear and angular velocities of the robot are randomly initialized at each episode, the robot might have some yaw angle from start, but this yaw angle should not change when the robot is walking forward.
- 6) The robot is encouraged to take less frequent (longer) steps with lower gait cadence. The "feet contacts" can be used to analyze the gait pattern of the robot. Each feature dimension (standing for

each foot) of the "feet contacts" tensor is encouraged to change between 0 and 1 less frequently in a trajectory.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.
- 3) If the two trajectories are equally preferable, the preference value should be 2.
- 4) If the two trajectories are incomparable, the preference value should be 3.

Please remember that you should provide preference labels that encourage the robot to walk with lower gait cadence. Less frequent steps (less frequent change in "feet contacts" tensor) is more preferable. Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Backflip

You are a robotics engineer trying to compare pairs of quadruped robot motion trajectories and decide which one is better in each pair.

Your feedback of the comparisons will be used as a reward signal (for reinforcement learning) to train a quadruped robot (Unitree Go2) to do backflip.

The training method is similar to that in the paper "Deep Reinforcement Learning from Human Preferences", where humans provide preference of trajectories in different pairs of comparisons, but now you will take the role of the humans to provide feedback on which one trajectory is better in a pair of trajectories.

Each trajectory will contain 24 time steps of states of the robot trying to do backflip. Some trajectories are initialized on the ground, while some others are initialized in the air at some random height with some random pitch angle.

The state includes:

- 1) "base linear velocity": the x, y, z positional velocities (m/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and x, y, z 3 dimensional velocities.
- 2) "base angular velocity": the roll, pitch, yaw angular velocities (rad/s) of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 angular velocities around the x, y, z axes.
- 4) "base height": the z position (height) of the robot base torso. The data shape is (24,), standing for the 24 steps of a trajectory.
- 5) "base roll pitch yaw": the roll, pitch, yaw radian angles of the robot base torso. The data shape is (24, 3), standing for 24 steps, and roll, pitch, yaw 3 rotation angles around the x, y, z axes.
- 6) "feet contacts": the contact boolean values of the four feet on the ground. 1 means touching the ground while 0 means in the air. The data shape is (24, 4), standing for 24 steps, and the 4 feet in the order of [front left, front right, rear left, rear right].

To decide which trajectory is better in a pair, here are some criteria:

- 1) The robot is encouraged to rotated backward to do a backflip, so a negative pitch rate is good, and a positive pitch rate is bad. The second value of the "base angular velocity" is the pitch rate.

- 2) The pitch angle of the robot is encouraged to keep decreasing. Since the range of the pitch angle is $-\pi$ (-3.14) to π (3.14), when the robot rotates back across the $-\pi$ angle, its pitch angle will jump to positive around π and then keep decreasing, and this behavior is very preferable. The second value of the "base roll pitch yaw" is the pitch angle.
- 3) The robot should jump high to have more time to do backflip. The "base height" value can measure the robot torso height.
- 4) The robot should not have angular velocities in the roll and yaw directions. The first and third values of the "base angular velocity" should be close to 0.
- 5) The robot should not have roll angle. The first value of the "base roll pitch yaw" should be close to 0.
- 6) The robot should have 0 velocity in the y direction of the body frame. The second digit of the "base linear velocity" can measure them.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

- 1) If the trajectory 0 is better, the preference value should be 0.
- 2) If the trajectory 1 is better, the preference value should be 1.
- 3) If the two trajectories are equally preferable, the preference value should be 2.
- 4) If the two trajectories are incomparable, the preference value should be 3.

Please give response with only one list of 5 preference values, e.g., [1, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indication which one is better in a pair.

Please be careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please be very careful about providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Kettle

You are a robotics engineer trying to compare pairs of shadow hands manipulation trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train the following task: this environment involves two hands, a kettle, and a bucket, we need to hold the kettle with one hand (left hand in current setting) and the bucket with the other hand (right hand), and pour the water from the kettle into the bucket.

Each trajectory will contain 16 timesteps of states of the shadow hand. To be specific, the state are as below:

- 1) "kettle spout position": the x, y, z position of the kettle's spout. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensions.
- 2) "kettle handle position": the x, y, z position of the kettle's handle. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensions.
- 3) "bucket position": the x, y, z position of the bucket. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensions.
- 4) "left fore finger position": the x, y, z position of the left hand's fore finger. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensions.
- 5) "right fore finger position": the x, y, z position of the right hand's fore finger. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensions.
- 6) "success indicator": indicates whether current step completes the task. The length is 16, standing for 16 steps. 1 stands for True and 0 for False.

To decide which trajectory is better in a pair, here are some criteria (importance by rank):

- 1) The trajectory that succeeds is better.
 - 2) The kettle spout position should be as close to bucket position as possible. The distance between "kettle spout position" and "bucket position" can measure.
 - 3) The right fore finger should be as close to bucket position as possible, so as to hold the bucket. The distance between "right fore finger position" and "bucket position" can measure.
 - 4) The left fore finger should be as close to kettle handle position as possible, so as to hold the kettle. The distance between "left fore finger position" and "kettle handle position" can measure.
- The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).
- 1) If the trajectory 0 is better, the preference value should be 0.
 - 2) If the trajectory 1 is better, the preference value should be 1.
 - 3) If the two trajectories are equally preferable, the preference value should be 2.
 - 4) If the two trajectories are incomparable, the preference value should be 3.
- Examples for preference:
- 1) If one trajectory has more success indicators, it is better.
 - 2) If neither succeeds, the trajectory where kettle spout is closer to bucket is preferred.
 - 3) If similar distance, the trajectory where left fore finger is closer to bucket is preferred.
 - 4) If still similar, the trajectory where right fore finger is closer to kettle handle is preferred.
- Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.
- Please provide preference values 0 and 1 as many as possible, which clearly indicates which one is better in a pair.
- Please be very careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.
- Please avoid providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Hand Over

You are a robotics engineer trying to compare pairs of shadow hands manipulation trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train tossing an object (a ball in this case) from hand 0 to hand 1.

For your reference, the palm position of hand 0, the releasing hand, is [0.000, -0.290, 0.490]. And the palm position of hand 1, the catching hand, is [0.000, -0.640, 0.540].

Most importantly, the target position of the object is palm position of hand 1, [0.000, -0.640, 0.540].

Each trajectory will contain 16 timesteps of states of the shadow hand. To be specific, the state are as below:

- 1) "object position": the x, y, z position of the object. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensional position.
- 2) "object linear velocity": the x, y, z positional velocities (m/s) of the object. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensional velocities.
- 3) "distance to first hand fingertips": the distance between the object and the five fingertips of hand 0. The data shape is (16, 5), standing for 16 steps, and 5 fingertips.
- 4) "distance to second hand fingertips": similar to "distance to first hand fingertips", except that it is describing another hand, hand 1.
- 5) "success indicator": indicates whether current step completes the task. The length is 16, standing for 16 steps. 1 stands for True and 0 for False.

To decide which trajectory is better in a pair, here are some criteria:

- 1) The trajectory that succeeds is better.

2) The object (ball) should be as close to the target position as possible. The distance between "object position" and target position can measure. And the second and third digits of "object position" should matter the most.

3) The object should keep a distance from any fingertips for both hands. Being smaller than $\text{*threshold of } 0.03\text{*}$ is highly penalized.

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

1) If the trajectory 0 is better, the preference value should be 0.

2) If the trajectory 1 is better, the preference value should be 1.

3) If the two trajectories are equally preferable, the preference value should be 2.

4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

1) If one trajectory has more success indicators, it is better.

2) If neither succeeds, the trajectory where object position is closer to target position is preferred.

3) If both succeed, the trajectory with closer distances in y and z axes between object and target is preferred.

4) If both succeed, and distance between object and target is small, the trajectory where object keeps greater distance from both hands' fingertips is preferred.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indicates which one is better in a pair.

Please be very careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please avoid providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

Prompt for Swing Cup

You are a robotics engineer trying to compare pairs of shadow hands manipulation trajectories. Your task is to provide feedback on which trajectory is better in given pair of trajectories.

Your feedback of the comparisons will be used as reward signal to train a pair of shadow hands to swing a cup with two handles positioned on opposite sides. They are pushing the handles in a coordinated manner to achieve a 180-degree counter-clockwise rotation along the z-axis

Most importantly, the goal rotation of the cup is [0.0000, -0.0000, -1.5708].

Each trajectory will contain 16 timesteps of states of the shadow hand. To be specific, the state are as below:

1) "object linear velocity": the x, y, z positional velocities (m/s) of the object. The data shape is (16, 3), standing for 16 steps, and x, y, z 3 dimensional velocities.

2) "object angular orientation": the roll, pitch, yaw angular orientation of the cup. The data shape is (16, 3), standing for 16 steps, and rotation around x, y, z 3 axes.

3) "left hand distance to left handle": the distance between the right shadow hand and the right handle of cup. The length is 16, standing for 16 steps.

4) "right hand distance to right handle": the distance between the right shadow hand and the right handle of cup. The length is 16, standing for 16 steps.

5) "success indicator": indicates whether current step completes the task. The length is 16, standing for 16 steps. 1 stands for True and 0 for False.

To decide which trajectory is better in a pair, here are some criteria (importance by rank):

1) The trajectory that succeeds is better.

2) The object rotation should be as close to target rotation as possible. The "object angular orientation" can help measure.

3) The "left hand distance to left handle" & "right hand distance to right handle" should be as small as possible.

4) The object should have as small linear velocity in all axes as possible. The "object linear velocity" can measure

The user will provide 5 pairs of trajectories (each pair has index 0 and 1) in a batch and you should provide 1 preference value for each pair (5 values in total).

1) If the trajectory 0 is better, the preference value should be 0.

2) If the trajectory 1 is better, the preference value should be 1.

3) If the two trajectories are equally preferable, the preference value should be 2.

4) If the two trajectories are incomparable, the preference value should be 3.

Examples for preference:

1) If one trajectory has more success indicators, it is better.

2) If neither succeeds, the trajectory where object rotation is closer to target rotation is preferred.

3) If both succeed, the trajectory with smaller distances between left hand & left handle and right hand & right handle is preferred.

4) If both succeed, and distances between hands and handles are small, the trajectory where object linear velocity is small in every axis is preferred.

Please give response with only one list of 5 preference values, e.g., [0, 0, 1, 2, 3]. Do not provide any other text such as your comments or thoughts. The preference value number can only be 0, 1, 2, or 3.

Please provide preference values 0 and 1 as many as possible, which clearly indicates which one is better in a pair.

Please be very careful about providing equally preferable value 2. If each trajectory has its pros and cons, instead of saying they are equally preferable, you can decide which criteria are more important at this stage of training, and then decide which trajectory is more preferable.

Please avoid providing incomparable value 3! Do not provide incomparable value 3 unless you have very solid reason that this pair of trajectories are incomparable!

A.2 Full Rewards

In this section, we provide all the explicit environment rewards for training with LAPP. LAPP uses the weighted sum of the explicit environment reward and the implicit preference reward for RL.

Reward for Flat Plane locomotion and bounding control and cadence control

```
def compute_reward(self):
    """ Compute rewards
        Compute each reward component first
        Then compute the total reward
        Return the total reward, and the recording of all reward components
    """
    env = self.env # Do not skip this line. Afterwards, use env.{parameter_name}
    ↪ to access parameters of the environment.

    # Tracking of linear velocity commands (xy axes)
    lin_vel_error = torch.sum(torch.square(env.commands[:, :2] -
    ↪ env.base_lin_vel[:, :2]), dim=1)
    tracking_lin_vel_reward = 1.0 * torch.exp(-lin_vel_error / 0.25)

    # Tracking of angular velocity commands (yaw)
```

```

ang_vel_error = torch.square(env.commands[:, 2] - env.base_ang_vel[:, 2])
tracking_ang_vel_reward = 0.5 * torch.exp(-ang_vel_error / 0.25)

# Penalize z axis base linear velocity
lin_vel_z_reward = -2.0 * torch.square(env.base_lin_vel[:, 2])

# Penalize xy axes base angular velocity
ang_vel_xy_reward = -0.05 * torch.sum(torch.square(env.base_ang_vel[:, :2]),
    ↪ dim=1)

# Penalize torques
torques_reward = -0.0002 * torch.sum(torch.square(env.torques), dim=1)

# Penalize dof accelerations
dof_acc_reward = -2.5e-7 * torch.sum(torch.square((env.last_dof_vel -
    ↪ env.dof_vel) / env.dt), dim=1)

# Reward long steps
# Need to filter the contacts because the contact reporting of PhysX is
    ↪ unreliable on meshes
contact = env.contact_forces[:, env.feet_indices, 2] > 1.
contact_filt = torch.logical_or(contact, env.last_contacts)
env.last_contacts = contact
first_contact = (env.feet_air_time > 0.) * contact_filt
env.feet_air_time += env.dt
rew_airTime = torch.sum((env.feet_air_time - 0.5) * first_contact, dim=1) #
    ↪ reward only on first contact with the ground
rew_airTime *= torch.norm(env.commands[:, :2], dim=1) > 0.1 # no reward for
    ↪ zero command
env.feet_air_time *= ~contact_filt
feet_air_time_reward = 1.0 * rew_airTime

# Penalize collisions on selected bodies
collision_reward = -1.0 * torch.sum(1. * (torch.norm(env.contact_forces[:,
    ↪ env.penalised_contact_indices, :], dim=-1) > 0.1), dim=1)

# Penalize changes in actions
action_rate_reward = -0.01 * torch.sum(torch.square(env.last_actions -
    ↪ env.actions), dim=1)

# Penalize dof positions too close to the limit
out_of_limits = -(env.dof_pos - env.dof_pos_limits[:, 0]).clip(max=0.) # lower
    ↪ limit
out_of_limits += (env.dof_pos - env.dof_pos_limits[:, 1]).clip(min=0.)
dof_pos_limits_reward = -10.0 * torch.sum(out_of_limits, dim=1)

# # Penalize base height away from target
# target_height_z = 0.34 # Ideal height of the robot's torso
# base_height = env.root_states[:, 2]
# height_reward = -0.05 * torch.square(base_height - target_height_z) # reward
    ↪ to maintain height

```

```

# Height reward component
target_height_z = 0.34 # Ideal height of the robot's torso
base_height = env.root_states[:, 2]
height_error = torch.abs(base_height - target_height_z)
temperature_height = 5.0 # Temperature parameter for the height reward
height_reward = 1.0 * torch.exp(-temperature_height * height_error) # More
↳ weight to maintain height

# Combine reward components to compute the total reward in this step
total_reward = (tracking_lin_vel_reward + tracking_ang_vel_reward +
↳ lin_vel_z_reward +
    ang_vel_xy_reward + torques_reward + dof_acc_reward +
    ↳ feet_air_time_reward +
    collision_reward + action_rate_reward + dof_pos_limits_reward +
    ↳ height_reward)

# # Normalizing the total reward to avoid exploding values
# total_reward = total_reward / (1 + torch.abs(total_reward)) # Additional
↳ normalization for stability

# Debug information
reward_components = {"tracking_lin_vel_reward": tracking_lin_vel_reward,
    "tracking_ang_vel_reward": tracking_ang_vel_reward,
    "lin_vel_z_reward": lin_vel_z_reward,
    "ang_vel_xy_reward": ang_vel_xy_reward,
    "torques_reward": torques_reward,
    "dof_acc_reward": dof_acc_reward,
    "feet_air_time_reward": feet_air_time_reward,
    "collision_reward": collision_reward,
    "action_rate_reward": action_rate_reward,
    "dof_pos_limits_reward": dof_pos_limits_reward,
    "height_reward": height_reward}
return total_reward, reward_components

```

Reward for Stairs

```

def compute_reward(self):
    """ Compute improved rewards
        Compute each reward component first
        Then compute the total reward
        Return the total reward, and the recording of all reward components
    """
    env = self.env # Do not skip this line. Afterwards, use env.{parameter_name}
    ↳ to access parameters of the environment.

    # Tracking of linear velocity commands (xy axes)
    lin_vel_error = torch.sum(torch.square(env.commands[:, :2] -
    ↳ env.base_lin_vel[:, :2]), dim=1)
    tracking_lin_vel_reward = 1.5 * torch.exp(-lin_vel_error / 0.20)

    # Tracking of angular velocity commands (yaw)

```

```

ang_vel_error = torch.square(env.commands[:, 2] - env.base_ang_vel[:, 2])
tracking_ang_vel_reward = 0.5 * torch.exp(-ang_vel_error / 0.1)

# # Penalize z axis base linear velocity
lin_vel_z_reward = -0.00001 * torch.square(env.base_lin_vel[:, 2])

# Penalize xy axes base angular velocity
ang_vel_xy_reward = -0.1 * torch.sum(torch.square(env.base_ang_vel[:, :2]),
    ↪ dim=1)

# Penalize torques
torques_reward = -0.0005 * torch.sum(torch.square(env.torques), dim=1)

# Penalize dof accelerations
dof_acc_reward = -1.0e-7 * torch.sum(torch.square((env.last_dof_vel -
    ↪ env.dof_vel) / env.dt), dim=1)

# Reward air time
contact = env.contact_forces[:, env.feet_indices, 2] > 1.
contact_filt = torch.logical_or(contact, env.last_contacts)
env.last_contacts = contact
first_contact = (env.feet_air_time > 0.) * contact_filt
env.feet_air_time += env.dt
rew_airTime = torch.sum((env.feet_air_time - 0.5) * first_contact, dim=1)
rew_airTime *= torch.norm(env.commands[:, :2], dim=1) > 0.1
env.feet_air_time *= ~contact_filt
feet_air_time_reward = 0.8 * rew_airTime

# Penalize collisions
collision_reward = -5.0 * torch.sum(1. * (torch.norm(env.contact_forces[:,
    ↪ env.penalised_contact_indices, :], dim=-1) > 0.1), dim=1)

# Penalize changes in actions
action_rate_reward = -0.008 * torch.sum(torch.square(env.last_actions -
    ↪ env.actions), dim=1)

# Penalize dofs close to limits
out_of_limits = -(env.dof_pos - env.dof_pos_limits[:, 0]).clip(max=0.)
out_of_limits += (env.dof_pos - env.dof_pos_limits[:, 1]).clip(min=0.)
dof_pos_limits_reward = -7.0 * torch.sum(out_of_limits, dim=1)

# Penalize base height away from target
target_height_z = 0.34
base_height = env.root_states[:, 2]
# get the ground height of the terrain
ground_x = env.root_states[:, 0]
ground_y = env.root_states[:, 1]
ground_z = env._get_stairs_terrain_heights(ground_x, ground_y)
# calculate the base-to-ground height
base2ground_height = base_height - ground_z
height_reward = -0.000002 * torch.square(base2ground_height - target_height_z)

```

```

# stumbling penalty
stumble = (torch.norm(env.contact_forces[:, env.feet_indices, :2], dim=2) > 5.)
↪ * (torch.abs(env.contact_forces[:, env.feet_indices, 2]) < 1.)
stumble_reward = -2.0 * torch.sum(stumble, dim=1)

# Combine reward components to compute the total reward in this step
total_reward = (tracking_lin_vel_reward + tracking_ang_vel_reward +
↪ lin_vel_z_reward +
               ang_vel_xy_reward + torques_reward + dof_acc_reward +
               ↪ feet_air_time_reward +
               collision_reward + action_rate_reward + dof_pos_limits_reward +
               ↪ height_reward + stumble_reward)

# Debug information
reward_components = {"tracking_lin_vel_reward": tracking_lin_vel_reward,
                    "tracking_ang_vel_reward": tracking_ang_vel_reward,
                    "lin_vel_z_reward": lin_vel_z_reward,
                    "ang_vel_xy_reward": ang_vel_xy_reward,
                    "torques_reward": torques_reward,
                    "dof_acc_reward": dof_acc_reward,
                    "feet_air_time_reward": feet_air_time_reward,
                    "collision_reward": collision_reward,
                    "action_rate_reward": action_rate_reward,
                    "dof_pos_limits_reward": dof_pos_limits_reward,
                    "height_reward": height_reward,
                    "stumble_reward": stumble_reward}

return total_reward, reward_components

```

Reward for Obstacles

```

def compute_reward(self):
    """ Compute rewards for wave terrain """
    env = self.env

    # 1. Linear velocity tracking along x-axis
    lin_vel_error = torch.sum(torch.square(env.commands[:, :2] -
↪ env.base_lin_vel[:, :2]), dim=1)
    tracking_lin_vel_reward = 2.3 * torch.exp(-lin_vel_error / 0.20)

    # 2. Tracking of angular velocity commands (yaw)
    ang_vel_error = torch.square(env.commands[:, 2] - env.base_ang_vel[:, 2])
    tracking_ang_vel_reward = 0.8 * torch.exp(-ang_vel_error / 0.1)

    # 3. Penalize z-axis velocity
    # lin_vel_z_reward = -0.001 * torch.square(env.base_lin_vel[:, 2])
    ang_vel_x_reward = -0.002 * torch.square(env.base_ang_vel[:, 0])

    # 4. Base height tracking (adjusted for wave terrain)
    target_height_z = 0.34

```

```

base_height = env.root_states[:, 2]
ground_x = env.root_states[:, 0]
ground_y = env.root_states[:, 1]
ground_z = env._get_terrain_heights(ground_x, ground_y)
base2ground_height = base_height - ground_z
height_reward = -1.5 * torch.square(base2ground_height - target_height_z)

# 5. Penalize torques
torques_reward = -0.00001 * torch.sum(torch.square(env.torques), dim=1)

# 6. Penalize changes in actions
action_rate_reward = -0.0055 * torch.sum(torch.square(env.last_actions -
↪ env.actions), dim=1)

# 7. Encourage smoother joint motions (penalize excessive joint accelerations)
dof_acc_penalty = -1e-8 * torch.sum(torch.square((env.dof_vel -
↪ env.last_dof_vel) / env.dt), dim=1)

# 8. Air time reward for dynamic gaits
contact = env.contact_forces[:, env.feet_indices, 2] > 1.0
contact_filt = torch.logical_or(contact, env.last_contacts)
env.last_contacts = contact
first_contact = (env.feet_air_time > 0.0) * contact_filt
env.feet_air_time += env.dt
rew_airTime = torch.sum((env.feet_air_time - 0.4) * first_contact, dim=1)
rew_airTime *= torch.norm(env.commands[:, :2], dim=1) > 0.1
env.feet_air_time *= ~contact_filt
air_time_reward = 0.5 * rew_airTime

# 9. Collision penalty (avoid collisions with terrain or robot parts)
collision_penalty = -1. * torch.sum(
    1.0 * (torch.norm(env.contact_forces[:, env.penalised_contact_indices, :],
↪ dim=-1) > 0.13),
    dim=1
)

# 10. Gait pattern reward (encourage trot gait using phase alignment)
diag_sync = (contact[:, 0] == contact[:, 3]) & (contact[:, 1] == contact[:, 2])
gait_pattern_reward = 0.0001 * torch.sum(diag_sync.float())

# 11. Penalize use only two feet
stumble = (torch.norm(env.contact_forces[:, env.feet_indices, :2], dim=2) > 5.)
↪ * (torch.abs(env.contact_forces[:, env.feet_indices, 2]) < 1.)
stumble_penalty = -20.0 * torch.sum(stumble, dim=1)

# Combine all components into the total reward
total_reward = (
    tracking_lin_vel_reward +
    tracking_ang_vel_reward +
    ang_vel_x_reward +
    height_reward +
    torques_reward +

```

```

        action_rate_reward +
        dof_acc_penalty +
        air_time_reward +
        collision_penalty +
        gait_pattern_reward +
        stumble_penalty
    )

    # Debug information for reward components
    reward_components = {
        "tracking_lin_vel_reward": tracking_lin_vel_reward,
        "tracking_ang_vel_reward": tracking_ang_vel_reward,
        "ang_vel_x_reward": ang_vel_x_reward,
        "height_reward": height_reward,
        "torques_reward": torques_reward,
        "action_rate_reward": action_rate_reward,
        "dof_acc_penalty": dof_acc_penalty,
        "air_time_reward": air_time_reward,
        "collision_penalty": collision_penalty,
        "gait_pattern_reward": gait_pattern_reward,
        "stumble_penalty": stumble_penalty
    }

    return total_reward, reward_components

```

Reward for Wave

```

def compute_reward(self):
    """ Compute rewards for wave terrain """
    env = self.env

    # 1. Tracking of linear velocity commands (xy axes)
    lin_vel_error = torch.sum(torch.square(env.commands[:, :2] -
        ↪ env.base_lin_vel[:, :2]), dim=1)
    tracking_lin_vel_reward = 2.3 * torch.exp(-lin_vel_error / 0.20)

    # 2. Tracking of angular velocity commands (yaw)
    ang_vel_error = torch.square(env.commands[:, 2] - env.base_ang_vel[:, 2])
    tracking_ang_vel_reward = 0.8 * torch.exp(-ang_vel_error / 0.1)

    # 3. Base height tracking (adjusted for wave terrain)
    target_height_z = 0.34
    base_height = env.root_states[:, 2]
    ground_x = env.root_states[:, 0]
    ground_y = env.root_states[:, 1]
    ground_z = env._get_terrain_heights(ground_x, ground_y)
    base2ground_height = base_height - ground_z
    height_reward = -1.5 * torch.square(base2ground_height - target_height_z)

    # 4. Penalize torques
    torques_reward = -0.00001 * torch.sum(torch.square(env.torques), dim=1)

```

```

# 5. Penalize changes in actions
action_rate_reward = -0.0045 * torch.sum(torch.square(env.last_actions -
↳ env.actions), dim=1)

# 6. Encourage smoother joint motions (penalize excessive joint accelerations)
dof_acc_penalty = -1e-8 * torch.sum(torch.square((env.dof_vel -
↳ env.last_dof_vel) / env.dt), dim=1)

# 7. Air time reward for dynamic gaits
contact = env.contact_forces[:, env.feet_indices, 2] > 1.0
contact_filt = torch.logical_or(contact, env.last_contacts)
env.last_contacts = contact
first_contact = (env.feet_air_time > 0.0) * contact_filt
env.feet_air_time += env.dt
rew_airTime = torch.sum((env.feet_air_time - 0.4) * first_contact, dim=1)
rew_airTime *= torch.norm(env.commands[:, :2], dim=1) > 0.1
env.feet_air_time *= ~contact_filt
air_time_reward = 0.5 * rew_airTime

# 8. Collision penalty (avoid collisions with terrain or robot parts)
collision_penalty = -1. * torch.sum(
    1.0 * (torch.norm(env.contact_forces[:, env.penalised_contact_indices, :],
↳ dim=-1) > 0.13),
    dim=1
)

# 9. Gait pattern reward (encourage trot gait using phase alignment)
diag_sync = (contact[:, 0] == contact[:, 3]) & (contact[:, 1] == contact[:, 2])
gait_pattern_reward = 0.0001 * torch.sum(diag_sync.float())

# 10. Penalize use only two feet
lack_of_foot_usage = (~contact).float().sum(dim=1)
lack_of_foot_usage_penalty = -0.01 * lack_of_foot_usage

# Combine all components into the total reward
total_reward = (
    tracking_lin_vel_reward +
    tracking_ang_vel_reward +
    height_reward +
    torques_reward +
    action_rate_reward +
    dof_acc_penalty +
    air_time_reward +
    collision_penalty +
    gait_pattern_reward +
    lack_of_foot_usage_penalty
)

# Debug information for reward components
reward_components = {
    "tracking_lin_vel_reward": tracking_lin_vel_reward,

```



```

    "tracking_ang_vel_reward": tracking_ang_vel_reward,
    "height_reward": height_reward,
    "torques_reward": torques_reward,
    "action_rate_reward": action_rate_reward,
    "dof_acc_penalty": dof_acc_penalty,
    "air_time_reward": air_time_reward,
    "collision_penalty": collision_penalty,
    "gait_pattern_reward": gait_pattern_reward,
    "lack_of_foot_usage_penalty": lack_of_foot_usage_penalty
}

return total_reward, reward_components

```

Reward for Slope

```

def compute_reward(self):
    """ Compute improved rewards
        Compute each reward component first
        Then compute the total reward
        Return the total reward, and the recording of all reward components
    """
    env = self.env # Do not skip this line. Afterwards, use
    ↪ env.{parameter_name} to access parameters of the environment.

    # Tracking of linear velocity commands (xy axes)
    lin_vel_error = torch.sum(torch.square(env.commands[:, :2] -
    ↪ env.base_lin_vel[:, :2]), dim=1)
    tracking_lin_vel_reward = 3.0 * torch.exp(-lin_vel_error / 0.10)

    # Tracking of angular velocity commands (yaw)
    ang_vel_error = torch.square(env.commands[:, 2] - env.base_ang_vel[:, 2])
    tracking_ang_vel_reward = 1.0 * torch.exp(-ang_vel_error / 0.05)

    # Penalize z axis base linear velocity
    lin_vel_z_reward = -0.00001 * torch.square(env.base_lin_vel[:, 2])

    # Penalize xy axes base angular velocity
    ang_vel_xy_reward = -0.1 * torch.sum(torch.square(env.base_ang_vel[:, :2]),
    ↪ dim=1)

    # Penalize torques
    torques_reward = -0.0001 * torch.sum(torch.square(env.torques), dim=1)

    # Penalize dof accelerations
    dof_acc_reward = -5.0e-8 * torch.sum(torch.square((env.last_dof_vel -
    ↪ env.dof_vel) / env.dt), dim=1)

    # Reward air time
    contact = env.contact_forces[:, env.feet_indices, 2] > 1.
    contact_filt = torch.logical_or(contact, env.last_contacts)
    env.last_contacts = contact
    first_contact = (env.feet_air_time > 0.) * contact_filt

```

```

env.feet_air_time += env.dt
rew_airTime = torch.sum((env.feet_air_time - 0.5) * first_contact, dim=1)
rew_airTime *= torch.norm(env.commands[:, :2], dim=1) > 0.1
env.feet_air_time *= ~contact_filt
feet_air_time_reward = 0.6 * rew_airTime

# Penalize collisions
collision_reward = -15.0 * torch.sum(1. * (torch.norm(env.contact_forces[:,
↪ env.penalised_contact_indices, :], dim=-1) > 0.1), dim=1)

# Penalize changes in actions
action_rate_reward = -0.015 * torch.sum(torch.square(env.last_actions -
↪ env.actions), dim=1)

# Penalize dofs close to limits
out_of_limits = -(env.dof_pos - env.dof_pos_limits[:, 0]).clip(max=0.)
out_of_limits += (env.dof_pos - env.dof_pos_limits[:, 1]).clip(min=0.)
dof_pos_limits_reward = -6.0 * torch.sum(out_of_limits, dim=1)

# Penalize base height away from target
target_height_z = 0.34
base_height = env.root_states[:, 2]
# get the ground height of the terrain
ground_x = env.root_states[:, 0]
ground_y = env.root_states[:, 1]
ground_z = env._get_terrain_heights(ground_x, ground_y)
# calculate the base-to-ground height
base2ground_height = base_height - ground_z
height_reward = -0.000001 * torch.square(base2ground_height -
↪ target_height_z)

# stumbling penalty
# stumble = (torch.norm(env.contact_forces[:, env.feet_indices, :2], dim=2)
↪ > 5.) * (torch.abs(env.contact_forces[:, env.feet_indices, 2]) < 1.)
# stumble_reward = -4.0 * torch.sum(stumble, dim=1)

# Combine reward components to compute the total reward in this step
total_reward = (tracking_lin_vel_reward + tracking_ang_vel_reward +
↪ lin_vel_z_reward +
               ang_vel_xy_reward + torques_reward + dof_acc_reward +
               ↪ feet_air_time_reward +
               collision_reward + action_rate_reward +
               ↪ dof_pos_limits_reward + height_reward)
               # + stumble_reward)

# Debug information
reward_components = {"tracking_lin_vel_reward": tracking_lin_vel_reward,
                    "tracking_ang_vel_reward": tracking_ang_vel_reward,
                    "lin_vel_z_reward": lin_vel_z_reward,
                    "ang_vel_xy_reward": ang_vel_xy_reward,
                    "torques_reward": torques_reward,
                    "dof_acc_reward": dof_acc_reward,

```

```

        "feet_air_time_reward": feet_air_time_reward,
        "collision_reward": collision_reward,
        "action_rate_reward": action_rate_reward,
        "dof_pos_limits_reward": dof_pos_limits_reward,
        "height_reward": height_reward,}

    return total_reward, reward_components

```

Reward for Jump high

```

def compute_reward(self):
    """ Compute rewards for the backflip task """
    env = self.env # Do not skip this line. Afterwards, use env.{parameter_name}
    ↪ to access parameters of the environment.

    # Penalize angular velocity around x-axis (roll) and z-axis (yaw)
    roll_rate = env.root_states[:, 10] # Angular velocity around x-axis (roll)
    yaw_rate = env.root_states[:, 12] # Angular velocity around z-axis (yaw)
    roll_rate_reward = -0.05 * torch.square(roll_rate) # Penalize deviation in
    ↪ roll, org -5.0, 0.5
    yaw_rate_reward = -0.2 * torch.square(yaw_rate) # Penalize deviation in yaw,
    ↪ org -5.0, -2.0

    # Penalize roll and yaw around x-axis (roll) and z-axis (yaw)
    roll = env.rpy[:, 0]
    pitch = env.rpy[:, 1]
    yaw = env.rpy[:, 2]
    roll_reward = -0.2 * torch.square(roll) # Penalize deviation in roll
    pitch_reward = -0.2 * torch.square(pitch) # Penalize deviation in pitch
    yaw_reward = -1.0 * torch.square(yaw) # Penalize deviation in yaw

    # Encourage z axis base linear velocity
    lin_vel_z_reward = 0.4 * torch.square(env.root_states[:, 9])

    # Penalize x axis base linear velocity forward
    lin_vel_x_reward = -0.4 * torch.square(torch.relu(env.root_states[:, 7]))

    # Penalize torques
    torques_reward = -0.0005 * torch.sum(torch.square(env.torques), dim=1)

    # Penalize dof accelerations
    dof_acc_reward = -1.0e-7 * torch.sum(torch.square((env.last_dof_vel -
    ↪ env.dof_vel) / env.dt), dim=1)

    # # Penalize dofs close to limits
    # out_of_limits = -(env.dof_pos - env.dof_pos_limits[:, 0]).clip(max=0.)
    # out_of_limits += (env.dof_pos - env.dof_pos_limits[:, 1]).clip(min=0.)
    # dof_pos_limits_reward = -7.0 * torch.sum(out_of_limits, dim=1)

    # Penalize base torso hitting the ground

```

```

base_hit_ground = torch.any(torch.norm(env.contact_forces[:,
↳ env.termination_contact_indices, :], dim=-1) > 0.1, dim=1) # >1.0
↳ originally
base_hit_ground_reward = -20.0 * base_hit_ground # 10

# Penalize non flat base orientation
gravity_reward = -0.01 * torch.sum(torch.square(env.projected_gravity[:, :2]),
↳ dim=1)

# Reward for highest z position reached in an episode
height_history = env.height_history_buf
height_history_reward = 1.0 * height_history # 0.5

# Combine reward components to compute the total reward in this step
total_reward = (roll_rate_reward + yaw_rate_reward + roll_reward + pitch_reward
↳ + yaw_reward + lin_vel_z_reward + lin_vel_x_reward +
torques_reward + dof_acc_reward+ base_hit_ground_reward +
↳ gravity_reward + height_history_reward)

# Debug information
reward_components = {
    "roll_rate_reward": roll_rate_reward,
    "yaw_rate_reward": yaw_rate_reward,
    "roll_reward": roll_reward,
    "pitch_reward": pitch_reward,
    "yaw_reward": yaw_reward,
    "lin_vel_z_reward": lin_vel_z_reward,
    "lin_vel_x_reward": lin_vel_x_reward,
    "torques_reward": torques_reward,
    "dof_acc_reward": dof_acc_reward,
    "base_hit_ground_reward": base_hit_ground_reward,
    "gravity_reward": gravity_reward,
    "height_history_reward": height_history_reward
}
return total_reward, reward_components

```

Reward for Backflip

```

def compute_reward(self):
    """ Compute rewards for the backflip task """
    env = self.env # Do not skip this line. Afterwards, use env.{parameter_name}
    ↳ to access parameters of the environment.

    # Penalize angular velocity around x-axis (roll) and z-axis (yaw)
    roll_rate = env.root_states[:, 10] # Angular velocity around x-axis (roll)
    yaw_rate = env.root_states[:, 12] # Angular velocity around z-axis (yaw)
    roll_rate_reward = -0.05 * torch.square(roll_rate) # Penalize deviation in
    ↳ roll, org -5.0, 0.5
    yaw_rate_reward = -0.2 * torch.square(yaw_rate) # Penalize deviation in yaw,
    ↳ org -5.0, -2.0

```

```

# Penalize roll and yaw around x-axis (roll) and z-axis (yaw)
roll = env.rpy[:, 0]
yaw = env.rpy[:, 2]
roll_reward = -0.2 * torch.square(roll) # Penalize deviation in roll
yaw_reward = -1.0 * torch.square(yaw) # Penalize deviation in yaw

# Reward for angular velocity around y-axis (pitch), encourage backflip
↪ rotation
pitch_rate = env.base_ang_vel[:, 1] # Angular velocity around y-axis (pitch)
# Clamp the pitch rate to a minimum of -7 rad/s (maximum negative rotation
↪ speed)
clamped_pitch_rate = torch.clamp(pitch_rate, max=5.0, min=-7.0)
# Compute the backflip reward
pitch_rate_reward = torch.where(
    clamped_pitch_rate < 0,
    1.0 * (-clamped_pitch_rate), # Positive reward for negative pitch rate
    -0.1 * clamped_pitch_rate # Negative penalty for positive pitch rate
)

# pitch_rate_reward = 1.0 * (-clamped_pitch_rate) # Reward negative pitch
↪ rates

# Reward for pitch frontwards for the back flip
pitch = env.rpy[:, 1]
last_pitch = env.last_rpy[:, 1]
delta_pitch = (pitch - last_pitch + torch.pi) % (2 * torch.pi) - torch.pi
delta_pitch_reward = torch.where(delta_pitch > 0, delta_pitch,
    ↪ torch.zeros_like(delta_pitch))
delta_pitch_reward = 20.0 * delta_pitch_reward

# Encourage z axis base linear velocity
lin_vel_z_reward = 0.2 * torch.square(env.root_states[:, 9])

# Penalize torques
torques_reward = -0.0005 * torch.sum(torch.square(env.torques), dim=1)

# Penalize dof accelerations
dof_acc_reward = -1.0e-7 * torch.sum(torch.square((env.last_dof_vel -
    ↪ env.dof_vel) / env.dt), dim=1)

# # Penalize dofs close to limits
# out_of_limits = -(env.dof_pos - env.dof_pos_limits[:, 0]).clip(max=0.)
# out_of_limits += (env.dof_pos - env.dof_pos_limits[:, 1]).clip(min=0.)
# dof_pos_limits_reward = -7.0 * torch.sum(out_of_limits, dim=1)

# Penalize base torso hitting the ground
base_hit_ground = torch.any(torch.norm(env.contact_forces[:,
    ↪ env.termination_contact_indices, :], dim=-1) > 0.1, dim=1) # >1.0
↪ originally
base_hit_ground_reward = -20.0 * base_hit_ground # 10

# Penalize projected gravity along y axis

```

```

gravity_reward = -0.01 * torch.square(env.projected_gravity[:, 1])

# Encourage backwards rotation of the projected gravity
delta_gravity_x = env.projected_gravity[:, 0] - env.last_projected_gravity[:,
↪ 0]
condition = env.projected_gravity[:, 2] < 0 # Corrected to use the z-axis as
↪ per your description
desired_direction = torch.where(
    condition,
    -1.0, # Encourage decrease in delta_gravity_x
    1.0 # Encourage increase in delta_gravity_x
)

# Compute the alignment between desired and actual change
alignment = desired_direction * delta_gravity_x

# Compute the reward with different scaling factors
rotate_gravity_reward = torch.where(
    alignment > 0,
    20.0 * torch.abs(delta_gravity_x), # Positive reward
    -2.0 * torch.abs(delta_gravity_x) # Negative penalty
)

# Reward for highest z position reached in an episode
height_history = env.height_history_buf
height_history_reward = 0.8 * height_history # 0.5

# Combine reward components to compute the total reward in this step
total_reward = (roll_rate_reward + yaw_rate_reward + roll_reward + yaw_reward +
↪ pitch_rate_reward +
    delta_pitch_reward + lin_vel_z_reward + torques_reward +
↪ dof_acc_reward+
    base_hit_ground_reward + gravity_reward + rotate_gravity_reward
↪ + height_history_reward)

# Debug information
reward_components = {
    "roll_rate_reward": roll_rate_reward,
    "yaw_rate_reward": yaw_rate_reward,
    "roll_reward": roll_reward,
    "yaw_reward": yaw_reward,
    "pitch_rate_reward": pitch_rate_reward,
    "delta_pitch_reward": delta_pitch_reward,
    "lin_vel_z_reward": lin_vel_z_reward,
    "torques_reward": torques_reward,
    "dof_acc_reward": dof_acc_reward,
    "base_hit_ground_reward": base_hit_ground_reward,
    "gravity_reward": gravity_reward,
    "rotate_gravity_reward": rotate_gravity_reward,
    "height_history_reward": height_history_reward
}

```

```
return total_reward, reward_components
```

Reward for Kettle

```
def kettle_compute_reward(kettle_spout_pos: torch.Tensor,
                          bucket_handle_pos: torch.Tensor,
                          left_hand_pos: torch.Tensor,
                          right_hand_pos: torch.Tensor,
                          left_hand_ff_pos: torch.Tensor,
                          right_hand_ff_pos: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:

    # Redefine proximity reward for less dominance
    kettle_to_bucket_distance = torch.norm(kettle_spout_pos - bucket_handle_pos,
        ↪ dim=1)
    temp_proximity = 1.0
    proximity_reward = torch.exp(-kettle_to_bucket_distance / temp_proximity)

    # Rescale grip rewards and refine detection logic
    left_grip_distance = torch.norm(left_hand_ff_pos - kettle_spout_pos, dim=1)
    right_grip_distance = torch.norm(right_hand_ff_pos - bucket_handle_pos, dim=1)

    temp_grip = 0.5
    left_grip_reward = torch.exp(-left_grip_distance / temp_grip)
    right_grip_reward = torch.exp(-right_grip_distance / temp_grip)

    # Improved task-specific reward detection logic
    task_success_indicator = torch.tensor([0], device=kettle_spout_pos.device) #
    ↪ Replace with task success condition
    task_completion_threshold = 0.1 # Define a threshold to reflect task
    ↪ completion
    task_complete = kettle_to_bucket_distance < task_completion_threshold
    transformed_task_score = task_complete.float()

    # Total reward balancing individual components
    total_reward = (
        0.2 * proximity_reward +
        0.3 * left_grip_reward +
        0.3 * right_grip_reward +
        1.0 * transformed_task_score
    )

    # Reward components provided for diagnosis
    reward_components = {
        "kettle_spout_proximity": proximity_reward,
        "kettle_handle_grip": left_grip_reward,
        "bucket_handle_grip": right_grip_reward,
        "task_success": transformed_task_score
    }
```

```
return total_reward, reward_components
```

Reward for Hand Over

```
def hand_over_compute_reward(object_pos: torch.Tensor,
                             object_rot: torch.Tensor,
                             object_linvel: torch.Tensor,
                             goal_pos: torch.Tensor,
                             goal_rot: torch.Tensor,
                             fingertip_pos: torch.Tensor,
                             fingertip_another_pos: torch.Tensor
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    # Constants
    distance_threshold: float = 0.03
    rotation_threshold: float = 0.1
    catch_reward_weight: float = 30.0
    toss_reward_weight: float = 10.0
    toss_temperature: float = 1.5
    catch_temperature: float = 0.5
    penalty_weight: float = 10.0

    # Compute distance and rotation differences between object and goal
    object_goal_distance = torch.norm(object_pos - goal_pos, dim=-1)
    object_goal_rotation_diff = torch.norm(object_rot - goal_rot, dim=-1)

    # Reward for object being close to the goal position and rotation
    toss_reward = torch.exp(-toss_reward_weight * (object_goal_distance /
    ↪ toss_temperature))

    # Calculate the catch reward based on the difference between the object's
    ↪ linear velocity and the goal position
    catch_reward = torch.norm(goal_pos - object_linvel, dim=-1)
    catch_reward = torch.exp(-catch_reward_weight * (catch_reward /
    ↪ catch_temperature))

    # Penalty for direct rolling or touching the target instead of tossing and
    ↪ catching
    penalty = torch.zeros_like(object_goal_distance)
    for i in range(fingertip_pos.shape[1]):
        dist_to_fingertip = torch.norm(object_pos - fingertip_pos[:, i, :], dim=-1)
        dist_to_fingertip_another = torch.norm(object_pos -
        ↪ fingertip_another_pos[:, i, :], dim=-1)
        penalty = torch.where(
            (dist_to_fingertip < distance_threshold) | (dist_to_fingertip_another <
            ↪ distance_threshold),
            penalty + 1.0, penalty
        )

    penalty_ratio = torch.sigmoid(penalty * penalty_weight) * 0.5
    penalty = catch_reward * penalty_ratio # Adapt the penalty to the
    ↪ catch_reward's dynamic range
```



```

# Total reward
total_reward = toss_reward + catch_reward - penalty
reward_terms = {"toss_reward": toss_reward, "catch_reward": catch_reward,
    ↪ "penalty": penalty}

return total_reward, reward_terms

```

Reward for Swing Cup

```

def swing_cup_compute_reward(object_pos: torch.Tensor,
                             object_rot: torch.Tensor,
                             object_linvel: torch.Tensor,
                             cup_right_handle_pos: torch.Tensor,
                             cup_left_handle_pos: torch.Tensor,
                             left_hand_pos: torch.Tensor,
                             right_hand_pos: torch.Tensor,
                             goal_pos: torch.Tensor,
                             goal_rot: torch.Tensor
    ) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    object_goal_distance = torch.norm(object_pos - goal_pos, dim=-1)
    distance_reward_temperature = 0.1
    object_goal_distance_reward_weight = 0.1
    object_goal_distance_reward = -torch.exp(-distance_reward_temperature *
    ↪ object_goal_distance) * object_goal_distance_reward_weight

    right_cup_handle_dist = torch.norm(cup_right_handle_pos - right_hand_pos,
    ↪ dim=-1)
    left_cup_handle_dist = torch.norm(cup_left_handle_pos - left_hand_pos, dim=-1)

    cup_orientation_diff = 1 - torch.sum(torch.mul(object_rot, goal_rot), dim=-1)
    ↪ ** 2
    cup_orientation_reward_weight = 1.
    cup_orientation_reward = -(cup_orientation_diff *
    ↪ cup_orientation_reward_weight)

    grasp_temperature_1 = 0.25
    grasp_temperature_2 = 0.25
    right_grasp_reward = torch.exp(-grasp_temperature_1 * right_cup_handle_dist)
    left_grasp_reward = torch.exp(-grasp_temperature_2 * left_cup_handle_dist)
    grasp_reward = (right_grasp_reward + left_grasp_reward - 1.0)

    cup_linvel_norm = torch.norm(object_linvel, dim=-1)
    cup_linvel_penalty_weight = 0.2
    cup_linvel_penalty = -(cup_linvel_norm * cup_linvel_penalty_weight)

    touch_reward_temperature = 0.25
    touch_reward_weight = 0.125
    touch_reward = (torch.exp(-touch_reward_temperature * right_cup_handle_dist) +
    ↪ torch.exp(-touch_reward_temperature * left_cup_handle_dist) - 1.0) *
    ↪ touch_reward_weight

```

```
total_reward = grasp_reward + object_goal_distance_reward +  
    ↪ cup_orientation_reward + cup_linvel_penalty + touch_reward  
  
reward_dict = {  
    "grasp_reward": grasp_reward,  
    "object_goal_distance_reward": object_goal_distance_reward,  
    "cup_orientation_reward": cup_orientation_reward,  
    "cup_linvel_penalty": cup_linvel_penalty,  
    "touch_reward": touch_reward  
}  
  
return total_reward, reward_dict
```