ME5418 - Machine Learning in Robotics
Group 16

Ji-Dog Quadrupedal Robots Locomotion via Reinforcement Learning

---

Group members:

Bai Yue (ID: A0304233U)

Ji Shuchen (ID: A0305152U)

Zhang Binlin (ID: A0304090U)

October 28, 2024

1 Why

Compared to legged animals, the range of wheeled robots is limited. Quadruped robots emulating animal locomotion can extend their operational range, enabling efficient performance in complex environments like search and rescue, extreme sports, and rugged terrains. While legged robots aim to match animal flexibility, there is still progress to be made in improving their adaptability to diverse terrains.

Adaptability to environmental changes is crucial for quadruped robots. These robots face varied terrains, requiring flexibility to complete tasks. In our project, we first train the quadruped robot with a 4-joint configuration on flat terrain, then if possible, extend it to an 8-joint configuration to enhance adaptability. Although existing control algorithms excel in some scenarios, they often rely on pre-planned movements, lacking flexibility in dynamic environments.

In this task, quadruped robots use local perception so they can only sense nearby environments. We focus on how well the robot adjusts its gait and posture while moving to handle changing conditions.

2 Convention Algorithms

Traditional control methods struggle with complexity, requiring intricate algorithms, lengthy design processes, and extensive parameter tuning. For example, Model Predictive Control (MPC) is computationally intensive, while Proportional Integration Differentiation (PID) lacks adaptability. Biomimetic methods like the Central Pattern Generator (CPG) replicate biological rhythms, producing better repetitive motions.

Conventional gait planning typically defines motions for specific tasks, such as programmed joint angles and velocities for walking or grasping. Predefined transition rules guide gait adjustments based on sensor inputs or task phases, which work well in controlled environments but falter in dynamic or unknown conditions. For instance, when facing unexpected obstacles or complex terrains, these methods may fail to adapt gaits quickly or efficiently.

3 Problem Statement

Initially, the goal of our project is to design and train a quadruped robot with a 4-joint configuration (one joint on each leg) in our NVIDIA Isaac Sim environment. The agent will learn to efficiently move forward, maintain balance, and adapt to relatively simple terrains, such as flat surfaces, and slightly uneven ground, focusing on optimizing its gait, and stability.

After successfully training the agent with the 4-joint configuration, then if possible, we will extend the model to an 8-joint configuration (two joints on each leg). This extended model will be tasked with navigating more complex terrains in NVIDIA Isaac Sim, including slopes, and staircases. For now, we focus on the 4-joint model as a foundational step.

4 RL Cast

4.1 State Space

To achieve effective control of the agent, the state space can be represented by a multidimensional vector. The state space is defined as follows:

$$S = [x, y, z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, \phi, \theta, \psi, \theta_1, \theta_2, \ldots, \theta_n, \dot{\theta}_1, \dot{\theta}_2, \ldots, \dot{\theta}_n, c_1, c_2, c_3, c_4],$$

where:

- (x, y, z): Position of the center of mass of the agent in 3D space.
- (v_x, v_y, v_z): Translational velocities of the center of mass along the x, y, and z axes.
- $(\omega_x, \omega_y, \omega_z)$: Angular velocities of the center of mass around the x, y, and z axes.
- (ϕ, θ, ψ): Euler angles representing the orientation (roll, pitch, yaw) of the agent.
- $(\theta_1, \theta_2, \theta_3, \theta_4)$: Joint angles of all 4 joints.
- $(\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4)$: Angular velocities of all 4 joints.
- (c_1, c_2, c_3, c_4): The state of each foot's contact with the ground (Boolean values) where $c_i = 1$ indicates that the i-th foot is in contact with the ground and $c_i = 0$ indicates no contact.

4.2 Action Space

The action space for the agent will progress through three stages: joint angle control, joint velocity control, and joint torque control. This gradual approach allows the agent to develop increasingly advanced locomotion strategies.

By choosing the joint-angle control method, the agent can control the angles of each joint. These joint angles are typically normalized to a range of $[-1, 1]$, ensuring consistent and stable control, represented as follows:

$$\mathrm{A} = [\theta_1, \theta_2, \theta_3, \theta_4],$$

where θ_i represents the target joint angle.

Next, the agent will control the angular velocity of each joint, allowing for smoother and more dynamic leg movements. This is represented as:

$$\mathrm{A} = [\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4].$$

Finally, the agent will manage the torque applied to each joint, offering direct control over the forces driving the agent's movements. The torque control is expressed as:

$$\mathrm{A} = [\tau_1, \tau_2, \tau_3, \tau_4].$$

4.3 Reward Structure

To effectively guide the process of reinforcement learning, we employ mixed rewards including both sparse and dense rewards to make sure the agent has continuous feedback during training and is also motivated towards long-term goals. Rewards/Penalties without * mark are the basic reward structures that have existed since the beginning of the training. While rewards/penalties with * mark are supplementary reward structures added subsequently after completing the basic task, intending to further optimize the locomotion of the agent.

- **Sparse Rewards and Penalties**
  - **Long-Term Goal Reward:** Once the agent achieves the target position, it will get a huge goal reward R_{goal} which is a positive constant.
  - **Terrain Adaptation Reward*:** Every time the agent successfully traverses a challenging terrain such as slope or stairs, it will get a terrain adaptation reward $R_{terrain} = h_s$, where h_s is the total height of the slope or stairs.
  - **Fall Penalty:** Every time the agent falls down ($z = 0$), it will get a fall penalty P_{fall} which is a negative constant.
  - **Contact Point Penalty:** Contact point penalty is set to prevent the agent from unsuitable gaits. When the agent's four legs are either all off or all on the ground at the same time, or when both legs on the same side leave the ground simultaneously, it will receive a penalty $P_{contact}$ which is a negative constant.
- **Dense Rewards and Penalties**
  - **Progress Reward:** Progress reward encourages the agent to move towards the goal. The closer the agent is to the goal, the greater the progress reward: $R_{progress} = e^{-(|x-x_{goal}|+|y-y_{goal}|)}$.
  - **Mass Centre Reward:** To guide the agent to maintain a reasonable mass centre height, the closer the mass centre height is to the expected height, the greater the mass centre reward: $R_{mass} = e^{-|z-z_{goal}|}$.
  - **Velocity Reward*:** For better velocity control, the closer the translational velocity of the agent is to the expected velocity, the greater the velocity reward: $R_{velocity} = e^{-(|v_x-v_{xgoal}|+|v_y-v_{ygoal}|)}$.
  - **Stability Penalty:** To avoid violent locomotion, when the agent's roll and pitch angles exceed the maximum allowable range ($\Delta\phi_{threshold}$ and $\Delta\theta_{threshold}$), it will receive a non-zero stability penalty which is positively proportional to the value of the angles: $P_{stability} = \min(0, \Delta\phi_{threshold} - |\phi|) + \min(0, \Delta\theta_{threshold} - |\theta|)$.
  - **Energy Penalty*:** Energy Penalty is set to encourage energy-efficient movements: $P_{energy} = -\sum_{i=1}^{4} |\tau_i \cdot \dot{\theta}_i|$.
- **The Overall Reward:** The overall reward function is the linear sum of the above rewards and penalties:

$$\mathrm{R} = R_{goal} + k_1 \sum R_{terrain} + k_2 R_{progress} + k_3 R_{mass} + k_4 R_{velocity} + \sum P_{fall} + \sum P_{contact} + k_5 P_{stability} + k_6 P_{energy},$$

  where $k_1, k_2, k_3, k_4, k_5, k_6$ are positive constants to normalize rewards/penalties.

5 Possible RL Algorithms

Since quadruped robots often have a continuous action space, policy-gradient-based deep reinforcement learning methods are more suitable for locomotion control. To address the sensitivity to hyper-parameters, we will apply reward normalization and input standardization, which help stabilize the training process and reduce the impact of hyper-parameter variations. Some possible policy-gradient-based deep reinforcement learning methods are as follows:

- **Asynchronous Advantage Actor-Critic (A3C):** A3C is an on-policy gradient method that uses multiple agents to collect data simultaneously and share the network asynchronously. It can converge faster and achieve good exploration due to its parallelism. However, it lacks sample efficiency and is sensitive to hyper-parameters.
- **Proximal Policy Optimization (PPO):** PPO is an improvement over TRPO, introducing penalties for large policy changes. It is stable and easier to implement than TRPO, but it is still less sample efficient than off-policy methods and sensitive to hyper-parameters.