BENCHMARKING MODEL PREDICTIVE CONTROL AND REINFORCEMENT LEARNING FOR LEGGED ROBOT LOCOMOTION

Shivayogi Akki
Michigan Technological University, sakki@mtu.edu

BENCHMARKING MODEL PREDICTIVE CONTROL AND REINFORCEMENT LEARNING FOR LEGGED ROBOT LOCOMOTION

By

Shivayogi Akki

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Mechatronics

MICHIGAN TECHNOLOGICAL UNIVERSITY

2023

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Mechatronics.

	Department of Electrical and Computer Engineering
Report Advisor:	*Dr. Tan Chen*
Committee Member:	*Dr. Vinh Nguyen*
Committee Member:	*Dr. Tao Liu*
Department Chair:	*Dr. Jin W. Choi*

Table of Contents

List of Figures

Acknowledgements

I express my gratitude towards Dr. Zherong Pan for his unwavering support in this research. I am also thankful to Michigan Technological University for providing me with the necessary equipment and resources to carry out the research successfully.

Abstract

This research delves into the realm of quadrupedal robotics, focusing on the comparative analysis of Model Predictive Control (MPC) and Reinforcement Learning (RL) as predominant control strategies. Through the comprehensive dataset compiled and the insights derived from this analysis, this research aims to serve as a valuable resource for the legged robotics community, guiding researchers and practitioners in the selection and implementation of control strategies. The ultimate goal is to contribute to the advancement of legged robot capabilities and facilitate their successful deployment in real-world applications.

In this study, we employ the Unitree Go1 quadrupedal robot as a testbed, subjecting it to a variety of conditions including different terrains and external perturbations to assess the performance of MPC and RL controllers. Our findings reveal that RL exhibits superior force rejection in scenarios involving external forces, albeit relying heavily on torque in a single joint, while MPC provides a balanced torque distribution across all joints. In stumbling scenarios, MPC outperforms RL in recovery time, although both controllers face challenges when the robot falls into a failure state.

Furthermore, the generalization capabilities of RL are evaluated across different terrains, demonstrating a performance drop in slippery conditions and uneven terrains compared to flat frictional surfaces. The temporal demands of RL, encompassing optimization and training phases, are contrasted with the real-time operation and parameter flexibility of MPC.

1 Introduction

The field of robotics has witnessed a paradigm shift with the advent of quadrupedal robots, distinguished by their exceptional ability to navigate complex terrains, surmount obstacles, and maintain balance in challenging environments. This unique set of capabilities positions them as ideal candidates for a broad spectrum of applications, ranging from industrial transport and package delivery, to search and rescue operations, and even planetary exploration. The versatility and agility of legged locomotion, however, bring forth a series of intricate design and control challenges that necessitate advanced solutions for ensuring stability and robust performance.

Within the vast landscape of control strategies for legged robots, Model Predictive Control (MPC) and Reinforcement Learning (RL) have emerged as predominant forces. MPC's strength lies in its ability to utilize explicit models of the robot and its environment, facilitating predictive optimization of the robot's behavior over a finite horizon. RL, on the other hand, adopts a model-free paradigm, learning optimal control policies through experiential interactions with the environment. Each of these methodologies exhibits unique advantages and potential drawbacks, and the selection of the appropriate controller is crucial for optimizing the robot's performance under varying conditions.

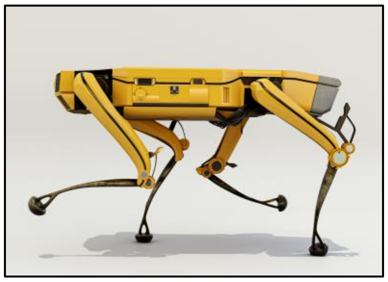


Figure 1. Quadruped robot

Despite the significant progress in the field of legged robotics, with notable examples including Boston Dynamics' Spot (Figure 1) and ANYmal showcasing impressive capabilities, there remains a noticeable lack of comprehensive comparative studies between MPC and RL controllers. This research endeavors to fill this gap by providing a thorough evaluation of these two major controllers when applied to the Unitree Go1 quadrupedal robot. By exposing the controllers to a diverse array of conditions, encompassing flat, slippery, and uneven terrains, as well as introducing perturbations and model uncertainties, we aim to meticulously assess their performance.

By compiling and analyzing this comprehensive dataset, this research aspires to serve as a valuable resource for the legged robotics community. We aim to empower researchers and practitioners with the knowledge and insights needed to make informed decisions in the selection and implementation of control strategies, ultimately contributing to the advancement of legged robot capabilities and their successful deployment in real-world applications.

2 Background and Related Work

The journey of quadruped robots began with the dream of creating machines capable of moving with the grace and agility of animals. The early days were marked by simplicity and rigidity, as roboticists sought to understand the complexities of legged locomotion. These initial robots were programmed with predefined movements, lacking the ability to adapt to the ever-changing environment around them.

2.1 Model Predictive Controller and Reinforcement Learning

The realization that true locomotion required adaptability led to the exploration of advanced control strategies. Model Predictive Control (MPC) emerged as a powerful tool, enabling robots to predict their future states and adjust their movements accordingly. Kim, J., & Park, F. C. (2019)[1] demonstrated the versatility of MPC in achieving dynamic walking in quadruped robots, highlighting its potential to transform legged locomotion.

The work of Winkler et al.[2] marked a significant milestone, introducing a phase-based parameterization of end-effector trajectories, seamlessly integrated into an MPC framework. This innovation allowed for unprecedented levels of stability and agility, setting a new standard for quadruped locomotion. The MIT Cheetah series, renowned for its speed and agility, showcased the true potential of MPC in quadruped robots. Di Carlo et al.[3] pushed the boundaries further with the Cheetah 3, implementing a convex MPC approach to achieve dynamic locomotion, even in the absence of visual feedback. As roboticists sought to deploy quadruped robots in increasingly complex environments, the importance of terrain perception came to the forefront. Focchi, M., et al. (2018)[4] explored the integration of terrain perception with MPC, aiming to enhance the robot's ability to navigate uneven terrains and overcome obstacles.

The advent of Reinforcement Learning (RL) marked the beginning of a new era, as robots transitioned from programmed to learned behavior. Hwangbo et al.[5] leveraged deep RL to teach quadruped robots agile and dynamic motor skills, demonstrating the power of learning in achieving previously unimaginable levels of performance. The application of RL to quadruped locomotion presented unique challenges, especially when it came to navigating challenging terrains. Xie et al.[6] delved into this domain, presenting a hardware case study that highlighted the practical challenges and potential solutions in deploying RL for real-world scenarios.

Recognizing the complementary nature of MPC and RL, researchers began to explore the integration of these two paradigms. Grandia, R., et al. (2019)[7] demonstrated the synergy between learning-based terrain estimation and MPC, paving the way for more robust and adaptable quadruped locomotion.

The journey of quadruped robots is far from over. As we look to the future, the integration of advanced control strategies, learning algorithms, and sensory perception holds the promise of creating quadrupeds that are not just stable and agile, but also intelligent and autonomous. Mastalli, C., et al. (2020)[8] presented a vision of this future, exploring the use of whole-body MPC for versatile and dynamic locomotion in quadruped robots.

2.2 MuJoCo Overview

MuJoCo[9], short for "Multi-Joint dynamics with Contact," is a physics engine designed to simulate and analyze the behavior of systems with complex interactions between rigid bodies and contacts. It is widely used in the fields of robotics, biomechanics, computer graphics, and machine learning. MuJoCo has gained popularity in the robotics community for its efficiency, accuracy, and ability to handle complex dynamics. Here's a more detailed overview of its features and capabilities:

1. **Physics Simulation:** MuJoCo uses a continuous time formulation to accurately model the dynamics of multi-body systems. It employs a differential algebraic equation (DAE) solver to simulate the motion of rigid bodies, including their interactions with other bodies and the environment.
2. **Contact Modeling:** One of the most challenging aspects of simulating robotic systems is accurately modeling contacts between rigid bodies. MuJoCo excels at this by using a soft contact model, which considers the deformation and friction between contacting surfaces. This approach allows for more realistic and stable simulations.
3. **High Efficiency:** MuJoCo is optimized for efficiency and can simulate complex systems with a large number of bodies and contacts in real time. This makes it suitable for applications that require high-speed simulations, such as real-time robot control or machine learning training.
4. **Modeling and Visualization:** MuJoCo provides an XML-based modeling language, allowing users to define and customize their simulations. The engine also comes with a built-in visualization tool that enables users to interactively view and analyze their simulations.
5. **Control and Reinforcement Learning:** MuJoCo is widely used in the development and testing of control algorithms and reinforcement learning policies. It provides APIs for various programming languages, including Python, C++, and MATLAB, enabling users to integrate their control and machine learning code with the simulation environment.
6. **Licensing and Availability:** MuJoCo is a proprietary software developed by Emo Todorov at the University of Washington. It offers free licenses for academic use, while commercial licenses are available for purchase. The engine is supported on multiple platforms, including Windows, Linux, and macOS.

In summary, MuJoCo is a powerful physics engine that offers accurate and efficient simulation of multi-body systems with complex dynamics. Its capabilities make it a popular choice for researchers and developers in robotics, biomechanics, and machine learning.

3 Methodology

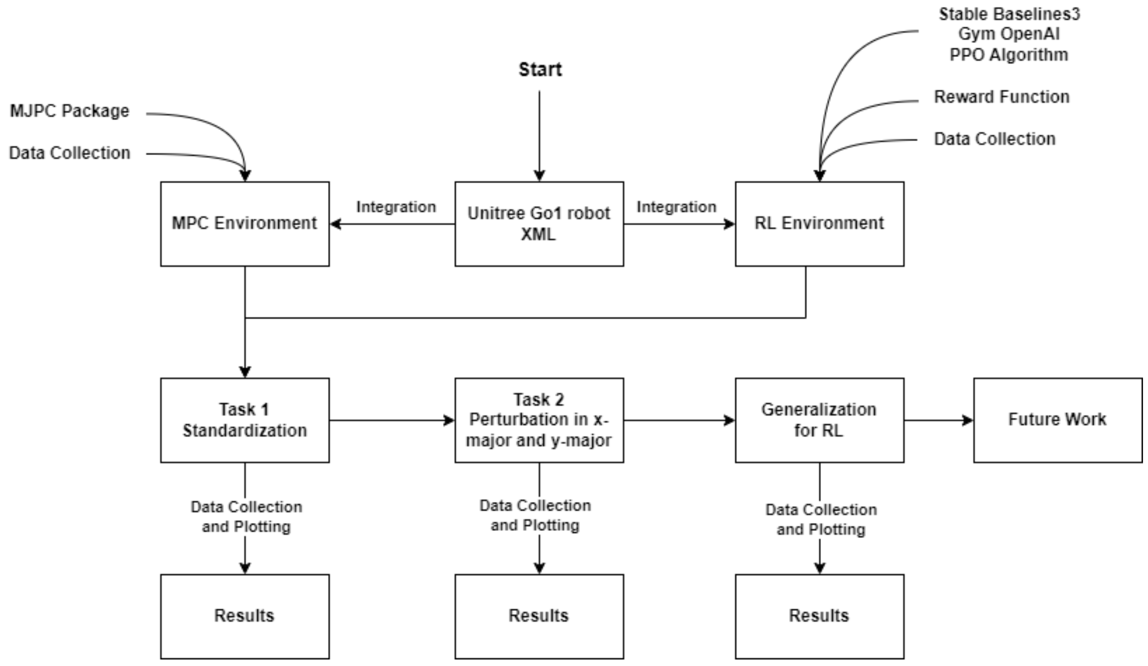


Figure 2. Overall workflow

The experiment is initialized by defining Unitree Go1 robot XML file (Figure 2). After initialization, the robot is integrated into MPC and RL environments. To establish MPC environment, MJPC package is utilized with some modifications for data collection, while establishment of RL environment involves Stable Baselines3 (rl-baselines3 zoo) and OpenAI Gym packages with modifications for reward function and data collection. In continuation, a series of tasks are performed followed by data collection and analysis at the end of each task to compare the two controllers.

3.1 Simulation Environment

3.1.1 Go1 Body Description

The robot XML model in the simulation environment replicates the real-world Unitree Go1 robot. It consists of a body with four legs, each with three joints. The robot's body and legs are defined as rigid bodies, while the joints are modeled as hinge joints that allow rotation in a single plane. The model includes accurate mass, inertia, and friction properties, ensuring that the simulation closely matches the real-world robot's behavior.

3.2 Controller

We set up two separate simulation environments for the MPC and RL controllers. Both environments share the same Go1 robot model and base physics settings, but they differ in their control inputs and reward functions.

3.2.1 MPC Controller

Model Predictive Control (MPC) is a model-based control strategy extensively used in robotics. In this research, we adapt MPC to manage the movements of a robot by predicting its future states and optimizing its trajectory based on a dynamic model and predefined objectives. The optimization problem at the heart of MPC is solved iteratively at each time step, producing a sequence of optimal control inputs.

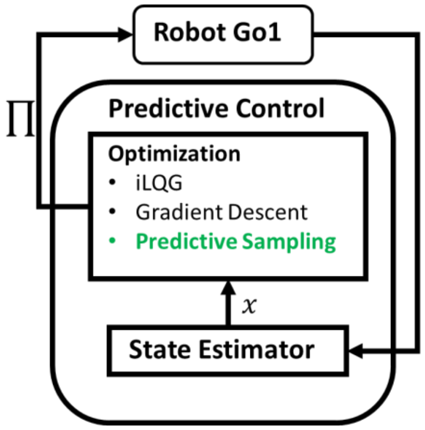


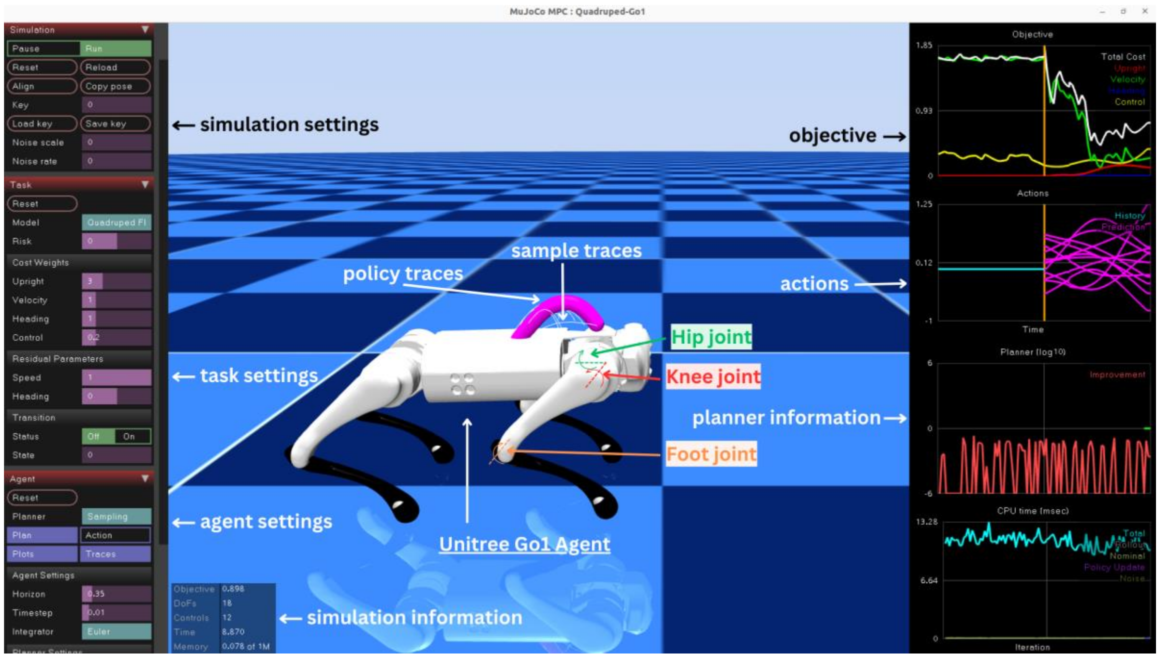
Figure 3. MPC Controller

The core of our approach lies in a predictive sampling algorithm, showcased in Figure 3. This algorithm refines a nominal sequence of actions, expressed as spline parameters, through a stochastic search process. In each iteration, we evaluate a set of N candidate splines. This set comprises the nominal spline and (N-1) perturbed samples, generated following a Gaussian distribution. All actions are kept within defined control limits. The candidate that yields the highest total return is selected as the best option, ensuring the robot's trajectory is continually optimized.

Originally, the MuJoCo MPC package, our tool of choice for this application, was calibrated and optimized for the Unitree A1 robot, renowned for its agility and robust performance. Inspired by the success of this application, we have extended the utility of the MPC approach to another robot in the Unitree lineup, the Go1 robot. Drawing parallels with the A1's environmental setup, we have crafted a specific Go1 XML environment to accommodate three distinct terrains: a flat fractioned surface, a flat frictionless surface, and an uneven surface. Through this adaptation, we aim to harness the power of MPC to enhance the Go1 robot's navigational capabilities across varied terrains.

3.2.1.1 MuJoCo MPC (MJPC) Overview

MuJoCo MPC (MJPC) is an open-source, interactive application and software framework for real-time predictive control, developed by Google DeepMind. MJPC is based on MuJoCo, a physics simulator that is widely used in robotics and machine learning. Figure 4 represents MPC simulation environment provided by the MJPC package using MuJoCo.



Simulation
Pause Run
Reset Reload
Align Copy pose
Key 0
Load key Save key
Noise scale 0
Noise rate 0

Task
Reset
Model Quadruped Fl
Risk 0
Cost Weights
Upright 3
Velocity 1
Heading 1
Control 0.2
Residual Parameters
Speed 1
Heading 0
Transition
Status Off On
State 0

Agent
Reset
Planner Sampling
Plan Action
Plots Traces
Agent Settings
Horizon 0.35
Timestep 0.01
Integrator Euler

Objective	0.898
DoFs	18
Controls	12
Time	8.870
Memory	0.078 of 1M

Figure 4. MPC Simulation environment

MJPC allows users to easily author and solve complex robotics tasks. It currently supports three shooting-based planners: derivative-based iLQG and Gradient Descent, and a simple derivative-free method called Predictive Sampling. It is a powerful tool for developing and testing predictive control algorithms. It is also a valuable resource for teaching and learning about predictive control.

Here is a brief overview of how MJPC works:

i. The user defines a task in terms of a desired goal state and a cost function that penalizes deviations from the goal state.
ii. MJPC uses a physics simulator to predict the system's dynamics over a time horizon.
iii. MJPC then optimizes a sequence of control actions to minimize the cost function while satisfying the system's constraints.
iv. The first control action in the sequence is executed on the real system.
v. Steps 2-4 are repeated in real time.

MJPC has been used to solve a variety of robotics tasks, including walking, running, jumping, and grasping. It has also been used to develop control algorithms for autonomous vehicles and other types of robots. In this study, MJPC environment will be used for MPC implementation on the Unitree Go1 robot.

3.2.2 RL Controller

Reinforcement Learning (RL) is a subset of machine learning focused on training agents to make decisions through interactions with their environment, adopting a model-free approach. The agent, in this context, takes actions based on observations and receives feedback in the form of rewards, aiming to maximize the expected cumulative reward over time, a process reminiscent of human learning from interaction and feedback.

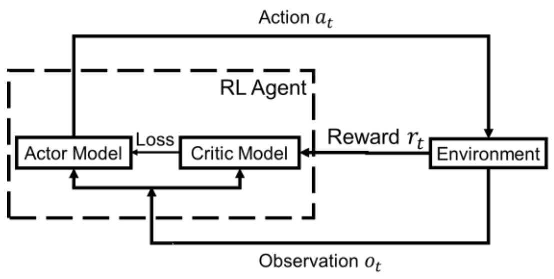


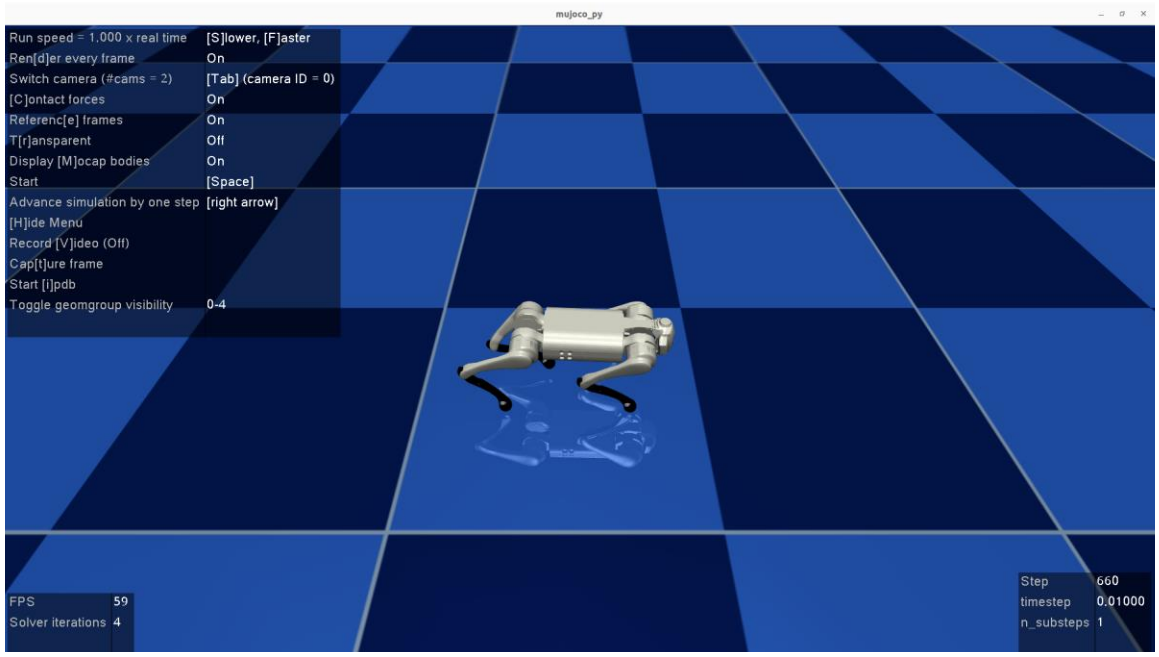
Figure 5. RL Controller

The fundamental components of RL encompass the agent (Figure 5), the decision-maker; the environment, where the agent operates and receives observations and rewards; the critic, a model predicting the value of state-action pairs aiding in action selection; and the actor, a model determining the agent's actions, which is updated based on the critic’s assessments and the agent’s experiences. The RL process involves the agent perceiving its environment, deciding on an action, acting, observing the new state, receiving a reward, and updating the critic and actor models. This loop continues until an episode concludes or a goal is reached, with the agent progressively favoring actions leading to higher rewards.

A pivotal challenge in RL is managing the exploration-exploitation trade-off. Exploration involves trying new actions to understand their effects, whereas exploitation involves choosing the best-known action to maximize rewards. Striking the right balance is crucial for learning an optimal policy.

This study focuses on the Unitree Go1 robot, with its environment modeled akin to the “HalfCheetah-v3” task from OpenAI Gym's MuJoCo environments. **The Go1 robot is tasked with moving along the x-axis while maintaining a 0.5 m/s velocity**, learning in a manner like the two-legged robot in “HalfCheetah-v3”, which aims to run forward swiftly.

In training the Go1 and HalfCheetah robots, the rl-baselines3-zoo library provides a suite of RL algorithms. Among them, Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Twin Delayed Deep Deterministic policy gradient (TD3) are predominant. PPO ensures stable policy updates, SAC promotes exploration for a balanced policy through a maximum entropy framework, and TD3 mitigates Q-value overestimation bias using dual Q-functions.



```
Run speed = 1.000 x real time    [S]lower, [F]aster
Ren[d]er every frame             On
Switch camera (#cams = 2)        [Tab] (camera ID = 0)
[C]ontact forces                 On
Referenc[e] frames               On
T[r]ansparent                    Off
Display [M]ocap bodies           On
Start                            [Space]
Advance simulation by one step   [right arrow]
[H]ide Menu
Record [V]ideo (Off)
Cap[t]ure frame
Start [i]pdb
Toggle geomgroup visibility      0-4

FPS                59
Solver iterations  4

Step        660
timestep    0.01000
n_substeps  1
```

Figure 6. RL Simulation environment

For this research, the Go1 robot is trained using the PPO algorithm, with hyperparameters meticulously tuned using Optuna. These parameters encompass learning rates, network architectures, batch sizes, and other algorithm-specific settings, ensuring optimal performance tailored to the Go1's unique task. Through this approach, the research taps into the depth of RL to empower the Go1 robot, enabling precise and efficient learning for navigation and control (Figure 6).

3.2.2.1 RL-baselines3-zoo

The rl-baselines3-zoo is a repository developed by Stable Baselines3, a popular library for Reinforcement Learning (RL) in Python. The rl-baselines3-zoo contains a collection of pre-trained RL agents using various algorithms, environments, and hyperparameters.

Applications of rl-baselines3-zoo include:

1. **Benchmarking RL Algorithms:** Researchers and practitioners can use pre-trained agents in rl-baselines3-zoo to benchmark and compare the performance of different RL algorithms on various environments.
2. **Transfer Learning:** Users can fine-tune pre-trained agents on their specific tasks or environments, enabling faster convergence and reducing the need for extensive training from scratch.
3. **Exploration and Learning:** Users can explore different environments and learn about RL by studying the provided pre-trained agents and their behavior.
4. **Experiment Reproducibility:** By using the provided pre-trained agents, researchers can reproduce results easily and verify the efficacy of their new algorithms or modifications.

Overall, rl-baselines3-zoo is a valuable resource for the RL community, facilitating research, development, and understanding of RL algorithms, as well as promoting reproducibility in the field. In this study, rl-baselines3-zoo environment was used for RL implementation on the Unitree Go1 robot.

3.2.2.2 Reward Function

Reward = FVR + HR + OR + LVR + CR …………………………………………….. (1)

where, FVR – Forward Velocity Reward

HR – Height Reward

OR – Orientation Reward

LVR – Lateral Velocity Reward

CR – Control Reward

FVR = Variable Reward × Forward Reward Weight …..…………………………….. (2)

where, Forward Reward Weight = 2.0,

$$\text{Variable Reward} = \begin{cases} -1\,;\, v_{x,CM} = 0, \\ qvel_{x,CM} - v_{x,target}\,;\, qvel_{x,CM} < v_{x,target}, \\ v_{x,target} - qvel_{x,CM}\,;\, qvel_{x,CM} > v_{x,target}, \\ v_{x,target}\,;\, otherwise, \end{cases}$$

$v_{x,target}$ = 0.5

This is for the robot to maintain a velocity of 0.5 m/s in the x axis.

$$\text{Height Reward} = \begin{cases} 3\,;\; qpos_{z,CM} = 0.26 \\ -\left|qpos_{z,CM} - 0.26\right|\,;\, otherwise \end{cases} \quad \text{..………………………………... (3)}$$

This is for the robot to maintain a standing posture.

Orientation Reward = $-|1 - w| - |x| - |y| - |z|$ …………………………………. (4)

where x, y, z and w are the respective quaternions for the Center of Mass.

This is for the robot to look in the direction of motion.

Lateral Velocity Reward = $-\left|qvel_{y,CM}\right|$ ……………………………………………..(5)

This is for the robot to reduce the deviation in y axis.

Control Reward = − control signals × 0.05 …………………………………………..(6)

This is for the robot to reduce the control signal to increase stability.

3.3 Data Collection

MuJoCo has two prebuilt data collection systems, mjModel and mjData. The mjModel class defines and records details about the model which is mostly used for the simulation. The mjData on the other hand, stores details such as Position, Velocity, External Force applied, contact force, and so on for the center of mass and each joint of the 4 legs which makes it more important to us. A few of the parameters used for metric analysis are explained below.

A. Position (Go1 robot):

In the context of MuJoCo and robotic simulations, position typically refers to the spatial location of an object or a joint in a 3D space. It is usually represented by Cartesian coordinates (x, y, z). In mjData position is labeled "QPOS", a matrix of (19×1) for each timestep.

QPOS = $[qpos_{(i,j)}]$

$qpos_{(1-3,1)}$ = Position of center of mass of robot in x, y and z axis respectively

$qpos_{(4-7,1)}$ = Orientation of center of mass of robot in quaternions x, y, z and w respectively

Elements $qpos_{(8-19,1)}$ are classified, on rotation about hip (x), knee (y) and foot (y) each, as

i. $qpos_{(8-10,1)}$ = Angular position of front right leg
ii. $qpos_{(11-13,1)}$ = Angular position of front left leg
iii. $qpos_{(14-16,1)}$ = Angular position of rare right leg
iv. $qpos_{(17-19,1)}$ = Angular position of rare left leg

B. Velocity (Go1 robot):

Velocity describes the rate of change of position with respect to time. In a 3D space, it is a vector quantity with both magnitude (speed) and direction. For rotational joints, velocity might refer to the angular velocity, which represents the rate of change of the angular displacement with respect to time. In mjData position is labeled "QVEL", a matrix of (18×1) for each timestep.

QVEL = $[qvel_{(i,j)}]$

$qvel_{(1-3,1)}$ = Translational velocity of center of mass in x, y and z axis respectively

$qvel_{(4-6,1)}$ = Angular velocity of center of mass in x, y and z axis respectively

Elements $qvel_{(7-18,1)}$ are classified, on rotation about hip (x), knee (y) and foot (y) each, as

i. $qvel_{(7-9,1)}$ = Angular velocity of front right leg
ii. $qvel_{(10-12,1)}$ = Angular velocity of front left leg
iii. $qvel_{(13-15,1)}$ = Angular velocity of rare right leg
iv. $qvel_{(16-18,1)}$ = Angular velocity of rare left leg

C. Control (Go1 robot):

Control, in the context of robotic simulations, usually refers to the commands or actions applied to the system to influence its behavior. The control inputs are typically the result of some control policy or algorithm, which could be hand-designed or learned through methods like reinforcement learning. In mjData position is labeled "CTRL", a matrix of (12×1) for each timestep.

CTRL = $[ctrl_{(i,j)}]$

Elements $ctrl_{(1-12,1)}$ are classified, into hip (x), knee (y) and foot (y) each, as

$ctrl_{(1-3,1)}$ = Control torque for hip, knee and foot joint of front right leg respectively

$ctrl_{(4-6,1)}$ = Control torque for hip, knee and foot joint of front left leg respectively

$ctrl_{(7-9,1)}$ = Control torque for hip, knee and foot joint of rare right leg respectively

$ctrl_{(10-12,1)}$ = Control torque for hip, knee and foot joint of rare left leg respectively

External Force (Go1 robot):

External forces are forces applied to the system from outside sources. This could include things like a push from a human, gravity, or forces due to collisions with other objects. In legged robots, understanding external forces is crucial, especially when navigating uneven terrains or interacting with the environment. The forces can influence the robot's stability and movement patterns. In mjData position is labeled "XFRC_APPLIED", a matrix of (14×6) for each timestep.

XFRC_APPLIED = $[xfrc\_applied_{(i,j)}]$

$xfrc\_applied_{(i,j)}$, where I represents body number (name assigned) and j represents force in x, y and z and torque in about x, y and z.

$xfrc\_applied_{(1,j)}$ = World body

$xfrc\_applied_{(2,j)}$ = Trunk

Elements $xfrc_applied_{(3-14,j)}$are classified, into hip (x), knee (y) and foot (y) each, as

$xfrc\_applied_{(3-5,j)}$ = Front right leg

$xfrc\_applied_{(6-8,j)}$ = Front left leg

$xfrc\_applied_{(9-11,j)}$ = Rare right leg

$xfrc\_applied_{(12-14,j)}$ = Rare left leg

3.3.1 MPC Data Collection environment

The mjData file was modified to append multiple timestep data. Here, the external perturbation force was applied through the MuJoCo MPC package GUI and all the variable data, namely position ($qpos_{(i,j)}$), velocity ($qvel_{(i,j)}$), external force applied ($xfrc\_applied_{(i,j)}$) and control ($ctrl_{(i,j)}$), was exported to a .csv file. Analysis is done in MATLAB.

3.3.2 RL Data Collection environment

From above variables (Section 3.3), external force applied ($xfrc\_applied_{(i,j)}$) are used as input to feed the perturbation data while the other variables, namely position ($qpos_{(i,j)}$), velocity ($qvel_{(i,j)}$) and control ($ctrl_{(i,j)}$) are the outputs. All this data was exported through a python code, appending multiple timestep data, into a .csv file and analyzed in MATLAB.

4 Experiments and Results

4.1 Experiment 1: Standardization of task

Prior to conducting a comparison between the two controllers—Model Predictive Control (MPC) and Reinforcement Learning (RL)—it is crucial to undertake a standardization procedure. This process ensures that both the controllers and their corresponding environments are finely tuned and calibrated, facilitating a fair and meaningful comparison.

In this specific scenario, the objective is to maintain a steady velocity of 0.5 m/s using the Unitree Go1 quadruped robot. By examining the provided plots in Figure 7, which illustrate the relationship between position, velocity, control torque, and time, we can glean valuable insights into the performance of both controllers.

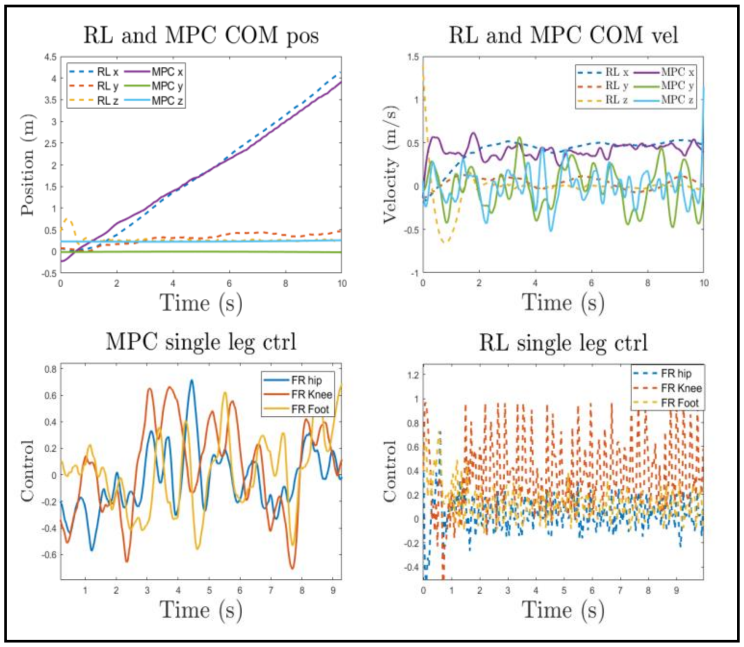


Figure 7. Comparison of Go1 maintaining 0.5 m/s with MPC and RL

From the position versus time graph, it is evident that the trajectories of both MPC and RL are relatively linear and closely aligned. However, the MPC demonstrates a slight instability in comparison to its counterpart. Analyzing the velocity versus time plot, we observe that both controllers successfully uphold an average velocity of 0.5 m/s along the x-axis, while maintaining a velocity of 0 m/s in both the y and z axes. Notwithstanding, the MPC showcases greater instability in its velocity profile.

When we turn our attention to the control torque versus time graph, noticeable differences emerge between the two controllers. The MPC exhibits control torque signals of a lower frequency compared to the RL, and substantial movements are observed across all three joints (hip, knee, and foot) for the MPC. On the other hand, the RL controller manifests minimal control torque in the hip and foot joints, with a more pronounced and consistent torque present in the knee joint.

In summary, while both controllers are capable of maintaining the desired velocity, the MPC demonstrates signs of instability in both position and velocity. The control torque signals of the MPC also exhibit a unique profile, distinguished by lower frequency and significant movements across all joints. The RL controller, in contrast, achieves a more stable performance with a distinctive torque distribution across the joints.

4.2 Experiment 2: Performance under perturbation

Having validated the standardization of both the controllers and their respective environments, we can now proceed to the perturbation testing phase.

4.2.1 Perturbation in x and y axis

Having validated the standardization of both the controllers and their respective environments, we can now proceed to the perturbation testing phase. In this context, two separate scenarios are presented, each depicted in distinct figures: Figure 8 showcases an Arbitrary Perturbation with a predominant component along the x-axis, measured at 270 N, while Figure 9 illustrates an Arbitrary Perturbation with its major component directed along the y-axis, registering at -270 N. Each figure provides a comprehensive visual analysis through plots of position versus time, velocity versus time, control torque versus time, and force versus time, all of which are demonstrated for both the Model Predictive Control (MPC) and Reinforcement Learning (RL) controllers.

Upon examination of the force plots, it is evident that the applied forces in both controllers remain constant, ensuring a uniform application of perturbation. This uniformity is crucial for accurately assessing the controllers' responses to external disturbances.

Focusing on the position plots, a clear trend emerges, highlighting the superior stability and linearity of the RL controller compared to the MPC. Furthermore, the RL controller exhibits a smaller deviation in response to the applied force, albeit with a very minimal difference. This trend is consistent and can be further validated through the velocity plots, where the impact of the force results in a more pronounced change in velocity for the MPC as opposed to the RL controller. The same can be noticed in Figure 8. with respect to y-axis.

In essence, the perturbation tests underscore the RL controller's enhanced stability and resilience to external forces, as reflected by its more stable position and velocity profiles. The MPC, while effective, demonstrates a higher susceptibility to changes in external conditions, as evidenced by the greater fluctuations in its position and velocity responses.

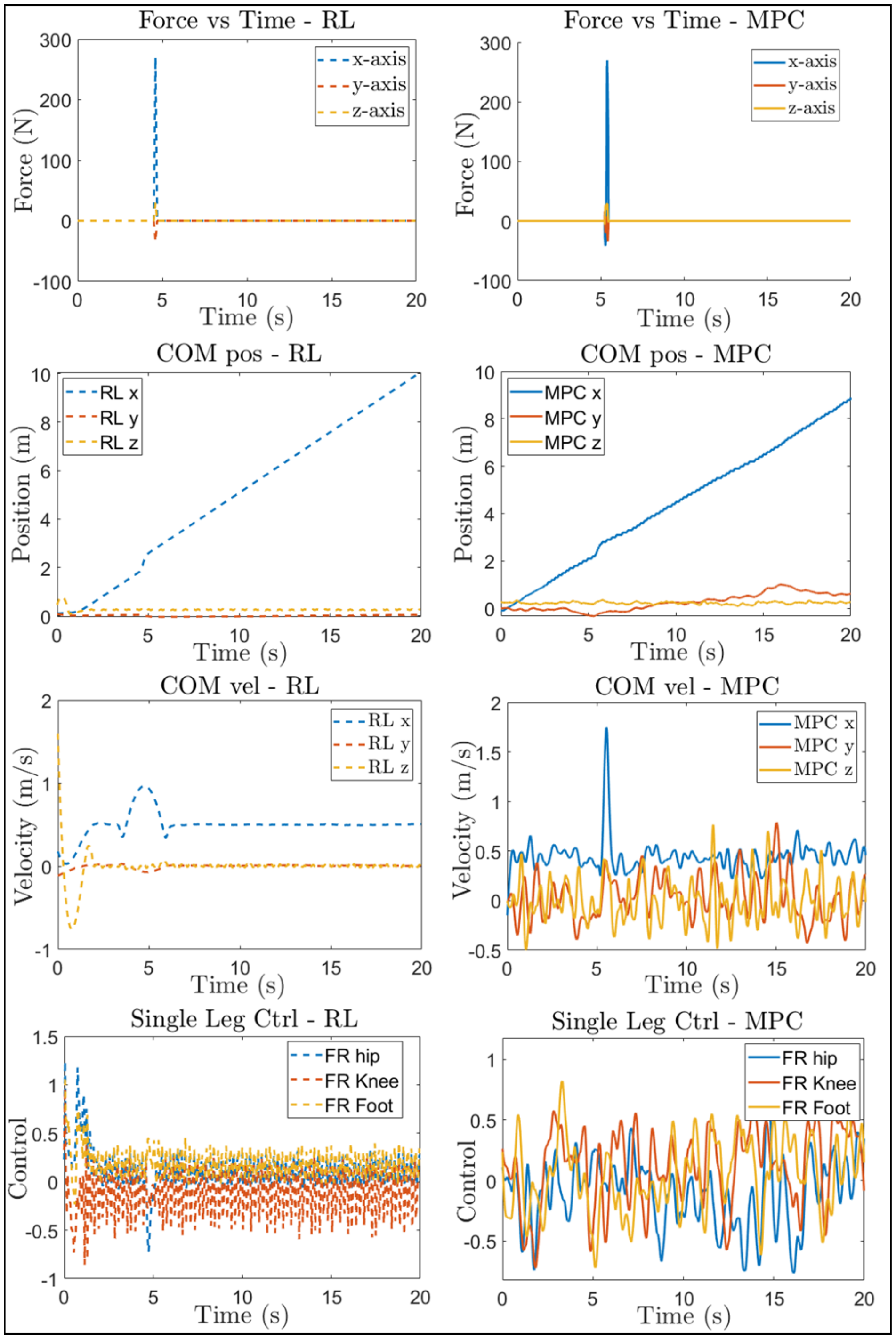


Figure 8. Perturbation 2 in x-axis with a 270 N

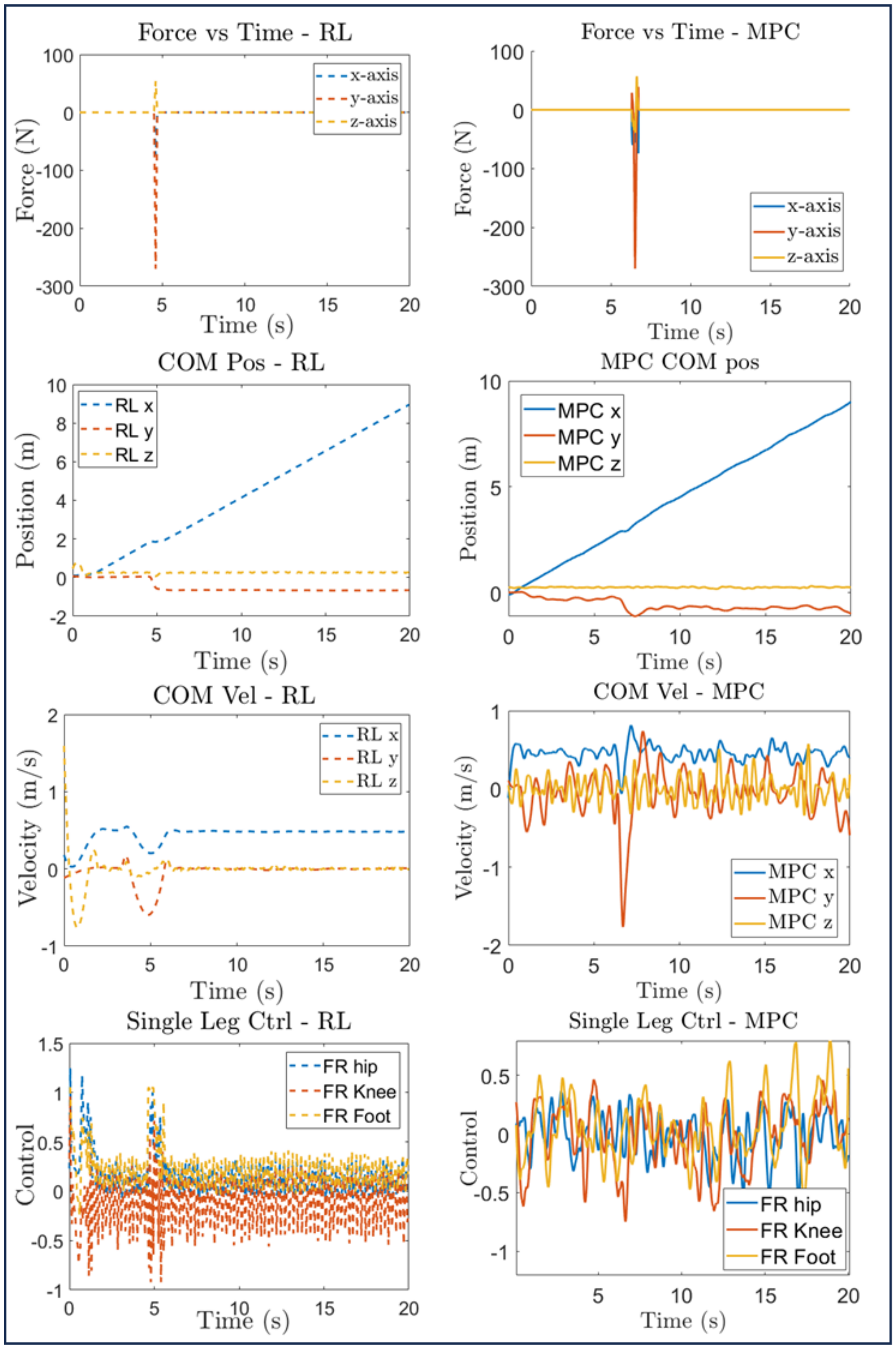


Figure 9. Perturbation 2 in y-axis with a -270 N

4.2.2 Perturbation showing recovery

While Reinforcement Learning (RL) demonstrates a marginally superior capacity for force rejection compared to Model Predictive Control (MPC), it is important to note that both controllers occasionally encounter failures. Such instances include scenarios where the robot’s legs become trapped, or the robot finds itself in an inverted position. There are also situations where the robot may stumble, momentarily making contact with the ground, yet manages to regain its balance and continue operation.

The following Figure 10 provides a visual representation of the robot’s recovery capabilities in these challenging circumstances. In this particular context, MPC exhibits a commendable performance, showcasing a significantly shorter recovery duration. This is primarily attributed to MPC's dependency on all three joints of the robot for locomotion.

On the contrary, the RL controller does not fare as well in terms of recovery time. The robot under RL control takes a longer duration to recover, which can be linked back to RL’s dependency on only one joint for most of its locomotion. This consistency, while generally advantageous, proves to be a hindrance in situations that require rapid and varied responses in all three joints to regain stability.

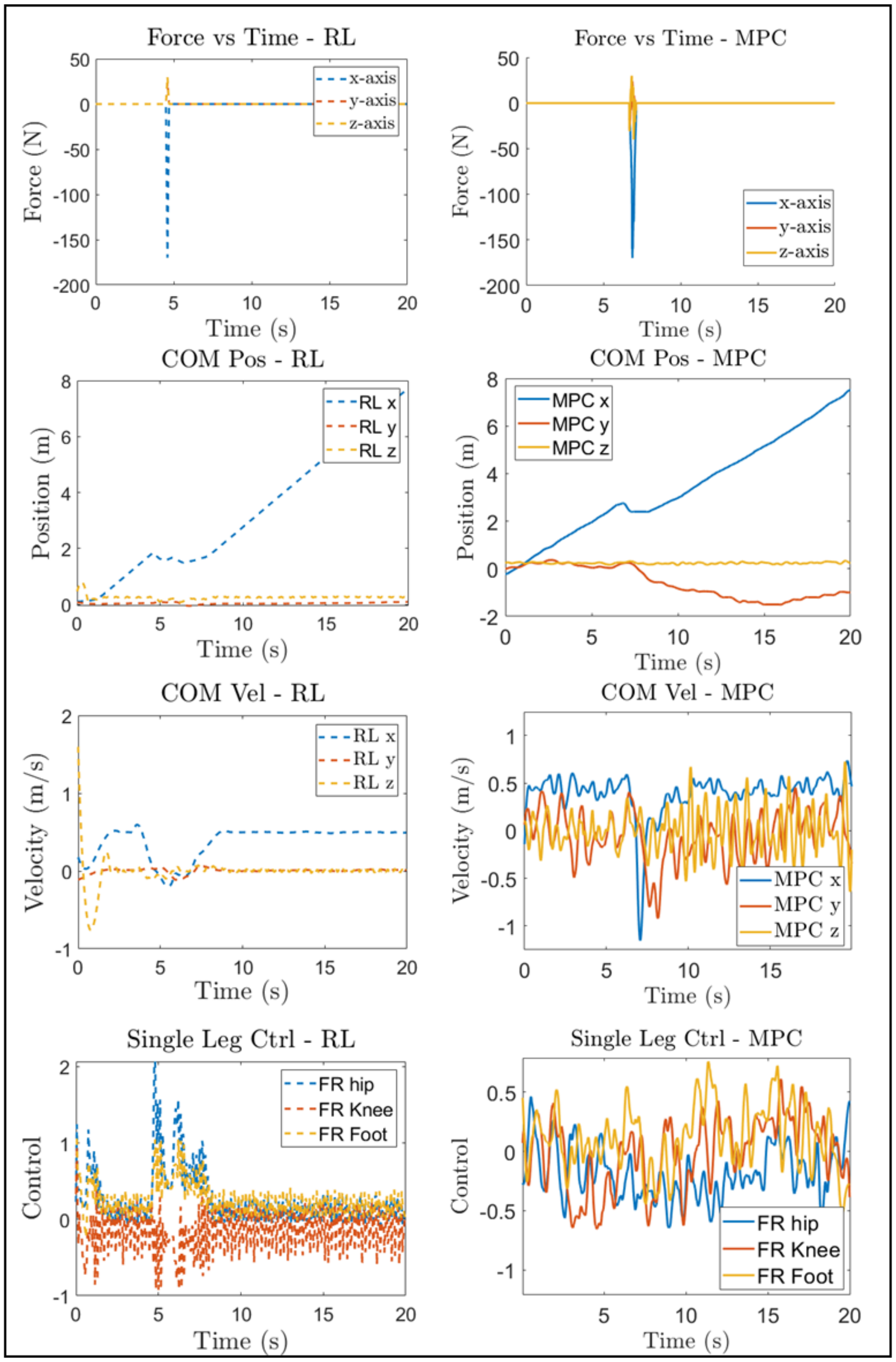


Figure 10. Perturbation 2 in x axis with a -170 N

4.3 Experiment 3: Generalization in RL

Task generalization is a crucial aspect of controller performance, assessing the controller's ability to adapt to varying conditions and environments. For instance, if a controller is specifically trained to navigate flat surfaces, its adaptability is put to the test when introduced to different terrains such as grasslands. Given the inherent unpredictability of real-world conditions, this capability is vital for the effective functioning of the controller.

Generalization across various tasks and environments presents a challenge for Model Predictive Control (MPC), largely due to its nature as a model-based controller. In MPC, decision-making and planning are conducted in real-time, with a direct reliance on the immediate and actual environmental conditions. This approach necessitates a precise and accurate model of the environment for optimal performance, which can be a limiting factor when encountering unforeseen or novel situations.

In contrast, Reinforcement Learning (RL) operates under a different paradigm. During the training phase, the robot is exposed to a specific environment, with the aim of learning optimal or near-optimal policies within that context. The underlying hope or assumption is that the training environment is representative of the conditions the robot will face during actual deployment.

In this particular analysis, the Unitree Go1 robot, governed by a Reinforcement Learning (RL) controller, has been initially trained to traverse a flat surface with friction. Subsequent to this training, the robot's adaptability is evaluated under two distinct conditions: a flat surface with reduced friction, and an uneven terrain. The ensuing performance under each scenario is meticulously captured through plots of position, velocity, and control torque against time.

4.3.1 Flat surface with friction

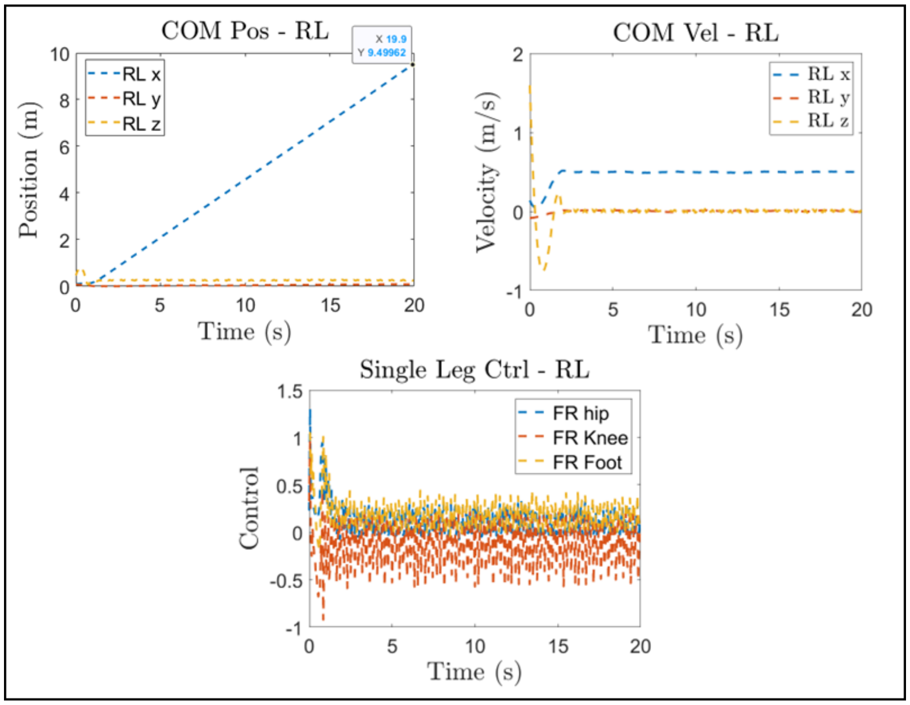


Figure 11. Friction Surface

On a flat surface endowed with friction, the robot exhibits commendable performance (Figure 11), covering a distance of 9.49962 meters within a span of 20 seconds. This aligns well with the set objective of maintaining a velocity of 0.5 m/s.

4.3.2 Flat surface with diminished friction

Upon transitioning to a flat surface with diminished friction, a noticeable decline in performance is observed. The robot's total distance covered drops to 8.93759 meters, and it struggles to quickly attain the target velocity of 0.5 m/s. This difficulty in acceleration is attributed to the reduced friction, hindering the robot's forward movement. The control torque plot in Figure 12 corroborates this observation, reflecting the challenges faced by the robot under these slippery conditions.

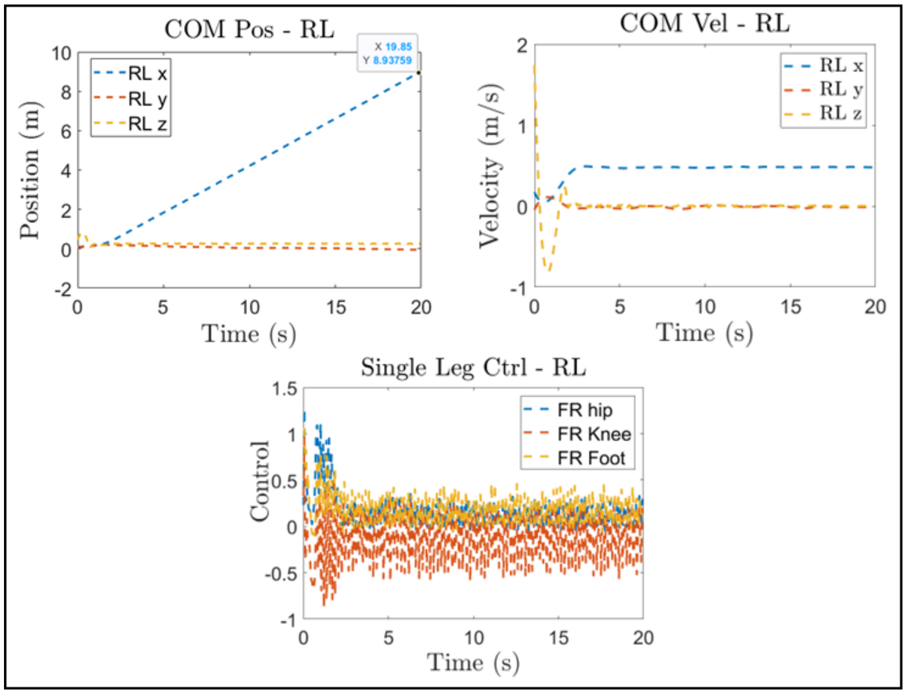


Figure 12. Slippery Surface

4.3.3 Uneven surface

The robot's performance is significantly impacted when introduced to an uneven terrain, as evidenced by the reduced distance of 5.98072 meters covered in the same 20-second timeframe. The complexity of navigating slopes contributes to this marked decrease in distance, consuming additional time for climbing. The position plot in Figure 13 reveals a departure from linearity, while the velocity and control torque curves exhibit fluctuations, indicating the inconsistency in the robot's movement across the uneven landscape.

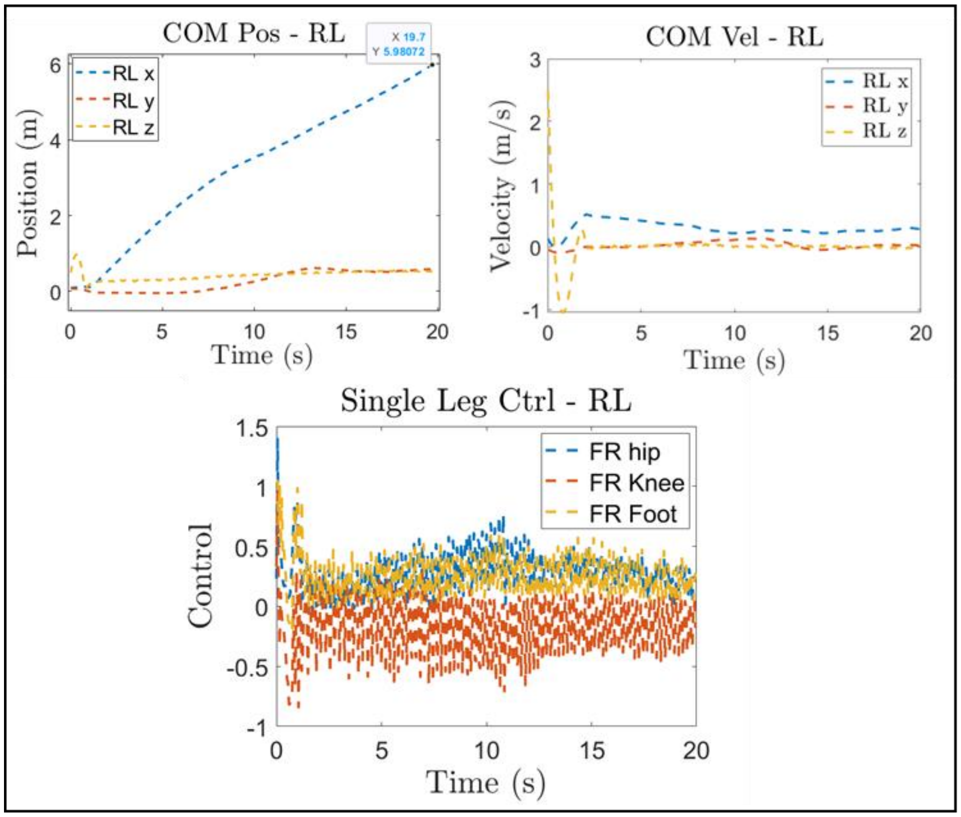


Figure 13. Uneven terrain

In summary, while the RL-controlled Go1 robot demonstrates proficiency in navigating flat, friction surfaces, its performance is noticeably affected when introduced to more challenging conditions such as slippery surfaces and uneven terrains. The reduction in friction leads to delayed acceleration and reduced distance covered, while the uneven terrain introduces complexities that result in non-linear movements and fluctuating velocity and control torque profiles. These scenarios underscore the importance of adaptability in controllers, highlighting areas for potential enhancement to ensure robust performance across diverse real-world conditions.

5 Conclusion

In our extensive exploration of legged robotics control strategies, this research meticulously evaluated the performances of Model Predictive Control (MPC) and Reinforcement Learning (RL) controllers across a diverse set of scenarios and environmental conditions. The dataset compiled and analyzed serves as a robust foundation for deriving critical insights into the operational nuances of these controllers.

In scenarios where the robot experienced an external force, RL demonstrated superior force rejection capabilities, though it predominantly relied on major control torque values in a single joint. On the other hand, MPC maintained a consistent torque across all joints, despite showcasing lower frequency control signals. In instances where the robot stumbled, MPC's ability to move three joints with high magnitude resulted in a faster recovery time compared to RL. However, both controllers were rendered ineffective when the robot fell into a position of failure.

Further testing of RL’s generalization revealed varied performances across terrains. On a flat frictional surface, the robot achieved a commendable distance of 9.49 meters in 20 seconds, aligning with the target velocity of 0.5 m/s. Conversely, on a slippery surface, the distance covered reduced to 8.9 meters, with a noticeable delay in acceleration, underscoring the impact of reduced friction. The uneven terrain posed additional challenges, reducing the distance traveled to 5.98 meters and introducing non-linearities in position and velocity plots.

From a practical standpoint, RL’s temporal expense due to its prerequisite optimization and training phases contrasts with MPC’s real-time operation and parameter adjustability. MPC's flexibility in real-time parameter manipulation spans velocity, direction, control, and noise, providing a strategic advantage in dynamic environments.

This research synthesizes these findings to offer a technical and nuanced perspective on legged robot control, aiming to empower the robotics community with the knowledge required for informed decision-making and further innovation in the field.

6 Reference List

1. J. Kim and F. C. Park, "Dynamic walking of a quadruped robot using model predictive control based on the linear inverted pendulum model," in IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3964-3971, 2019.
2. A. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," in IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 1560-1567, 2018.
3. J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1-9, 2018.
4. M. Focchi, G. Medrano-Cerda, M. Camurri, V. Barasuol, D. Caldwell, and C. Semini, "Heuristic planning for rough terrain locomotion in presence of external disturbances and variable perception quality," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1-9, 2018.
5. J. Hwangbo, J. Lee, A. Dosovitskiy, C. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," in Science Robotics, vol. 4, no. 26, 2019.
6. Lee, Joonho, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning quadrupedal locomotion over challenging terrain." Science robotics 5, no. 47 (2020): eabc5986.
7. R. Grandia, R. Ranftl, H. Boaventura, and M. Hutter, "Feedback MPC for Torque-Controlled Legged Robots," in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4735-4742, 2019.
8. C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control," in 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 2536-2542, 2020.
9. Todorov, Emanuel and Erez, Tom and Tassa, Yuval, “MuJoCo: A physics engine for model-based control,” in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5026-5033, 2012.

A Title of Appendix

MPC Simulation

Clone the repository,

```
git clone https://github.com/google-deepmind/mujoco_mpc.git
```

Configure the project with CMake (a pop-up should appear in VSCode)

Build and run the mjpc target in "release" mode (VSCode defaults to "debug"). This will open and run the graphical user interface.

RL Simulation

1. Optimization
   To optimize the hyperparameters, we employ Optuna.
   Adjust the PPO hyperparameters with a budget of 1000 trials and a maximum of 50000 steps, using a random sampler and median pruner in two concurrent jobs:

   ```
   python -m rl_zoo3.train --algo algo_name --env env_id -n 50000 -optimize --n-trials 1000 --n-jobs 2  --sampler random --pruner median
   ```

2. Training
   If this file contains the environment, you can use it to train an agent by using:

   ```
   python -m rl_zoo3.train --algo algo_name --env env_id
   ```

3. Simulation
   If the trained agent exists, then you can see it in action using:

   ```
   python -m rl_zoo3.enjoy --algo algo_name --env env_id
   ```