Learning agile and dynamic motor skills for legged robots

Stephane Hatgis-Kessell

10/26/23

Legged locomotion systems are very capable

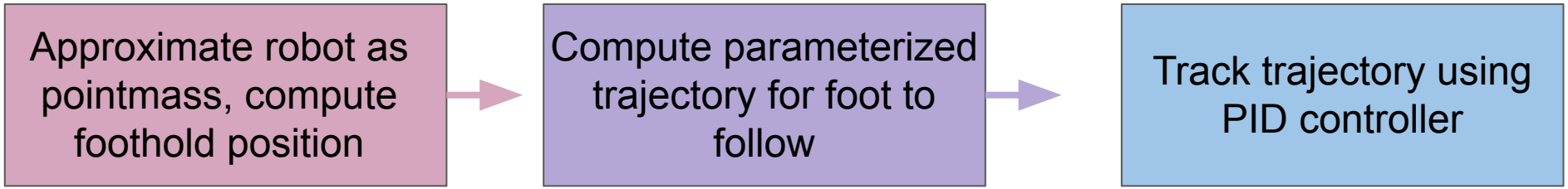
But designing control algorithms for legged robotics systems is very difficult.

- These robots are high-dimensional and nonsmooth systems with many physical constraints.
- Contact points change over the course of time and depending on the maneuver being executed
- Analytical models of the robots are often inaccurate and cause uncertainties in the dynamics.
- A complex sensor suite and multiple layers of software bring noise and delays to information transfer.

Conventional control theories are often insufficient to deal with these problems effectively, and specialized control methods are very difficult to design.

Conventional control theories are often insufficient to deal with these problems effectively, and specialized control methods are very difficult to design.

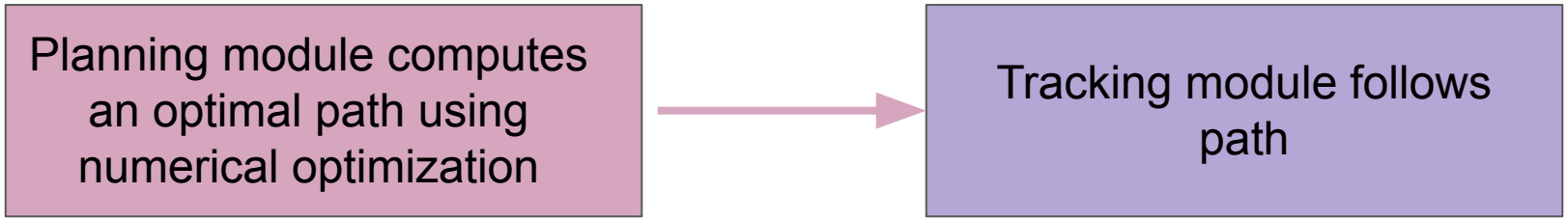
Modular controller design approaches:



See [7-12] from paper for examples of this approach.

Conventional control theories are often insufficient to deal with these problems effectively, and specialized control methods are very difficult to design.

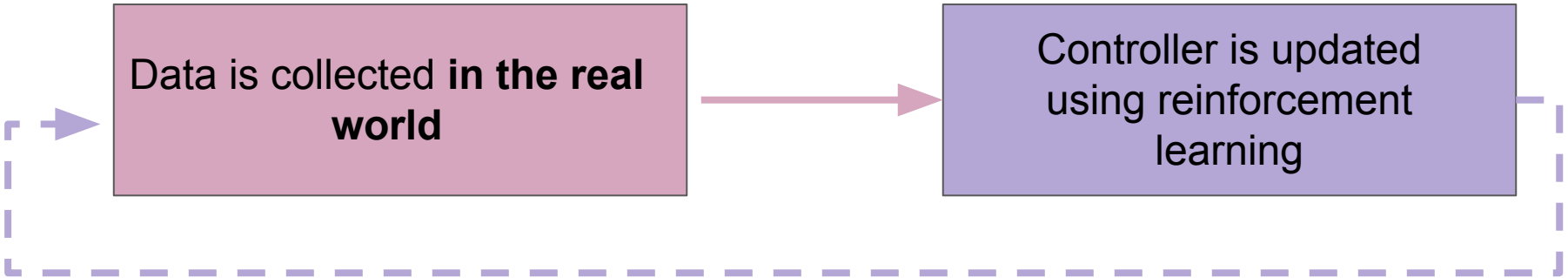
Trajectory optimization approaches:



See [13-17] from paper for examples of this approach.

Conventional control theories are often insufficient to deal with these problems effectively, and specialized control methods are very difficult to design.

Data-driven optimization approaches:



See [18-20] from paper for examples of this approach.

Can we train an RL policy for quadruped locomotion in simulation and then execute it in the real world?

Handling the simulation-reality gap

Improve simulation fidelity (analytically or data-driven)

Make controller robust to differences in variations, allowing for better transfer.

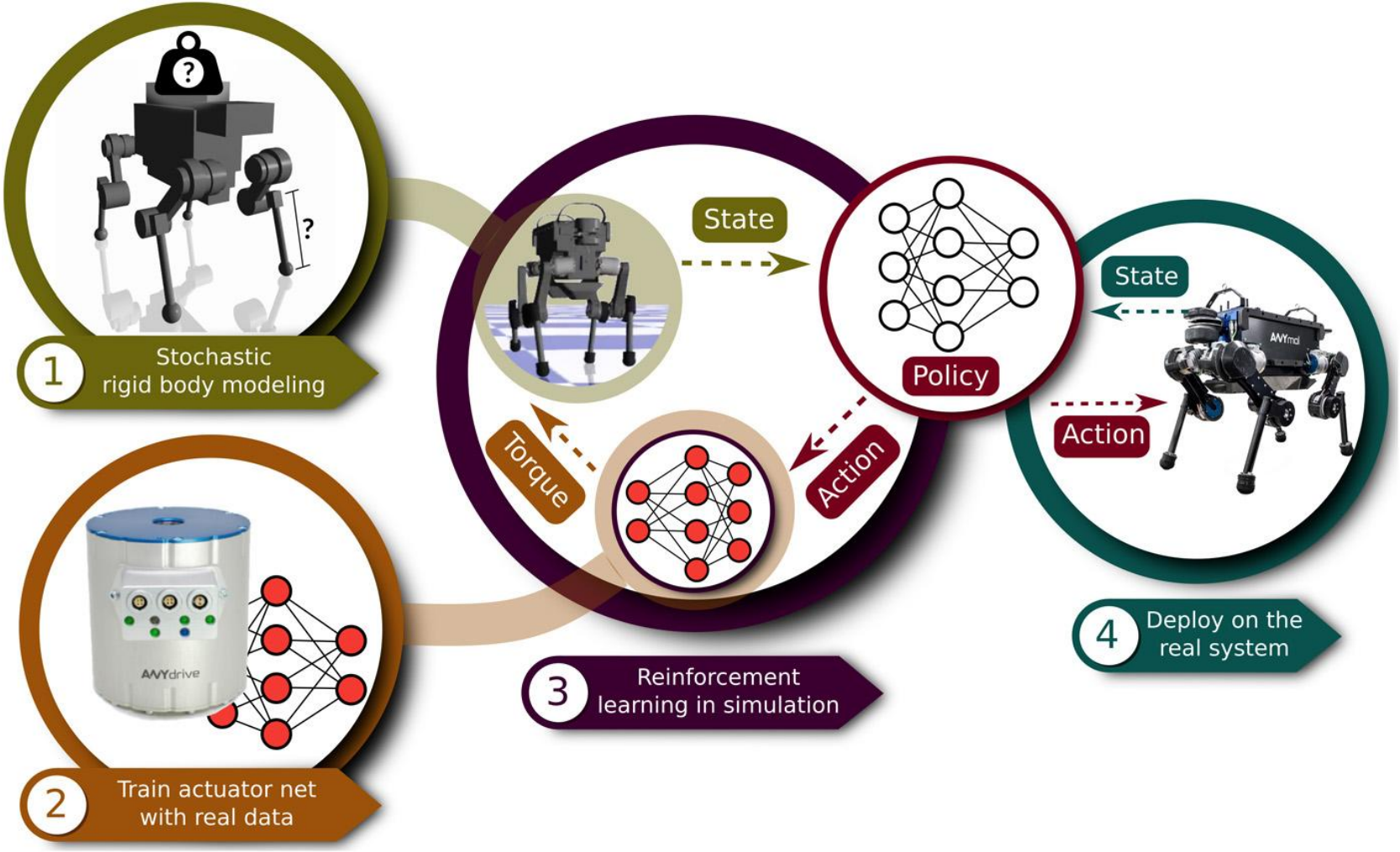
Handling the simulation-reality gap

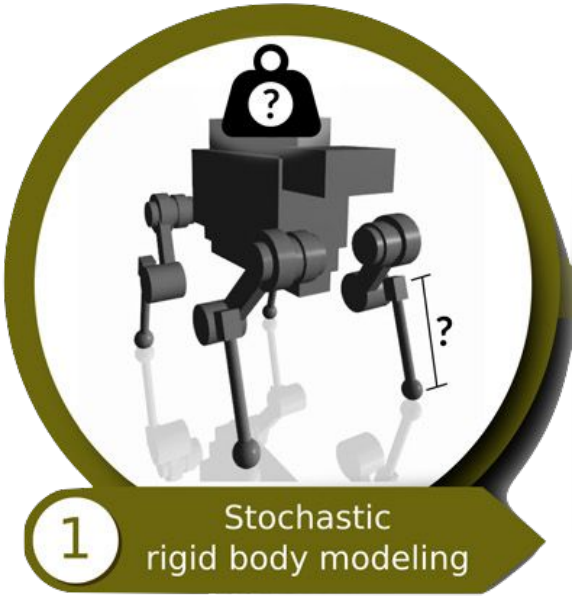
Improve simulation fidelity (analytically or data-driven)

Classical models representing well-known articulated system and contact dynamics + learning methods that can handle complex actuation

Make controller robust to differences in variations, allowing for better transfer.

RL policy trained with lots of data in simulation


?
?
1
Stochastic
rigid body modeling
AMYdrive
2
Train actuator net
with real data
State
Torque
Action
Policy
3
Reinforcement
learning in simulation
State
Action
AMYmal
4
Deploy on the
real system



We need to model the dynamics of intermittent contacts when walking

- Authors use a rigid-body contact solver from their previous work [41]

We need to model the inertial property of links

- Uses supplied CAD model
- Roughly 20% error in estimation due to unmodeled factors (ie: cables, electronics)
- 30 different models are used when training, with each model having stochastically sampled inertial properties.

We need to model the actuators

- Uses supervised learning to train an MLP that maps from actions to torques.
- Assumes the dynamics of the actuators are independent to each other, so a separate model is learned per actuator.
- Internal states of the actuators (e.g., states of the internal controllers and motor velocity) cannot be measured directly and must be inferred by the learned model.

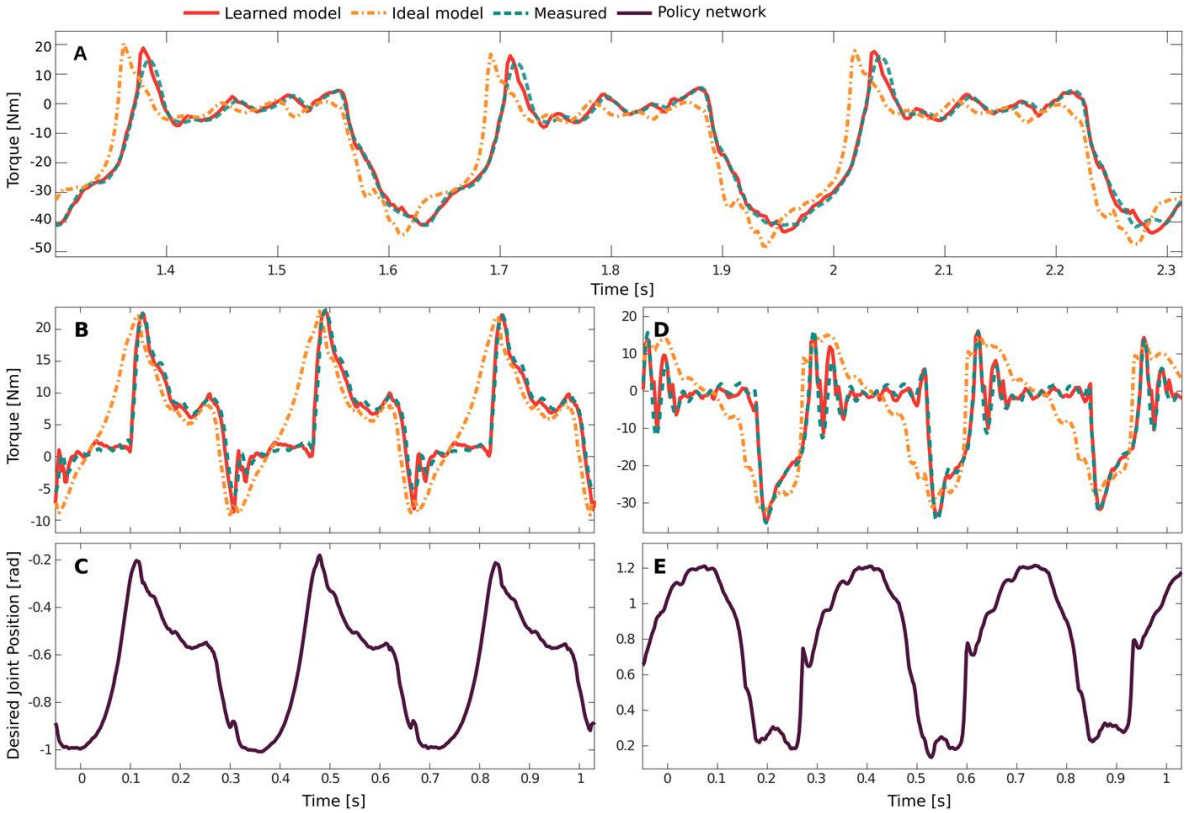
History of position errors (actual position - commanded position)

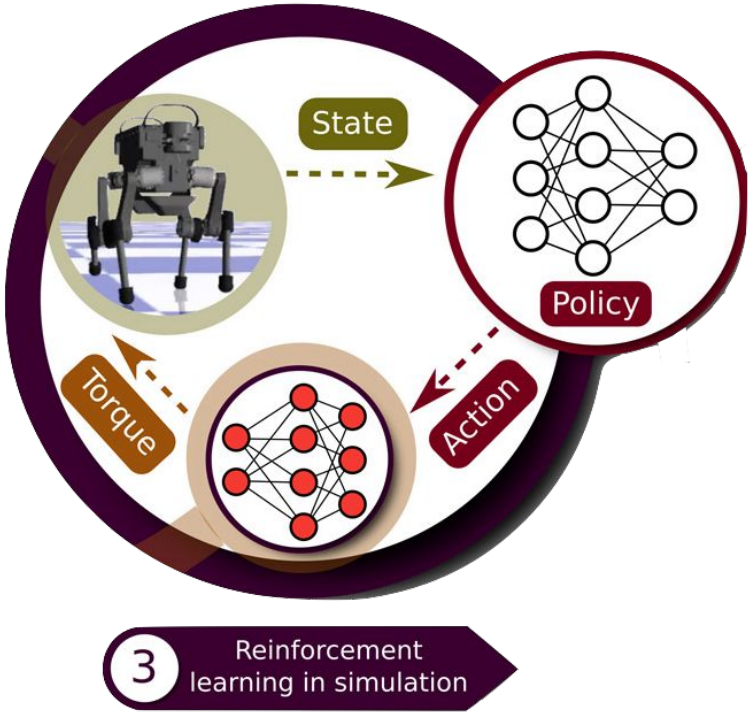
Joint velocity history

Actuator net

Estimated joint torques

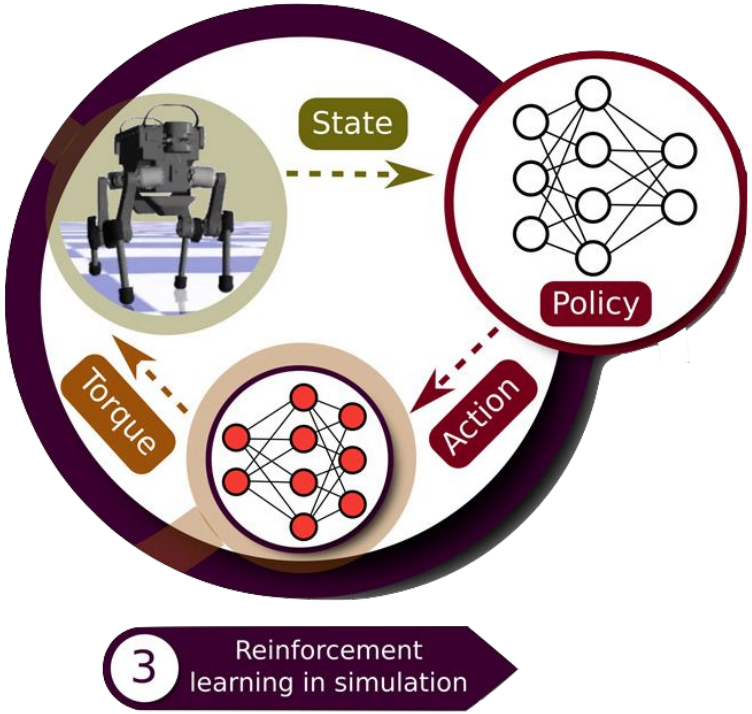

ANYdrive
2
Train actuator net
with real data


Learned model
Ideal model
Measured
Policy network
A
B
C
D
E
Torque [Nm]
Desired Joint Position [rad]
Time [s]



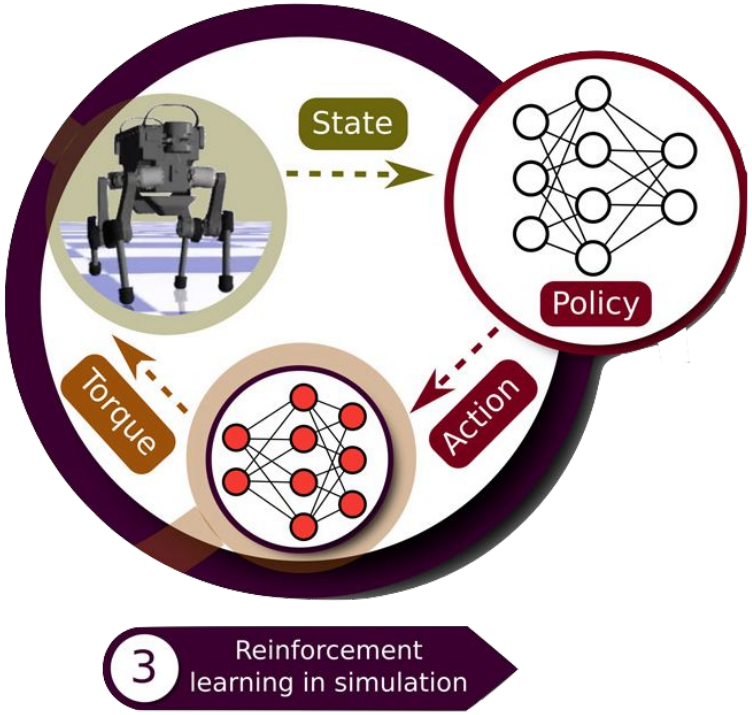
Inputted Observation:

- Current command
  - Forward/lateral velocity, turning rate
  - Can be specified by user via teleoperation
- Height, linear, and angular velocities of robot base
- Positions and velocities of robot joints
- Previous action
- Sparsely sampled joint state history.
  - Enables contact detection, avoiding the need for force sensors



Outputted Action:

- Joint position commands
- In simulation: fed into the actuator net to get resulting joint torques
- In real-world: fed directly to robot



RL algorithm: TRPO
Discount factor: (tuned to be) 0.9988 for walking/running and 0.993 for recovery
Initial state: sampled from previous trajectory or random distribution
Curriculum:

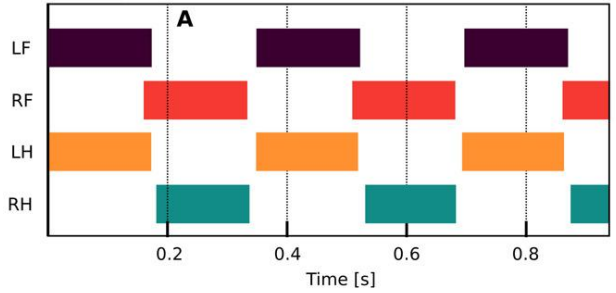
- A curriculum factor, k_c, is definition initially between 0 and 1. k_c increases monotonically and asymptotically to 1.
- All cost terms are multiplied by k_c, except those related to the objective.
- Encourages robot to first learn how to achieve the objective and then satisfy constraints.

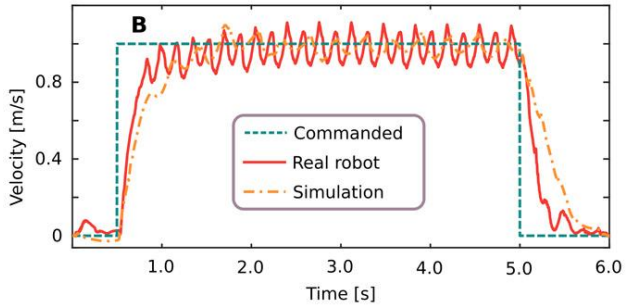
Results

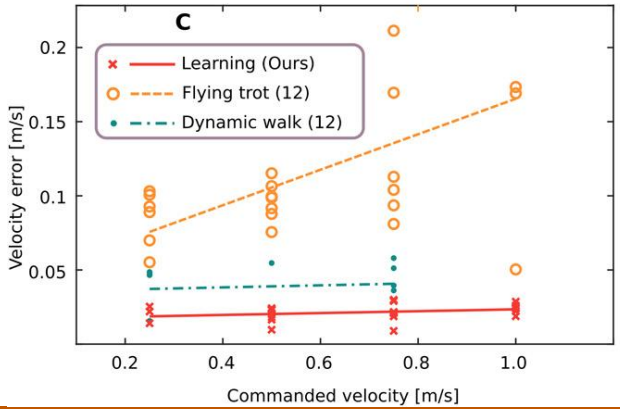
When only training for 4-11 hours on a personal laptop entirely in simulation, and then running the learned policy directly on the robot with no further modifications…

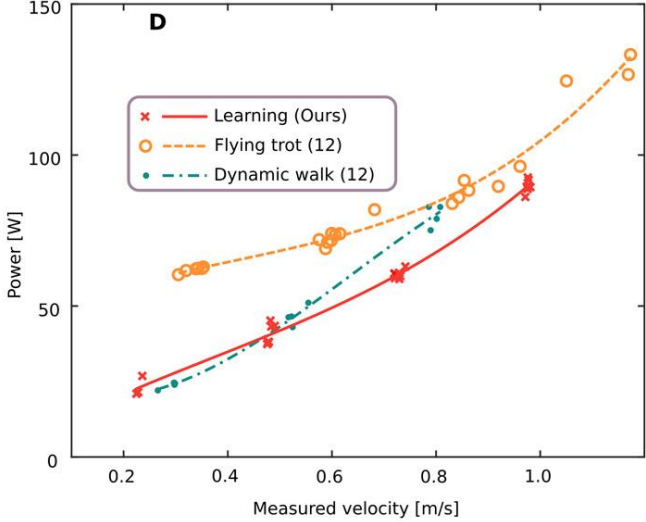
Results: Command Conditioned Locomotion

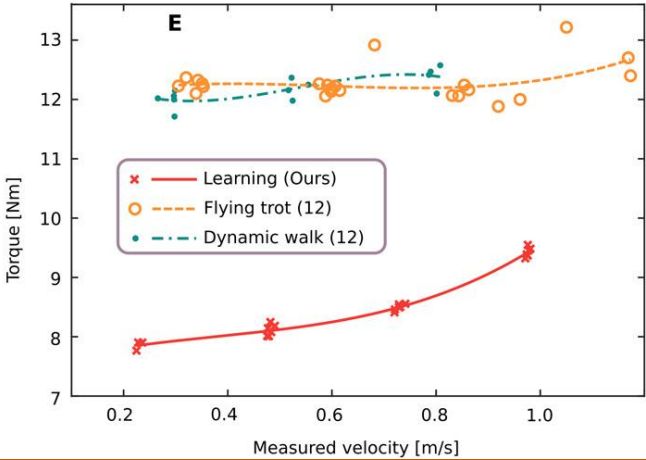
The motion of the robot is controlled by high level navigation commands.









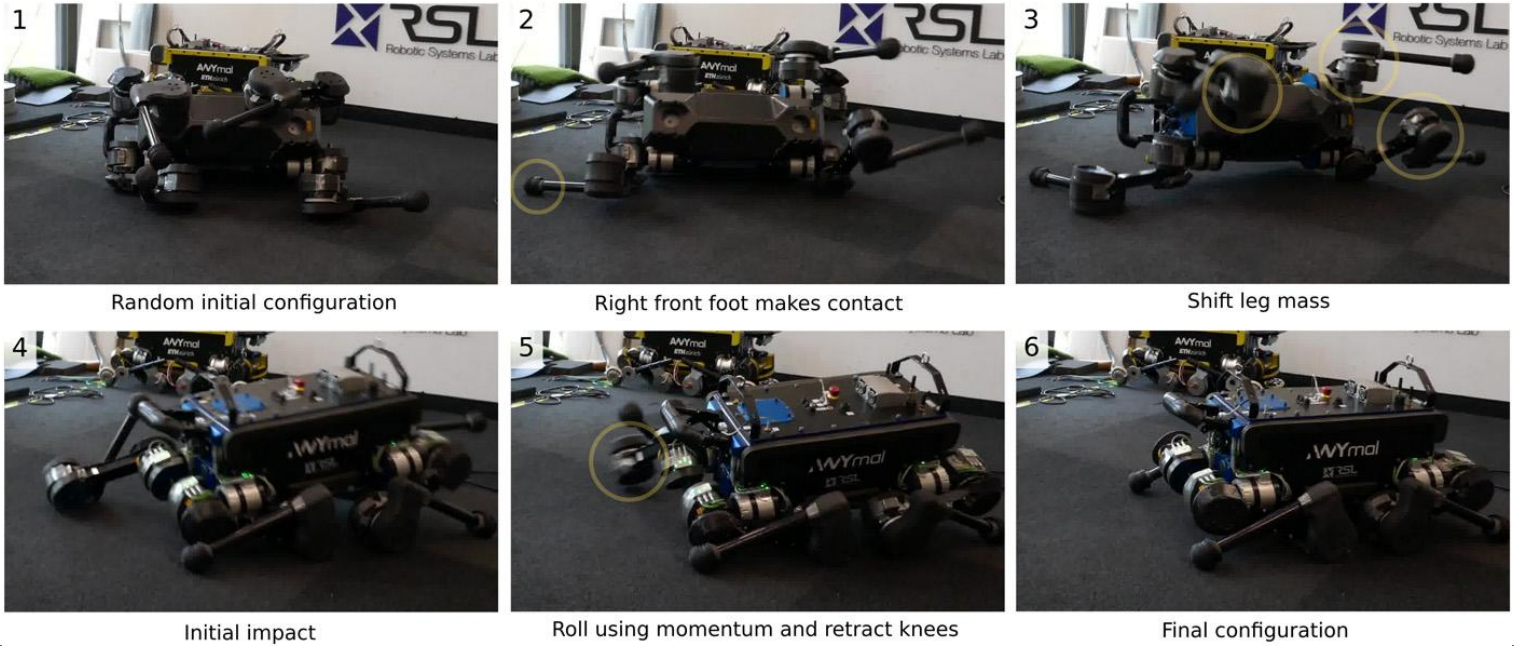


High speed locomotion

- Achieves 25% faster walk than previous fastest walk on the ANYmal robot.
- Shows that the learned policy can exploit the full capacity of the hardware, even when trained in simulation!

Recovery from fall

- The recovery controller successfully enables ANYmal to stand up from all tested fallen positions.



Highlights of this method:

- Achieves a high level of locomotion skill when trained entirely in simulation.
- The training procedure can be run in half a day on a personal computer.
- Does not require tedious tuning of parameters (that often takes months and many engineers) like all previous methods.
- Only requires a cost function, an initial state distribution, randomization parameters, and a CAD model.
- Learned policies are robust to changes in the hardware, such as through wear and tear or part replacements.
- Inference on the real robot takes ~25 μs.

Limitations of this method:

- A cost function and state distribution still need to be defined, which can be very tricky and require lots of trial and error.
- A CAD model that contains information needed for rigid-body modelling is still required. Such a CAD model may not exist for all robots.
- The learned actuator network cannot handle actuators that rely on coupled dynamics (ie: hydraulic actuators sharing a single accumulator).

Open questions and future work

- How can we more effectively specify cost functions for training different maneuvers? Can we learn such cost functions from human feedback?
- How good does simulation fidelity need to be, and how much of the remaining gap can be handled with a robust controller?
- Does having a simulator with very high fidelity impede the training of a robust controller? We do not want to overfit to the simulator.
- Can we iteratively improve simulation fidelity using collected real-world data?
- Training does not have to happen entirely in simulation. Can we use small amounts of policy rollouts in the real world to improve simulation fidelity or controller robustness?

Extended Readings

- Rethinking Sim2Real: Lower Fidelity Simulation Leads to Higher Sim2Real Transfer in Navigation
- Stochastic Grounded Action Transformation for Robot Learning in Simulation
- An Imitation from Observation Approach to Transfer Learning with Dynamics Mismatch
- Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization
- Domain randomization for transferring deep neural networks from simulation to the real world
- Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model

Motivation: Legged robots are attractive alternatives to tracked/wheeled robots for applications with rough terrain and complex cluttered environments, but learning good locomotion policies is very challenging.

Limitations of prior work: Most previous approaches require teams of engineers to spend months tuning modular controllers for each new maneuver. Data-driven approaches, on the other hand, are limited by the constraints of training in the real-world.

Problem: How can we train a locomotion policy using RL in simulation and then directly transfer this policy to the real robot?

Key insights: With a learned actuator network, a stochastic rigid body model of the robot, and a carefully designed RL training procedure, the authors are able to successfully learn numerous transferable maneuvers for the ANYmal robot.