Towards Dynamic Locomotion: A Blueprint for Accessible Sim-to-Real Bipedal Robotics

Kayden Knapik
Eindhoven University of Technology
Bachelor Mechanical Engineering
RBT Number: RBTBEP25
Student Number: 1776967
Email: k.a.knapik@student.tue.nl
Supervisors: René van de Molengraft, Danny Hameeteman & Ruben Beumer
Github Repository Link: https://github.com/KaydenKnapik/BDX-R

***Abstract*—Research in Reinforcement Learning (RL) for walking robots is often inaccessible due to high hardware costs and the steep learning curve of simulation to reality (sim-to-real) deployment. To make this research accessible, this paper provides a detailed blueprint for building a legged robot for RL. We demonstrate this blueprint on our low-cost bipedal robot, adapted from Disney's BD-X Bipedal Robot Platform [1]. We highlight every step of the complete workflow, from converting a Computer-Aided Design (CAD) model into a digital twin in the NVIDIA Isaac Sim environment, to training an RL policy with Isaac Lab. We validate this blueprint by successfully deploying an Inverse Kinematics (IK) policy and a standing policy on our developed platform. This policy can achieve stable, reactive balancing against external perturbations. By open-sourcing our work, we lower the entry barrier for research in dynamic legged robotics, providing a proven and accessible roadmap for students and researchers.**

I. INTRODUCTION

The ability for robots to walk and navigate complex environments represents a grand challenge in robotics. Recently, Reinforcement Learning (RL) for walking robots has become very popular as an alternative approach to typical model-based control, allowing robots to learn agile and robust locomotion policies directly from simulation in a matter of hours.

Despite this, for anyone looking to build a robot and deploy an RL policy, there is no clear start-to-finish plan to follow. Unfortunately, building such a robot is already a challenge in itself, and not having a single resource that one can follow creates a large barrier for anyone considering using RL.

For someone who wants to solely focus on training and deploying RL policies, they are limited by the accessibility and high hardware costs of walking robots. Many RL platforms are rather expensive, effectively restricting the field from students, hobbyists, and smaller research groups.

To address these challenges, we provide a complete blueprint from start to finish for building robots with RL. We prove the feasibility of this blueprint by building the BDX-Robstride (BDX-R), which can be seen in Figure 1, and is a low-cost, dynamic bipedal robot.

Fig. 1: Disney BD-X (Left) & Our Version (Right)

The main contributions of this work are:

- The design and construction of a low-cost, 10-degree-of-freedom (DOF) bipedal robot.
- A step-by-step documentation of preparing a Computer-Aided Design (CAD) model for the NVIDIA Isaac Sim environment to train RL policies using Isaac Lab.
- The open-sourcing of all hardware designs, software, and training configurations to serve as a resource for the community.

We begin by discussing related work that inspired the project, including important decisions towards the design and control of the robot. Next, we detail the design and hardware architecture of the BDX-R robot, highlighting the major differences between it and Disney's BD-X platform. Subsequently, we present the methodology for converting the CAD model into a digital twin and training the control policy using RL in Isaac Lab. The simulation-to-reality (sim-to-real) transfer is then described, where we present the results from deploying the policies on the physical hardware. Finally, we conclude with an analysis and discussion of our findings and outline directions for future work.

II. RELATED WORK

If one wants to build a legged robot, whether bipedal or quadrupedal, they must make important decisions to determine the hardware, the learning strategy, and the simulation environment. This section covers what is available in each of these areas and explains how we landed on the final choices for the BDX-R platform

A. Bipedal Hardware Platforms

Current bipedal robot platforms face two major challenges today: their high cost [2] and closed-source nature [3] make them inaccessible for most researchers. Research platforms like Boston Dynamics' Atlas [4] or Agility Robotics' Cassie [5] are great examples of this. While they show what is physically possible, they are extremely inaccessible for most students and hobbyists. Robot companies like Unitree have begun offering advanced robotics platforms like the G1 Humanoid [6]. Unitree offers a significant amount of open-source resources, including full-body datasets and the training framework. However, the education version of the robot still costs upwards of $40,000, still making it only accessible to those with a large budget.

We based our project on the design of Disney's BD-X robot. The BD-X was a great proof-of-concept, showing that a simple bipedal robot could still be highly dynamic and expressive. The original design used motors that were very difficult for individuals to buy at the time. Our solution was to adapt the design for more accessible actuators. This approach puts our work in line with other open-source projects that also use quasi-direct drive (QDD) motors to achieve a good trade-off between cost and performance.

B. Learning Approach: Reinforcement vs. Imitation

For decades, walking robots have relied on having a near-perfect mathematical model of the robot and its environment, making them extremely fragile. This is why the majority of the robotics field has shifted so heavily towards learning-based methods like RL and Imitation Learning (IL). Instead of being programmed with a fixed model, the robot learns a control policy through experience in simulation, making it far more robust to the unpredictable nature of the real world.

A major difference from the original BD-X is our learning approach. Disney used Imitation Learning (IL), creating reference motions for the robot to follow and rewarding the policy for tracking this prior data [1]. While this is a powerful technique when high-quality motion data is available, the lack of such data for our hardware led us to use a pure RL approach. Instead of tracking a predefined motion, our policy discovers how to walk from scratch, guided only by rewards that define the style of the motion. This RL-from-scratch method is often more straightforward for custom or novel robots, where generating realistic reference trajectories for imitation learning can be prohibitively time-consuming.

C. Simulation Environments for Sim-to-Real

With the learning strategy selected, the policy can be trained. This can either be done directly on the hardware or in simulation. Training directly on the hardware is impractical and dangerous due to the amount of time it would take to train a single policy, which would require an infeasible amount of real-world time and lead to significant wear and tear from countless falls. Therefore, training is almost exclusively done in a physics simulator. For years, the standard for RL research has been CPU-based simulators like MuJoCo [7], which is known for its speed and accuracy.

However, a newer generation of GPU-accelerated simulators has emerged, which can dramatically speed up training. For this work, we chose NVIDIA Isaac Lab [8], which is the newer version of Isaac Gym [9], the same RL software utilized by Disney to build their BD-X robot. The main benefit of Isaac Lab is massive parallelism. Isaac Lab has the ability to run thousands of simulated robots on a single GPU, gathering millions of data points in minutes, reducing overall training time from days to hours. Isaac Lab is now the standard for training many state-of-the-art robots, including the Unitree and ANYmal quadrupeds [10]. The problem that training in simulation creates is notably known in robotics as the sim-to-real gap. To allow for the sim-to-real gap to be reduced, we rely heavily on a common RL technique called domain randomization [11]. This involves constantly changing the physics in the simulation—like the robot's mass, motor strength, and ground friction, which results in the policy having to learn a more robust strategy that accounts for all kinds of randomization that is not tuned to one perfect set of conditions.

D. The Gap in End-to-End Replication

One other major challenge when it comes to building legged robotics with RL is the knowledge gap. Resources that can be used to guide someone from starting with a CAD file to simulation setup, policy training, and real-world deployment are very limited. This absence means that many of the crucial implementation details required for locomotion are never formally documented. Our project directly addresses this issue by presenting not just a robot, but a complete methodology, aiming to lower the barrier to entry for future work in dynamic bipedal locomotion by highlighting the complete process, detailed in Figure 2.

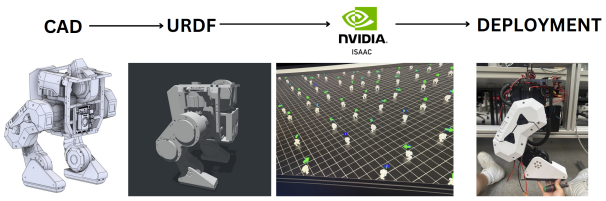


Fig. 2: Workflow to go from CAD to deploying an RL policy on physical hardware

III. METHODOLOGY

A. THE BDX-R ROBOT PLATFORM

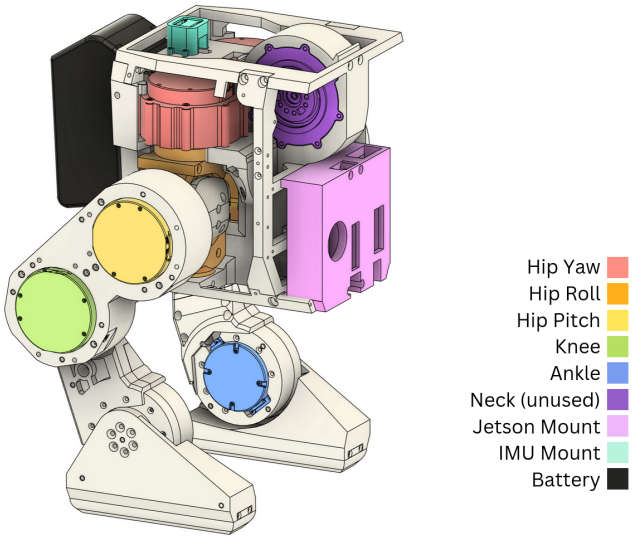


Fig. 3: CAD model highlighting different components and their functions

The BDX-R, highlighted in Figure 3, is a 10-DOF bipedal robot that serves as the hardware platform for this research. The design is an adaptation of the Disney BD-X robot without the neck and head assembly, due to the focus of this paper on investigating standing and walking policies.

1) Actuator Choice: One of the first choices when designing a robot is deciding on the physical hardware to use. The original Disney BD-X utilizes Unitree motors, which, at the time, were rather inaccessible and unaffordable, and thus, cheaper alternatives were needed.

We ultimately decided on using Robstride QDD actuators to serve as the robot's joints, a choice also featured in other open-source humanoid projects like the K-bot. These actuators are both affordable and have higher torques than the motors used by the Disney research team. The motor parameters are highlighted in Table III.

- **Robstride O3 actuators** are used for the hip roll, pitch, yaw, and knee joints. These joints bear the highest loads and require significant torque.
- **Robstride O2 actuators** are used for the ankle joints, where lower torque is sufficient.

2) Onboard Computing and Communication: A Jetson Orin Nano Super was chosen for the computer for this robot, which is responsible for running the high-level control policy and communicating with the actuators. The communication architecture is based on a CAN bus protocol. Each leg's actuators are connected on a separate CAN bus, which are connected to the Jetson with two USB-to-CAN debuggers. Each leg consists of a daisy chain of 5 motors, from the hip yaw all the way down to the ankle motor, which reduces the need for excessive wiring.

3) Sensing: For state estimation, we use an Adafruit BNO055 Inertial Measurement Unit (IMU) to track the robot's orientation. This sensor was chosen for its low cost and accessibility, aligning with the project's goals, and is mounted centrally on the upper chassis. It provides the control policy with essential observation data—projected gravity and angular velocity—by communicating with the Jetson over the I2C protocol.

4) Power System: The entire system is powered by a commercial off-the-shelf 40V, 5.0 Ah, 200Wh power tool battery, aligning with the project's low-cost objective. The nominal 40V output directly powers the Robstride actuators, while a step-down voltage regulator provides the necessary lower voltage for the Jetson Orin Nano.

5) Motor Control Scheme: The Robstride actuators are managed by a low-level controller implementing a versatile "operation control mode," sometimes called MIT control [12]. This scheme provides precise torque regulation by combining feedforward commands with Proportional-Derivative (PD) feedback, as shown in Figure 4. The comprehensive control law, which can use five input parameters, is:

$$t_{\text{ref}} = K_p(p_{\text{set}} - p_{\text{actual}}) + K_d(v_{\text{set}} - v_{\text{actual}}) + t_{\text{ff}}$$

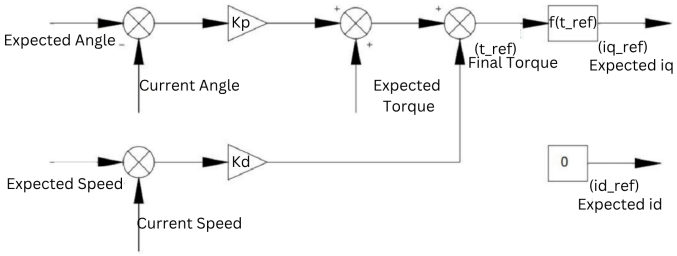


Fig. 4: Operation Control Mode block diagram, illustrating how expected angle, speed, and torque are combined to produce the final motor command.

Simplification for RL: For our RL application, the high-level policy is designed to output only pure position targets. This choice is important because it removes the complicated, low-level motor dynamics, allowing the RL agent to operate in a simpler action space focused on strategic goals (i.e., "where should the joint be?"). This division is essential for efficient learning and robust sim-to-real transfer. As a result, the control interface is simplified by setting the feedforward torque and target velocity to zero ($t_{\text{ff}} = 0$, $v_{\text{set}} = 0$).

This simplification yields a direct PD control law focused exclusively on position tracking and active damping. The governing formula for our implementation becomes:

$$t_{\text{ref}} = K_p(p_{\text{set}} - p_{\text{actual}}) - K_d \cdot v_{\text{actual}}$$

The two components of this simplified controller serve distinct roles:

- **Proportional Term** ($K_p(p_{\text{set}} - p_{\text{actual}})$): This term drives the motor toward the target position commanded by the

RL policy. The torque is proportional to the position error, meaning a larger error results in a stronger corrective force. The gain K_p determines the "stiffness" of the joint.
- **Derivative Term** ($-K_d \cdot v_{\textbf{actual}}$): This term provides active damping by generating a torque that opposes the motor's current velocity. It is critical for preventing overshoot and oscillations, ensuring smooth and stable motion. The gain K_d dictates the magnitude of this damping force.

While this formula is structured for position tracking, its application in legged robotics with low gains requires a more nuanced interpretation. In this context, the controller functions as an impedance controller, where the primary goal is not to eliminate position error, but to use that error to generate a desired torque. The position setpoint acts as a virtual "anchor" point, and the gains act as a virtual "spring stiffness." This approach allows the robot to be compliant with its environment, absorbing impacts and handling unexpected contacts. As one key study notes, "At low gains, these setpoints are not intended as desired position targets; rather, they serve as intermediate variables to generate desired torques" [13]. Therefore, a steady error between the target and actual joint positions is not only expected but is a clear indicator that the controller is working as intended.

Although the Robstride actuators have absolute encoders, a stand was needed in order to hold the robot while it was not powered on. This stand was also responsible for setting the motor's zero position, and was done on every startup, so that every time the robot was perfectly in the zero position, which is the same as in the simulation before a policy is deployed. This stand idea is also utilized by Disney and other robots that must be zeroed, as shown in Figure 5.

Fig. 5: Image Highlighting the Test Setup, including the stand for the robot (left) and the hook used to attach to the tether.

B. DIGITAL TWIN CREATION

Once all the necessary hardware components were chosen, the CAD model could be created. Due to the high popularity of the Disney BD-X droids, files had already been created for the version with the Unitree motors [14]. This model was modified to fit Robstride actuators and a different battery. Once the CAD model was created, it was then converted to a Unified Robot Description Format (URDF) using a conversion script. The URDF describes all of the joints and inertia for the robot.

The URDF can be directly imported into NVIDIA Isaac Sim where it will automatically be converted into a Universal Scene Description (USD) file. We then create an Articulation Configuration (ArticulationCfg) file for the robot. This file describes the actuator properties that will be used for training. We make use of the Implicit Actuator model provided by Isaac Lab, which directly simulates a low-level PD controller for each joint. A critical step in bridging the sim-to-real gap is to make sure that the simulated actuators behave like the physical actuators. To achieve this, the proportional (K_p) and derivative (K_d) gains were tuned empirically on the physical robot to find a desired balance between responsiveness and stability. We also set the armature, friction loss, and maximum joint velocity. Each of these values was specified in the Robstride motor datasheets and can be shown in Table III.

C. SIMULATION-BASED CONTROL

We use NVIDIA's Isaac Lab to train the RL policy that is later deployed on the physical robot. Isaac Lab was chosen due to its fast training time.

For the RL algorithm itself, we use the Robotic Systems Lab RL (RSL-RL) [15] framework developed at ETH Zürich. RSL-RL was chosen due to its easy accessibility in Isaac Lab and the fact that it is specifically designed for legged locomotion. It provides a stable and well-tested implementation of the Proximal Policy Optimization (PPO) algorithm, which is the standard for this type of research. PPO was chosen because it is a policy-gradient method that balances learning efficiency and stability by updating the policy in small, constrained steps, making it particularly effective for continuous control tasks like locomotion [16].

The policy needs information from the environment to make effective decisions. In RL, this information is referred to as the observation and forms the input to the policy network. For our robot, the observation is represented as a 39-dimensional vector, detailed in Table I. A key principle when designing this vector was sim-to-real feasibility: every quantity provided in simulation must also be measurable reliably on the physical robot.

From the onboard IMU, we only use the projected gravity and the robot's angular velocity. This is a common approach in RL robots that use IMU observations due to the extremely noisy linear acceleration data that IMUs can produce while walking.

The robot's internal configuration is described by its joint positions and velocities. This data comes directly from the

TABLE I: Robot Observation Vector

Observation Component	Description	Dimension
Projected Gravity	Gravity Vector (x, y, z)	3
Angular Velocity	Body angular velocity (x, y, z)	3
Commanded Velocity	Desired linear (x, y) and angular (z) velocity.	3
Joint Positions	Angular position of each of the 10 joints.	10
Joint Velocities	Angular velocity of each of the 10 joints.	10
Last Actions	The previously commanded action for each of the 10 joints.	10
Total Dimension		**39**

motor encoders, giving the policy an accurate understanding of its own body posture.

The commanded velocity is the goal that the reward function will be based on.

Finally, we include the action that the policy took in the previous timestep. This helps the policy learn to generate smoother motor commands, as it can see the command it just sent. This helps it understand joint acceleration without needing to rely on noisy sensor data.

With a combination of these observations, the robot can get a good understanding of its state, which is crucial for the RL policy. Once the observation vector has been chosen, the reward shaping process can begin.

The reward function is the most important aspect of an RL policy, as it is solely responsible for shaping the robot's behavior and ensuring that it is feasible for deployment onto physical hardware. We have broken it down into three main components: task-specific rewards, regularization penalties, and a termination penalty. The final weights, presented in Table II, are the result of utilizing basic rewards from similar legged robotics while also combining stylistic rewards to achieve a better-looking walk.

The task-specific rewards are the most important drivers of the desired behavior. The Track Linear Vel. and Track Angular Vel. terms are heavily weighted (5.0) to incentivize the robot to closely follow the policy commands, as this is its main objective. The Air Time reward encourages large steps by rewarding the duration a foot is in the air, while the Foot Clearance was added so that the robot lifts its feet high enough to achieve a stylistic walking gait.

The regularization penalties were added so that the actuators perform in a way that is physically feasible on the real robot. This is why these penalties are crucial for successful sim-to-real transfer.

Stability: The Flat Orientation and Base Height penalties are extremely important, as the robot would tend to learn to walk with its base at a very low height due to the perturbations. These penalties discourage this by aiming to keep the base at a set, stable height.

Smoothness and Efficiency: The Action Rate, DoF Acceleration, and DoF Torques penalties are small but vital. They smooth out the robot's movements by penalizing jerky actions and high motor torques. This not only leads to a more natural-looking gait, but it also prevents the policy from learning actions that are overly aggressive and may cause damage to the physical hardware.

Posture and Safety: Penalties like Hip Deviation, Joint Position, and Ankle Limits encourage the robot to maintain a neutral posture and stay within its physical joint limits, preventing awkward or damaging configurations. The Foot Slip penalty was significant for preventing the robot from learning to shuffle its feet in the simulator.

Once a walking motion was achieved in simulation, stylistic rewards were then added on top to create a more stylized walking gait. We noticed that once the robot achieved a basic, stable gait, it had a tendency to "walk in place" when commanded to stand still (a commanded velocity of zero). This is extremely common in many bipedal robots and is a direct consequence of the Air Time reward, which rewards lifting the feet even when stationary. To address this, a specific penalty was introduced using the Joint Pos. reward. This penalty is seen in Table II When the commanded velocity is close to zero, the robot's joints are penalized for movement that deviates from the standing position. The clearance, foot slip, and joint position penalty previously described were all utilized in the Boston Dynamics Spot Sim-To-Real [17] paper provided by Isaac Lab; they were simplified and modified from a quadrupedal reward to a bipedal reward. This two-stage approach—first achieving the core task, then refining the style—was crucial for developing a policy that behaves appropriately across different command scenarios.

Finally, the Termination Penalty provides a large negative reward (-200.0) when the robot makes contact with the ground. This helps the policy learn to stand rather quickly, as it wants to avoid unstable states at all costs. The final set of weights represents a carefully tuned balance between accomplishing locomotion while also doing so in a stylistic way, similar to the way that Disney's BD-X robot walks.

TABLE II: Weighted Reward Terms

Name	Reward Term	Weight
	Task-Specific Rewards	
Track Lin. Vel.	$\exp(-\alpha\|\|\mathbf{v}_{xy} - \hat{\mathbf{v}}_{xy}\|\|^2)$	5.0
Track Ang. Vel.	$\exp(-\beta\|\|\omega_z - \hat{\omega}_z\|\|^2)$	5.0
Air Time	$R_{\text{gait}} \propto \sum_i \min(\max(t_{\text{air},i}, t_{\text{contact},i}), t_{\text{mode}})$	5.0
Foot Clearance	$\exp\left(-\frac{1}{\sigma}\sum_i (z_i - z_{\text{tgt}})^2 \tanh(\gamma\|\|\mathbf{v}_{xy,i}\|\|)\right)$	2.0
	Regularization Penalties	
Flat Orientation	$-\|\|\theta_{xy}\|\|^2$	-2.0
Action Rate	$-\|\|\mathbf{a}_t - \mathbf{a}_{t-1}\|\|^2$	-0.05
DoF Acceleration	$-\|\|\ddot{\mathbf{q}}\|\|^2$	-1.25e-7
DoF Torques	$-\|\|\tau\|\|^2$	-5.0e-6
Hip Deviation	$-\|\|\mathbf{q}_{hip} - \hat{\mathbf{q}}_{\text{def}}\|\|$	-1.0
Ankle Limits	Penalty for ankle joints near limits	-1.0
Joint Pos.	$-\|\|\mathbf{q} - \mathbf{q}_{\text{def}}\|\|^2$	-1.0
Base Height	$-\|\|p_z - \hat{p}_z\|\|^2$	-2.0
Foot Slip	$\sum_i \mathbb{I}(f_{z,i} > f_{\text{thr}}) \cdot \|\|\mathbf{v}_{xy,i}\|\|$	-0.5
	Termination Penalties	
Termination	Large penalty on termination	-200.0

Currently, the robot is trained only to move forward to a maximum velocity of 0.7 m/s. The robot is also trained with curriculum learning, so after 1500 training steps, the robot experiences external perturbations. This allows the robot to have time to learn to stand and walk first before it can experience forced pushes, which results in a more stylized walk.

We utilize domain randomization, as does every other RL study, as this significantly reduces the sim-to-real gap, as the policy can learn to adapt to imperfections. The parameters that are randomized can be shown in Table IV.

D. SIM-TO-REAL TRANSFER PROCEDURE

The last stage of the methodology regards the sim-to-real transfer, where the RL policy that was trained in simulation gets deployed on the physical hardware. When the policy finishes training, it gets exported as a PyTorch file, which contains the neural network's architecture and learned weights. This essentially functions as the robot's brain, which directly maps the observations to actions, which are the motor commands.

We created a deployment script on the robot, which executes a high-frequency control loop at 50 Hz to match the frequency in the simulator. Therefore, every 0.02 seconds, the data from the IMU and motor encoders are passed into the 39-dimensional observation vector (Table I), which is then fed into the policy network, which computes the target joint positions. These target joint positions are the "actions" that are commanded to the low-level PD controllers, enabling the robot to react dynamically to its state.

Deploying a walking policy directly onto hardware will often be unstable and unsafe. Before deploying, the observation order, joint directions, and IMU orientation all had to be checked to make sure they were consistent in both simulation and in reality. A good way to do this was to first deploy a simpler Inverse Kinematics (IK) policy trained only to move a single leg [18]. This step also reduces the risk of damage as it is much safer than while deploying the walking policy, as the robot can be elevated and mounted to a platform.

When deploying the walking policy, there were significant vibrations. This happened because the policy, optimized in simulation, generated high-frequency, noisy actions. Therefore, a low-pass filter was included on the joint velocities and the actions due to them being extremely noisy. Specifically, we added a first-order Exponential Moving Average (EMA) filter:

$$y_t = \alpha \cdot x_t + (1 - \alpha) \cdot y_{t-1} \tag{1}$$

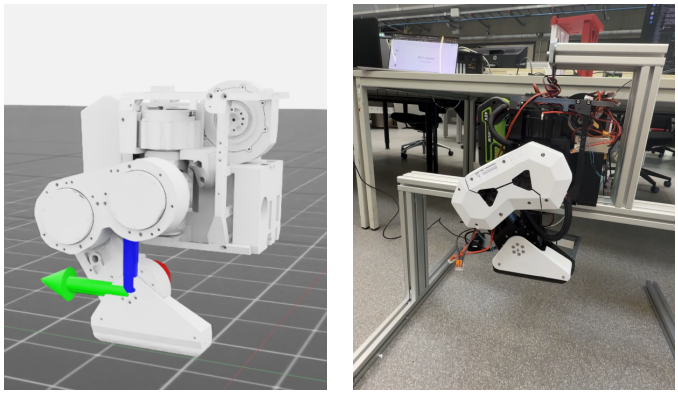
where y_t is the current filtered value, x_t is the new raw value, and α is the smoothing factor. Applying this filter to actions was particularly crucial. The α value was tuned manually, with a value of 0.1 in order to smooth out jerky motions while still maintaining responsiveness.

IV. RESULTS

This section presents the results of the sim-to-real transfer, highlighting both the initial single-leg IK policy deployment and the final standing policy.

A. Single-Leg Policy Validation

The RL IK policy was successfully deployed on the physical robot without the need for the EMA filter previously mentioned, as the outputs were already smooth. The simulation can be compared with the physical setup as shown in Figure 6. This successful deployment confirmed that the physical actuators could accurately follow the policy's position commands.



(a) Policy deployed in simulation.

(b) Policy deployed on the real robot.

Fig. 6: Sim-to-Real transfer of the IK RL policy for a single leg. The policy, trained in simulation (a), is successfully deployed on the physical hardware (b) without further training.

In order to visualize performance, the live motor positions were graphed against the target positions. The results demonstrate the effectiveness of the learned policy. For example, the hip pitch actuator displays strong tracking performance in both simulation and reality, shown in Figure 7 and Figure 8. This data is also representative of the yaw, knee, and ankle actuators, and therefore, the data was omitted due to redundancy.

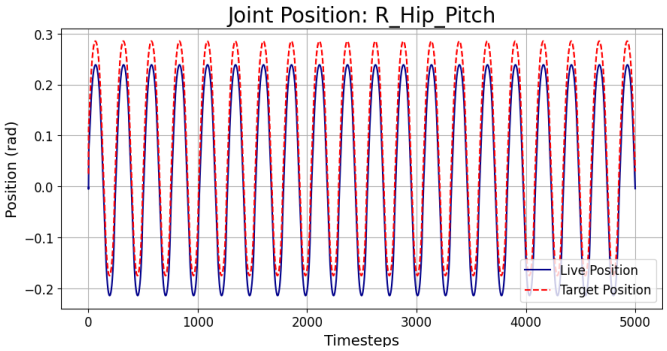


Fig. 7: Target vs. Actual Hip Pitch Position in Simulation.

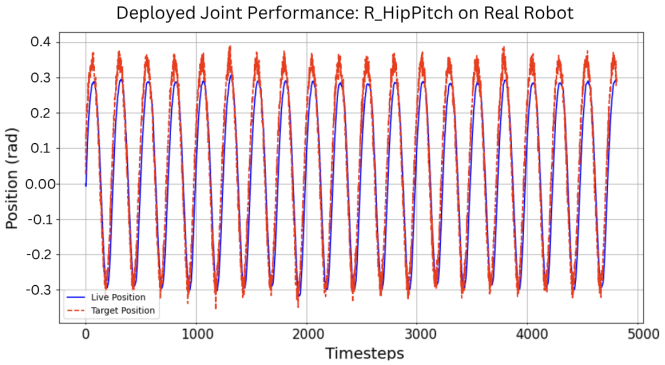


Fig. 8: Target vs. Actual Hip Pitch Position on the Real Robot.

In contrast, the hip roll data, appearing in Figure 11, highlights a different but equally important aspect of the controller's success. This graph shows a persistent but stable offset between the target and actual positions, also highlighted in simulation in Figure 10. This offset is not a sign of failure but is precisely the expected behavior of the low-gain PD controller operating as an impedance controller, as discussed in the methodology.

B. Standing & Walking Policy Deployments

Once the observation order, joint directions, and IMU orientation were all consistent with the simulator, the full walking policy was deployed with a commanded velocity of zero. This results in a standing policy, and this deployment was also successful and can be seen in Figure 9. The robot was able to stand with the low gains and recover from perturbations, due to its training on similar perturbations. When pushed gently by hand, the robot can balance and return to its standing position.

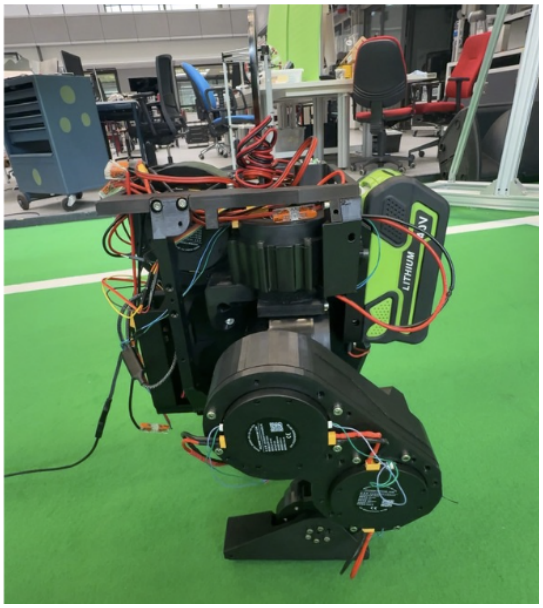


Fig. 9: Deployment of the standing policy on the physical BDX-R.

The stability of the robot is also demonstrated by the projected gravity data in Figure 13 and the angular velocity data inFigure 12. The projected gravity displays the gravity only being in the negative z direction until the robot is acted upon by a user. This is consistent with the body's angular velocity remaining centered around zero, which is direct evidence of a stable stance. The policy's ability to hold the robot's posture is shown in Figure 14, which displays the tracking performance for a representative knee joint. The perturbations are visible in the graph, with the largest occurring around timestep 200,000. This event leads to a fall, which is clearly evident in both the projected gravity and angular velocity data.

The final experiment was to incrementally increase the commanded forward velocity. During these trials, the robot exhibited stepping behaviors. However, it was ultimately unable to achieve a stable, forward-progressing gait and would lose its balance after a step or two. The successful deployment of a dynamic walking gait was not achieved within the time frame of this project.

V. DISCUSSION

While we were able to successfully deploy two policies, an IK policy and a standing policy, we were ultimately unsuccessful in deploying a stable walking policy. In this discussion, we highlight why our blueprint succeeded for both IK and standing policies, yet failed for the walking policy.

The success of both policies shows that our core methodology is sound. Our step-by-step deployment strategy, which started with a simple single-leg test, was crucial. It gave us a safe way to confirm that the software and hardware were communicating correctly, and it was during this phase that we tuned the PD gains. When deploying the standing policy, we discovered the need for a low-pass filter. That filter was essential because RL policies trained in a perfect simulator often produce noisy, high-frequency commands that a real robot cannot physically follow. The filter smoothed these commands out, making them mechanically possible.

The inability to achieve walking locomotion within the time span of this project highlights the immense difficulty of dynamic locomotion compared to the simpler task of standing still.

We believe the failure comes down to three main factors. The first is that the sim-to-real gap becomes much worse during dynamic movement. While our digital twin was good enough for balancing, walking magnifies the small differences between the model and reality, such as motor backlash, minor flexing in the chassis, or complex friction. During the rapid weight shifts of a single step, these tiny, unmodeled effects can quickly add up and lead to a fall that the policy, trained in a perfect world, could not predict or prevent.

Secondly, the walking policy was only trained for 50,000 steps, and while this may sound like a lot, it is likely not nearly enough for a policy to learn a stable gait. The behavior we saw during deployment supports this, as the robot learned the basic motion of taking a step, but it does not have the balance or recovery to remain stable. In legged robotics, it is common for

policies to need hundreds of thousands of iterations to master skills like walking.

Finally, our reward function might have been part of the problem. Specifically, the regularization penalties may not be large enough. For instance, there should not be a need to add a low-pass filter to the policy actions, as this can be handled by adding a larger penalty for the action rate. This low-pass filter may also be hindering the stable walking, as the policy was not retrained with the filter in mind, so it may be filtering out the necessary commands that may cause the robot to walk.

Both the PD gains and the low-pass filter's alpha value were tuned by hand, based on the most effective settings in practice. A more formal system identification process would likely lead to a more accurate simulation and better performance.

While bipedal locomotion was unfortunately not accomplished, this limitation does not take away from our main accomplishment. Our main accomplishment was creating a complete blueprint that one can follow to go from CAD design to both an IK and a standing RL policy. We built the platform, validated the simulation environment, and showed that the sim-to-real transfer is viable. This provides a solid foundation, and the next logical step is to build upon it with more extensive training and a better model of the system's dynamics.

VI. CONCLUSION AND FUTURE WORK

This work provides a complete blueprint for the entire bipedal robotics pipeline, successfully demonstrating the workflow from initial design to final policy deployment. The primary success lies in the creation of a fully functional sim-to-real framework, validated by the deployment of a policy that enabled the physical BDX-R robot to perform IK tasks, maintain a stable stand, and robustly recover from significant external perturbations. The inability to produce a stable walk with RL within the time span of this project highlights the extreme difficulty of bridging the sim-to-real gap for dynamic locomotion. To reduce the sim-to-real gap, a more accurate model of the actuators must be made.

By creating and documenting a complete and accessible workflow from the selection of low-cost actuators to the creation of a digital twin and the implementation of a robust deployment strategy, we have created a validated foundation for future research in this area. This project effectively lowers the barrier to entry, enabling others to build upon this platform or platforms of their own to explore the next frontier: achieving truly stable and dynamic bipedal locomotion.

The results suggest that achieving stable locomotion is not a question of if, but when, given further iteration. The immediate future steps are clear:

Prolonged and Scaled Training: The most critical next step is to increase the training duration (e.g., to 100,000 or 200,000+ steps) to allow the policy to converge fully. This improvement was noticed significantly when going from 20,000 to 50,000 steps. Better Actuator Models: In order to reduce the sim-to-real gap, a more accurate model of the actuators must be made. This can be done by performing system identification to more accurately model dynamic effects like joint friction and armature for specific joints. Systematic Reward Shaping: A thorough analysis and tuning of the reward function are necessary, potentially increasing penalties for the action rate and other instability metrics. This should help smooth out the policy on its own, potentially eliminating the need for a low-pass filter.

REFERENCES

[1] R. Grandia, E. Knoop, M. A. Hopkins, and et al., "Design and control of a bipedal robotic character," in *ACM SIGGRAPH 2023 Conference Papers (SIGGRAPH '23)*. New York, NY, USA: Association for Computing Machinery, aug 2023, pp. 1–13.

[2] Generation Robots, "Decoding the cost of robots: Between advanced research, cutting-edge equipment, and manufacturing challenges," oct 2023, accessed: 2025-08-25. [Online]. Available: https://www.generationrobots.com/blog/en/decoding-the-cost-of-robots-between-advanced-research-cutting-edge-equipment-and-manufacturing-challenges/

[3] S. Maximilian, "Hugging face warns that closed-source robots threaten user control," The Decoder, mar 2024. [Online]. Available: https://the-decoder.com/hugging-face-warns-that-closed-source-robots-threaten-user-control/

[4] Boston Dynamics, "Atlas," https://bostondynamics.com/atlas/, 2025.

[5] Agility Robotics, "Cassie bipedal robot," https://www.agilityrobotics.com/, 2025.

[6] Unitree Robotics, "Unitree g1 humanoid robot," https://www.unitree.com/g1/, 2025.

[7] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[8] V. Varadarajan, J. Liang, A. Handa, and G. State, "Isaac lab: Nvidia's open-source simulation environment for robot learning," NVIDIA Technical Blog, Oct 2023. [Online]. Available: https://github.com/isaac-sim/IsaacLab

[9] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, R. Grandia, D. Hoeller, G. Ginting, M. Hutter, and G. State, "Isaac gym: High-performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[10] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proceedings of the 5th Conference on Robot Learning (CoRL)*, nov 2021. [Online]. Available: https://arxiv.org/abs/2109.11978

[11] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[12] B. Katz, "A simplified approach to high-performance brushless motor control for robotics," M.S. Thesis, Massachusetts Institute of Technology, 2021.

[13] Q. Liao, T. E. Truong, X. Huang, G. Tevet, K. Sreenath, and C. K. Liu, "Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion," *arXiv preprint arXiv:2508.08241*, aug 2025. [Online]. Available: https://arxiv.org/abs/2508.08241

[14] M. Reynisson, "Adverserial waddle dynamics (awd)," https://github.com/rimim/AWD, 2023.

[15] ETH Zurich, Robotic Systems Lab, "Rsl-rl: Reinforcement learning framework for legged robots," https://github.com/leggedrobotics/rsl_rl, 2022.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] O. Omotuyi, D. Hoeller, and T. Burnham, "Closing the sim-to-real gap: Training spot quadruped locomotion with nvidia isaac lab," Jun 2024. [Online]. Available: https://developer.nvidia.com/blog/closing-the-sim-to-real-gap-training-spot-quadruped-locomotion-with-nvidia-isaac-lab/

[18] L. Le Lay, "Kinova Gen3 RL & Sim2Real Toolkit," jul 2025. [Online]. Available: https://github.com/louislelay/kinova_isaaclab_sim2real

APPENDIX A
HARDWARE

TABLE III: Robstride Motor Parameters

Parameter	Robstride O3	Robstride O2	Units
Max Torque (τ_{max})	60.00	17.00	N m
Max Velocity	18.85	37.70	rad s^{-1}
Weight	880.0	405.0	g
Armature	0.02	0.0042	kg m^2
Friction Loss (μ_s)	0.20	0.10	N m
Proportional Gain (k_P)	10.00	10.00	N m rad^{-1}
Derivative Gain (k_D)	0.30	0.30	N m s rad^{-1}

TABLE IV: Domain Randomization Parameters

Parameter	Range	Units
Physics Randomization (on Reset)		
Base Mass	Additive: [-0.5, 0.5]	kg
Static Friction (μ_s)	[0.1, 2.0]	-
Dynamic Friction (μ_d)	[0.1, 2.0]	-
Actuator Stiffness	Scale: [0.6, 1.4]	× default
Actuator Damping	Scale: [0.6, 1.4]	× default
Joint Friction	Scale: [0.8, 1.2]	× default
Joint Armature	Scale: [0.8, 1.2]	× default
IMU Position Offset (x,y,z)	[-0.05, 0.05]	m
IMU Orientation Offset (R,P,Y)	[-0.1, 0.1]	rad
Observation Noise (Uniform)		
Projected Gravity	[-0.1, 0.1]	m/s^2
Angular Velocity	[-0.1, 0.1]	rad/s
Joint Positions	[-0.05, 0.05]	rad
Joint Velocities	[-0.5, 0.5]	rad/s
Initial State Randomization (on Reset)		
Base Position (x, y)	[-0.5, 0.5]	m
Base Yaw	[$-\pi$, π]	rad
Joint Positions	Scale: [0.8, 1.2]	× default

TABLE V: PPO Hyperparameters for BDX-R Training

Hyperparameter	Value
Runner Configuration	
Steps per Environment	24
Total Training Iterations	50000
Empirical Normalization	False
Policy Network (Actor-Critic)	
Actor Hidden Dimensions	[256, 128, 128]
Critic Hidden Dimensions	[256, 128, 128]
Activation Function	ELU
Initial Action Noise Standard Deviation	1.0
PPO Algorithm	
Learning Rate	1.0e-3
Learning Rate Schedule	Adaptive
Desired KL Divergence (for adaptive LR)	0.01
Discount Factor (γ)	0.99
GAE Parameter (λ)	0.95
Learning Epochs per Iteration	5
Mini-batches per Epoch	4
Clipping Parameter (ϵ)	0.2
Value Function Loss Coefficient	1.0
Entropy Bonus Coefficient	0.008
Max Gradient Norm	1.0
Clipped Value Loss	True

APPENDIX B
SIM-TO-REAL DEPLOYMENT OF THE IK POLICY

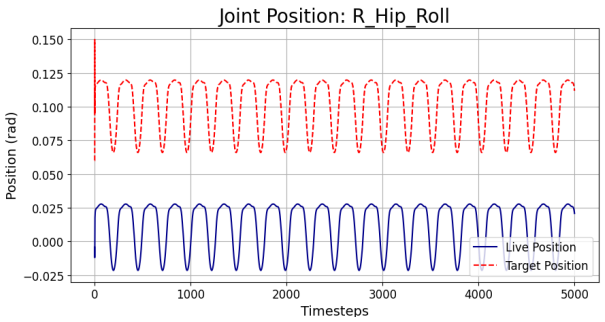


Fig. 10: This graph shows the target positions commanded by the single-leg IK policy (red) and the resulting joint positions from the simulated actuator model (blue) in NVIDIA Isaac Sim.

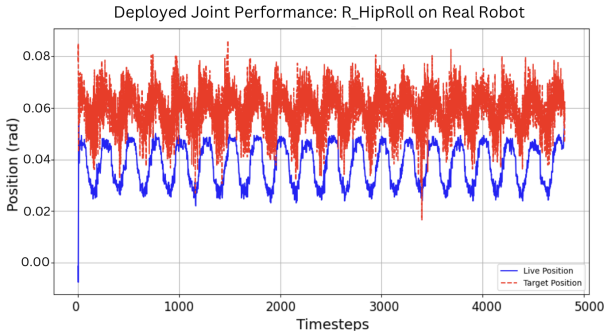


Fig. 11: This graph shows the target positions commanded by the single-leg IK policy (red) and the resulting joint positions (blue) when deployed on the real robot.

APPENDIX C
SIM-TO-REAL DEPLOYMENT OF STANDING POLICY

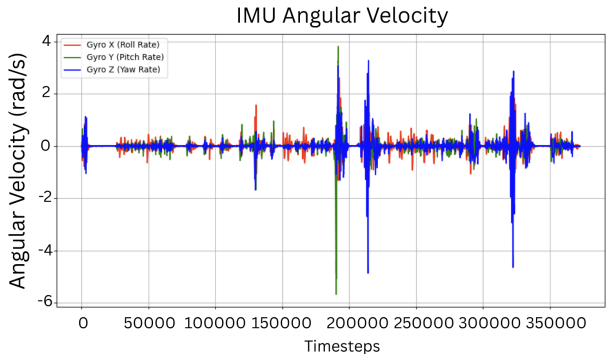


Fig. 12: Angular Velocity Obtained During Deployment of Standing Policy on Physical Hardware

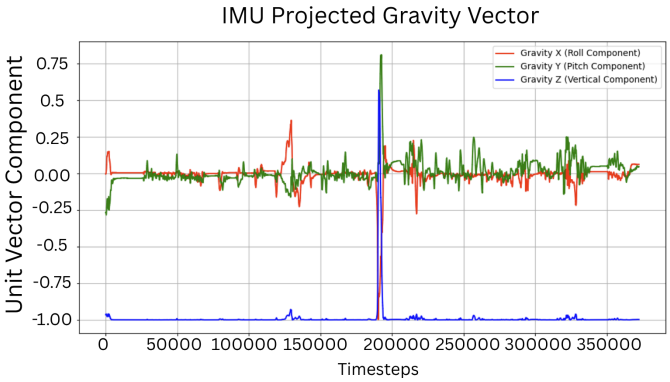


Fig. 13: Projected Gravity Vector Data During Deployment of Standing Policy on Physical Hardware

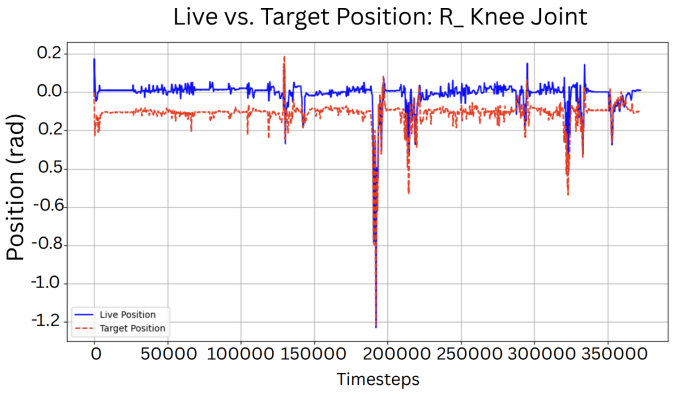


Fig. 14: Live vs Target Position of Right Knee Joint During Deployment of Standing Policy on Physical Hardware