Webots Incompatibilities During Hexapod RL/IRL Development: A Collection of Issues and Workarounds

Rafael André, Pedro Fernandes, Eduardo Passos

Index

1 Integration with Gymnasium

Following the simulation setup, our next objective was to integrate the system with `gymnasium`, enabling reinforcement learning workflows. This step introduced a number of non-trivial challenges.

Initially, we invested nearly a full week attempting to integrate the `deepbots` framework. Unfortunately, this approach proved unworkable due to a series of compatibility issues—particularly during the build process, where installation of necessary wheels repeatedly failed. These dependency and environment constraints rendered `deepbots` unusable in our case.

As a result, we abandoned the prebuilt framework and embraced the more demanding path of developing our own gym-compatible environment entirely from scratch.

2 Reinforcement Learning Development: Architecture and Challenges

2.1 Increasing Project Flexibility

The first step in customization is setting the robot to configurable states. This is done by setting the robot as a **base node**, which opens the node tree for modification. Logically, our first attempt at adding sensors was made through the GUI, but that did not work. We then decided to work directly with the proto file, as `Webots` never allowed addition of sensors (or anything else, for that matter) as direct children of the robot, even after being set as a base node.

This was the largest limiting factor in development: Whenever we attempted to manually add anything as a child of the main robot frame, the software would crash, initializing the default "empty.wbt" world, with no error message.

Countless weeks were wasted on attempts at increasing flexibility. Even with professor Gonçalo Leão's help, we were unable to find a solution.

2.2 Sensor Limitations and Environment Workarounds

Although our control script was designed to enable locomotion across uneven terrain, formerly mentioned limitations in sensor integration restricted us to flat surfaces. The `mantis` robot model does not allow new sensors to be added, making real-time terrain mapping impossible.

To compensate, we relied on the supervisor mode as well as the standard included IMU (only sensor that works, since it's integrated into the original project), which provide only relative position data and robot frame tilt. This workaround, however, introduced additional constraints.

2.2.1 Attempted Sensor Strategies

The most useful sensor values would've been LiDAR (should detect direct floor distance, would only return ∞ or simply collisions with the inside of the robot) and Rotational Motor Position Sensors (should display rotational motor angles relative to the arm, would only return hinge physics-related values).

Strategy	**Result**
Center of Mass (COM) control	Infeasible; COM values are robot-relative, not world-relative
LiDAR inside robot	Worked, but only detected self-collisions
LiDAR outside robot	Crashed 'Webots' or returned infinite readings
Bounding boxes/floor tweaking	No improvement; LiDAR still failed to detect terrain
Position Sensors for elbow angle	Values inconsistent with observed angles
Supervisor mode	Worked for flat terrain; used as final fallback

Table 1: Sensor strategy attempts and outcomes

As a result, we constrained the agent to a flat surface. We even tried to follow the logic of "If LiDAR can detect collision with the robot's body, then we can simply replicate its physics and characteristics, and make an entire floor follow those rules", but even then, it would return ∞.

In homage to Professor Luís Paulo Reis and his work in robotic soccer, we finally decided on using a soccer field as the training environment. Future work aims to reintroduce uneven terrain and additional tasks such as ball interaction.

3 Hardware and Simulation Issues

3.1 Headless Execution Constraints

Our initial architectural design aimed to enable fully headless training of the robot agent within `Webots`, eliminating the need for graphical rendering, and thereby improving automation and scalability. However, this proved infeasible due to `Webots`' sensor rendering dependencies.

Specifically, certain modules, such as the LiDAR sensor, require an active graphical rendering pipeline to function correctly. This hardware-tied limitation forced us to reintegrate GUI-based execution into our workflow.

Eventually, after understanding that LiDAR was not going to work, we decided to keep the graphical interface in order to follow the entire training development. The bottleneck in training time is based on the `Webots` software and not on the graphical rendering. This means that even if we used no GUI, the overall time would be the same.

Design Intent	Outcome
Headless simulation via CLI	Infeasible due to GUI requirement for sensor rendering
Sensor-only training loop	Blocked by real-time graphical dependency in ‘Webots‘

Table 2: Headless training goals vs. limitations

3.2 Motor Position Sensing Failures

We encountered a complete absence of valid position readings from the motors. Despite correcting actuator syntax and manually adding position sensors in the `.wbt` file, no usable data was returned.

3.3 Lack of Motor Integrity Constraints

With no limits on motor input, joints received arbitrary position values and behaved erratically, often resulting in physically implausible motion. This was later solved by adapting expert observed sinusoidal functions.

3.4 Instability and Jittering

Control signals from the RL model often led to highly unstable movement. This was caused by the mismatch between policy output magnitudes and motor response scales.

This obstacle was later mitigated through empirically defined, custom velocity limits for motor actuation.

3.5 Randomized Poses and Inertial Instability

Pose randomization for training diversity and difficulty levels led to frequent collapses due to imbalance in the center of mass. Any random start, even when guaranteeing correct orientation and integrity limits, would result in the same position. The method was abandoned due to unreliability.

3.5.1 Overall Issues Summarized

Issue	**Impact on Training**
No position sensor feedback	Inhibited validation and interpretability
Unlimited motor input	Caused uncontrolled, extreme joint movement
Unscaled RL outputs	Led to jittering, flipping, and erratic gait
Random start poses	Caused frequent physical instability

Table 3: Hardware and control issues

4 Final Solutions and Environment Refinement

Despite significant obstacles, we implemented a series of mitigations to enable robust learning and stable simulation:

Solution	Justification
Ignored position sensors	Gymnasium could infer feedback via rewards
Manual integrity constraints	Based on ‘Webots‘ demo values; prevented unrealistic behavior
Velocity limit tuning	Derived with professor’s assistance, to ensure precision
Custom initial poses	Ensured stability while supporting difficulty scaling
Soccer-field environment	Chosen for simplicity and thematic homage

Table 4: Final solutions adopted in simulation and training

5 Summary: Features We Aimed For but Could Not Implement

The following table summarizes the features we intended to include and the specific limitations that prevented their implementation:

Intended Feature	**Reason for Failure**	**Fallback or Resolution**
Headless simulation	GUI required for sensor rendering	Re-enabled GUI rendering in development loop
LiDAR-based terrain mapping	Sensor placement restricted; collisions undetected	Switched to flat terrain with supervisor mode
COM-based balance control	COM is robot-relative only	Discarded in favor of position constraints
Motor position sensors	Failed to return valid data	Ignored; relied on model feedback and rewards
Pose randomization	Caused imbalance and instability	Replaced with fixed, safe starting poses
Complex terrain navigation	Impossible to detect terrain height	Confined agent to flat soccer field

Table 5: Summary of planned vs. feasible features