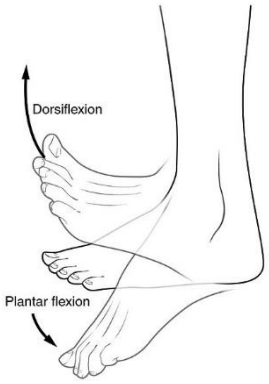
Ankle Exoskeleton Programming Challenge

Background:

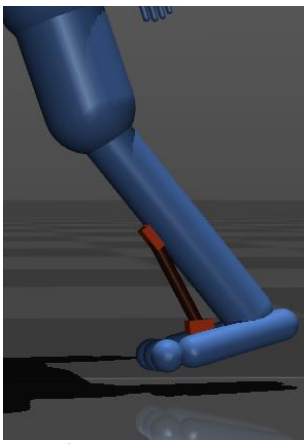
You'll be applying your existing knowledge on forward simulations with MuJoCo and explore the control problem of a cable-driven ankle exoskeleton. Foot drop is a common issue associated with stroke and other conditions, caused by a neuromechanical deficit during ankle dorsiflexion (e.g. insufficient neural drive to the tibialis anterior muscle).



Connexions, CC BY 3.0

Assistance from an exoskeleton can supply the missing dynamics. However, this must be properly synchronised with the user's movement. Furthermore, as little actuation as possible is preferred, to extend the exoskeleton's lifetime, reduce restrictions in degrees of freedom and to encourage the user's participation in the movement.

Awad and Walsh, et al., 2017. A soft robotic exosuit improves walking in patients after stroke.



Simplified model used in this challenge.

Goals:

Define the control logic for a cable driven exoskeleton model in MuJoCo, worn by a humanoid that's mimicking walking. By default no actuation happens in the left ankle, leading to tripping and large contact forces at the toe. Additionally large contact forces may arise after heel-strike which leads to foot slap due to a lack of eccentric dorsiflexion. Excessive dorsiflexion or actuation forces must also be avoided.

These goals are quantified in a reward function provided to you.

Signals:

- Accelerometer and gyroscope data from the shin and foot (shown in red).

Output:

- Cable tension force (in [0, 50] N range).

Tips:

A simple real-time plotting system is already implemented in the for of the DataCollecter class. It is used to plot the reward collected in real time, but you can use it to visualize the signals collected as well. Slow down the simulation (sim speed managed by the + and – keys) and observe how the signals evolve and what properties you could exploit. Try to manually change the "exo" control signal in the simulation using the slider interface to prevent tripping with the least force.

There's a good chance that the signals will be a noisy due to contacts and collisions. You should consider applying some filtering method to the window of data collected in a DataCollecter.

Feel free to edit the parameters of the DataCollecters. If you'd like tuneable parameters for your control logic, additional interactive sliders are included in the simulation. Feel free to edit their limits in the XML file. You can then adjust their values in real time and define and fine tune your behaviour based on them.

I recommend you try to implement a rule-based controller first, switching between different levels of exo activation. A well-tuned rule-based controller can get up to around 0.7/0.8 smoothed reward.

Bonus challenges:

State prediction: Explicitly predict early stance, late stance, early swing and late swing gait phases. Create a system to transition between the tracked states (a Finite State Machine). How far ahead can you predict it? You can explore using classifiers such as Support Vector Machines, with convenient implementations in packages such as `scikit-learn`. When collecting training data for a model like this, feel free to use ground-truth information for the training labels (e.g. progress in the gait cycle) that would otherwise not be available to you for your sensors.

Environment engineering: Edit the `track_mocap()`function to generate variable walking speeds smoothly increasing and decreasing, or modulated by a low frequency noise. How does this impact your controller? What is the most noise you can inject into your system before it can no longer control the exo?

Exoskeleton design: Edit the XML file to include actuation for plantarflexion as well, cable driven or otherwise. Can you apply push-off force at late stance? What happens if you enable inversion-eversion in the ankle?

Reward design: What other aspects would be important to include in the reward function? You could for example encourage following the mocap data for the ankle, or to discourage too early lift-off.