BE 124 Labs: Exoskeleton Control with the Biomotum Spark

Anway Pimpalkar

Abstract

This project introduces a sequence of hands-on labs designed to teach core principles of exoskeleton control using the Biomotum Spark ankle exoskeleton. Students begin with two foundational controllers: proportional torque control and gait phase-based torque control, to understand how torque magnitude, timing, and stance detection shape the assistance delivered during walking. Building on these ideas, the second half of the lab sequence focuses on human-in-the-loop optimization. A simplified version of Covariance Matrix Adaptation is first implemented in simulation to illustrate how controller parameters can be tuned through iterative sampling and performance feedback. The same framework is then deployed on the Spark, where torque-profile parameters are updated based on stride-speed measurements from an external IMU. Across the four modules, students move from fixed, hand-tuned controllers toward adaptive strategies that personalize assistance to the individual user. The labs provide a practical introduction to algorithmic control design, real-time sensing, and optimization in wearable robotics.

Note

Based on my discussion with Prof. Patrick Slade after submitting the first draft, I re-wrote this document as a lab manual for BE 124 rather than a standard project report following the syllabus format. I hope this provides a structure and layout that can serve as a more useful starting point for the teaching team to refine and build upon next semester.

Submission

1. Project report / write-up (this document). Editable .docx on GitHub.
2. GitHub: https://github.com/anwaypimpalkar/BE124-Labs
3. Setup guide: https://github.com/anwaypimpalkar/BE124-Labs/blob/main/docs/Setup_BiomotumSpark.pdf

---

BE 124 Labs: Exoskeleton Control with the Biomotum Spark

GitHub: https://github.com/anwaypimpalkar/BE124-Labs

Overview

The goal of these labs is to introduce you to the fundamentals of control strategies in assistive robotics, particularly exoskeletons. In the two labs, you will learn how different controllers work, when to use them, and how to implement them on the Biomotum Spark ankle exoskeleton.

You will complete build a total of four modules:

1. Lab 1.1: Proportional Torque Control
2. Lab 1.2: Gait Phase-based Torque Control
3. Lab 2.1: Simulating HILO with Gait Phase-Based Torque Control
4. Lab 2.2: Implementing HILO with Gait Phase-Based Torque Control

Pre-Lab Reading

Before starting Lab 1, read the following materials based on material discussed in class so far. You should understand the high-level idea of each controller, how it works, and what problem it solves.

Biomotum Spark Ankle Exoskeleton

In these labs you will use the Biomotum Spark ankle exoskeleton (Fig. 1), a bilateral device that assists ankle plantarflexion during walking. The system combines onboard sensing, actuation, and a vendor-provided software interface so that we can modify a limited set of control parameters and observe how they affect torque profiles and gait.

Actuation:

- Motor drive that provides assistive ankle torque up to ~23Nm.

Onboard Sensors:

- Force-sensing resistors (FSRs) under the foot to detect stance vs. swing.
- An in-line torque sensor to measure ankle torque (up to ~35 Nm).

Control Interfaces:

- A GUI application (Spark software) used to start sessions, select control modes, and save data.
- A Python API that allows you to:
    - Turn motors on/off
    - Change stance/swing torque setpoints
    - Adjust FSR thresholds
    - Run FSR calibration
    - Read basic telemetry (torque, FSR state, stance/swing state, elapsed time, battery level)

Device Electronics:

- All electronics including the rechargeable battery, motor drivers, and microcontroller are housed inside the waist-unit.

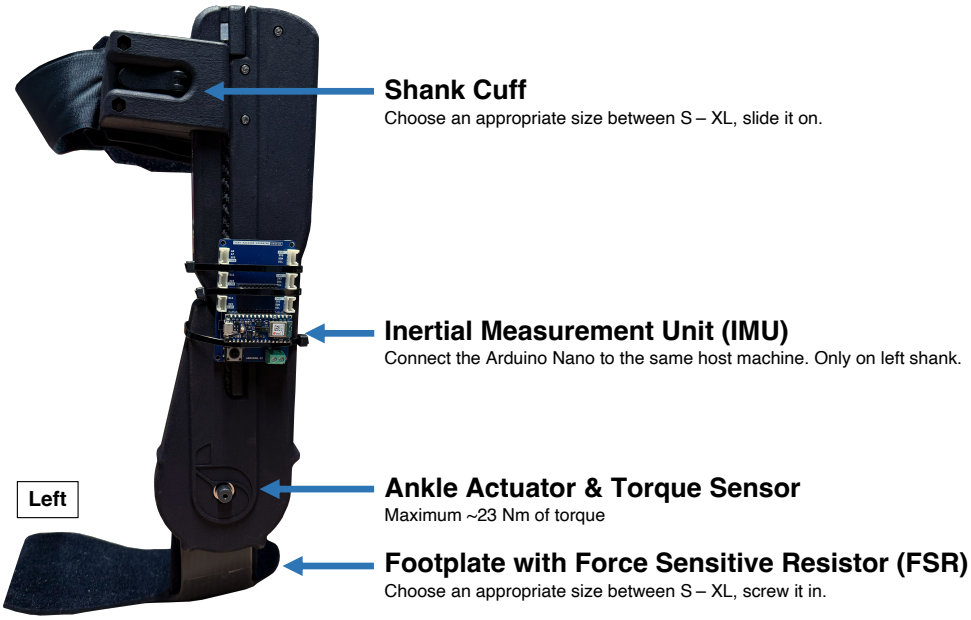

Fig. 1. Biomotum Spark ankle exoskeleton.

External IMU for Shank Angle Measurement:
We add an external IMU to estimate shank orientation and use those measurements to compute stride speed. The external sensor is an Arduino Nano 33 BLE Sense with an onboard 9-axis LSM9DS1 IMU. It is mounted on the lateral side of the left shank, aligned so that the IMU's pitch axis reflects the shank's forward-backward rotation.

Read more:
Bishop, E., & Li, Q. (2010). Walking speed estimation using shank-mounted accelerometers. In 2010 IEEE International Conference on Robotics and Automation (pp. 5096–5101). 2010 IEEE International Conference on Robotics and Automation (ICRA 2010). IEEE. https://doi.org/10.1109/robot.2010.5509504

For this course, we will treat the Spark as a partially "closed-loop" system: we adjust the parameters that are exposed through their API (e.g., torque setpoints and simple thresholds) and then observe how these changes influence the delivered torque and the user's gait.

Proportional Torque Control

In proportional torque control, the exoskeleton delivers torque that is proportional to an internally estimated biological weight (T_{BWT}) torque during stance (Fig 2 left). The Spark uses its FSRs to detect when the foot is in stance, and during that period it computes a reference torque and multiplies it by a user-selected gain k.

The controller computes torque as:

$$T_{exo} = k \times T_{BWT}$$

How the Spark Estimates T_{BWT}:

Although the Spark does not measure biological ankle torque directly, it approximates it using:

- User body weight (entered during setup)
- A built-in normalized ankle moment profile derived from gait datasets
- Stance detection from FSRs to scale the moment over the stance phase

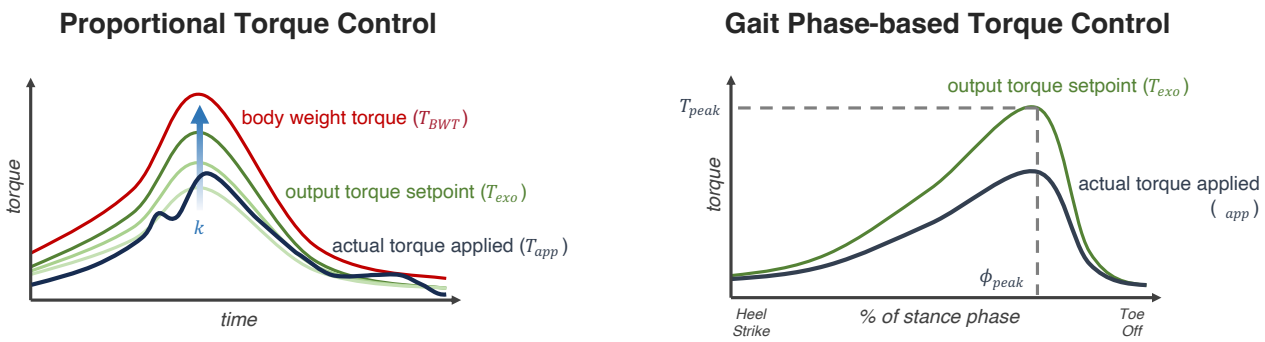

Fig. 2. Representative figures for proportional torque control (left) and gait phase-based torque control.

The process is conceptually:

1. Use body weight W to scale a *normative ankle moment curve:*
$$T_{norm}(p) \rightarrow T_{norm}(p) \times W$$
where p is percent of stance.
2. Use FSR timing to map the curve onto the actual stance duration for each step.
3. Output T_{BWT} as the system's estimate of the user's expected biological plantarflexion torque.

This produces a reference biological torque waveform that has a fixed shape (taken from normative data), a timing aligned to the user's stance phase, and a magnitude scaled by body weight.

Gait-phase Based Torque Control

Instead of basing torque directly on an instantaneous measurement (as in proportional control), gait-phase based control defines torque as a function of stance-phase percentage. The exoskeleton estimates where the user is within stance (0% at heel-strike to 100% at toe-off) and generates a torque profile that follows a predefined curve over this interval (Fig 2 right). Each step is scaled to this 0 to 100% timeline, regardless of how long the step actually takes.

The torque profile is defined by two key points:

- Peak torque (T_{peak}): The maximum desired torque amplitude.
- Peak timing (ϕ_{peak}): The time point in stance phase where torque reaches its maximum.

The torque command is defined as:

$$T_{exo}(\phi) = T_{peak} S(\phi - \phi_{peak})$$

where:

- $S(\cdot)$ has a maximum value of 1 at $\phi = \phi_{\text{peak}}$.
- $S(\cdot)$ smoothly returns to zero near the start and end of stance.
- Changing T_{peak} scales the profile vertically.
- Shifting ϕ_{peak} moves the profile earlier or later in stance.

Human-in-the-Loop Optimization with Gait-phase Based Torque Control

Human-in-the-loop optimization (HILO) replaces manual parameter tuning with an algorithm that automatically searches for torque-control parameters that improve a measurable aspect of human performance. Instead of

guessing onset timing, peak timing, or torque magnitude, we let the device iteratively try different settings, measure their effect, and learn which settings work best.

In the context of exoskeleton control, HILO treats the parameters T_{peak} and ϕ_{peak} as tunable variables in an optimization problem. For each trial, the controller uses a specific set of parameters, and we evaluate how well that parameter set supports the walking task.

To optimize a controller, we need a performance metric that can be measured for each trial. Examples include metabolic cost or kinematic metrics (such as stride speed, shank angle at initial contact, step symmetry, step length asymmetry). The objective is to make this metric match a desired target (e.g., for stride speed, achieve a higher speed). A lower error means the chosen control parameters produced a more desirable gait outcome.

Broadly, a typical HILO loop consists of:

1. The algorithm selects a candidate set of control parameters (e.g., T_{peak} and ϕ_{peak}).
2. The exoskeleton applies the torque profile defined by those parameters.
3. We compute the performance metric.
4. The algorithm uses the performance feedback to propose new, improved parameter values.
5. Over successive iterations, parameter choices tend to improve, ideally converging toward a better assistance strategy.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

CMA is a derivative-free, population-based stochastic optimizer, which means it does not require gradients or a known model of the human-exoskeleton system. Instead, it searches for good controller parameters by sampling, testing, and evolving a population of candidate solutions. The process repeats in cycles, as illustrated in the diagram Fig 3.

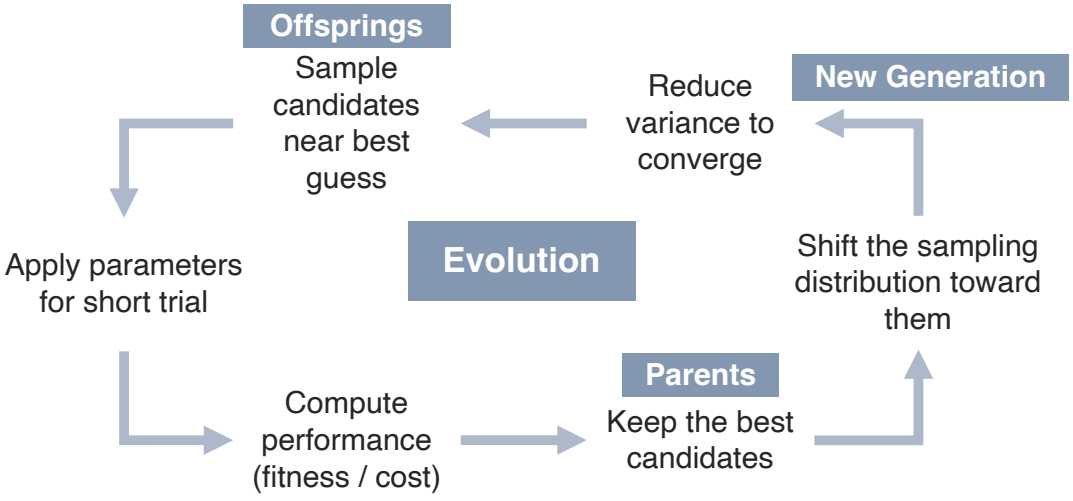

Fig. 3. Schematic representation of CMA-ES.

1. **Offsprings (sample new candidates):** CMA begins with a best guess for the parameters (e.g., peak torque and peak timing). It then generates a batch of nearby parameter sets, called offspring, by sampling from a Gaussian distribution centered around the current best estimate. These samples represent different control strategies the algorithm wants to test.
2. **Apply parameters for a short trial:** Each sampled parameter set is used during a brief walking trial.

3. **Compute performance (fitness or cost):** After each trial, the algorithm computes a performance score based on the metric of interest. This step tells CMA how well each candidate worked.
4. **Parents (keep the best candidates):** CMA selects the top-performing candidates from the offspring population. These “parents” represent parameter sets that most effectively improved the walking metric. CMA ignores poorly performing candidates; they are not used to guide the next generation.
5. **New generation (shift the sampling distribution):** The algorithm then updates its sampling distribution by shifting the mean toward the best candidates (moves the search to more promising regions) and adjusting the covariance (expands exploration when uncertain or reduces variance to converge when close to a good solution). This is how CMA “learns” which directions in parameter space are helpful.
6. **Evolution (repeat the cycle):** With the updated sampling distribution, CMA generates a new batch of candidates and the cycle repeats. Over iterations, good parameter sets become more common, poor parameter sets fade out, and the search narrows as CMA converges.

Read more:

Slade, P., Atkeson, C., Donelan, J. M., Houdijk, H., Ingraham, K. A., Kim, M., Kong, K., Poggensee, K. L., Riener, R., Steinert, M., Zhang, J., & Collins, S. H. (2024). On human-in-the-loop optimization of human–robot interaction. Nature, 633(8031), 779–788. https://doi.org/10.1038/s41586-024-07697-2

Slade, P., Kochenderfer, M. J., Delp, S. L., & Collins, S. H. (2022). Personalizing exoskeleton assistance while walking in the real world. Nature, 610(7931), 277–282. https://doi.org/10.1038/s41586-022-05191-1

Lab 1.1: Proportional Torque Controller

In the first part of this lab, you will implement and test a proportional torque controller on the Biomotum Spark using the provided Python script. The goal is to understand how scaling the reference T_{BWT} affects the magnitude of the delivered torque during stance.

Setup:

- Follow the setup instructions to enable Scripted Control in the Spark GUI.
- Only allow a group member to wear the Spark exoskeleton when a TF is present for supervision.

Complete `Lab_1-1.py`**:**

The template script is provided in the course GitHub. Download the repository and complete the following section in the script:

```
################################
# TODO: Complete the parameters #
################################

K_VALUES = [ ... ]      # scaling factors to iterate through
DWELL_SECONDS = 20      # time to walk for each k (seconds)
RAMP_UP_STEPS = 1       # ramp-up period in steps
```

Run the Script:

The script will step through each gain value in K_VALUES. For each gain:

1. The controller sets $T_{exo} = k \times T_{BWT}$.
2. The user wearing the exoskeleton can walk continuously, and the other group members can notify the user when the new gain is set by the script.
3. Torque and sensor telemetry are logged during the trial.

Your responsibility is to:

1. Confirm that stance detection (FSRs) is functioning.
2. Verify that torque magnitudes increase as k increases.
3. Annotate any qualitative differences the user reports (comfort, stability, effort).
4. After completing all gain conditions, use the Spark GUI to save the session data as a .csv file.

Reflect:

1. How did increasing or decreasing the scaling factor k change the perceived assistance during stance? (e.g., did higher k feel supportive, intrusive, destabilizing, or harder to control?)
2. How did different k values influence the exoskeleton's measured torque tracking? (e.g., did the actual torque match the commanded setpoint better or worse as k increased?)
3. Did changes in k affect your gait pattern, such as stance duration, symmetry, or smoothness? (if so, how did your body adapt at low vs. high scaling factors?)

Lab 1.2: Gait Phase-Based Torque Controller

In this second part of this lab, you will implement and evaluate a gait phase-based torque controller using the Biomotum Spark. You will run walking trials with nine different torque profiles that vary in T_{peak} and ϕ_{peak}, and observe how these choices affect the delivered assistance during stance.

Setup:

- Follow the setup instructions to enable Scripted Control in the Spark GUI.
- Only allow a group member to wear the Spark exoskeleton when a TF is present for supervision.

Complete `Lab_1-2.py`**:**

The template script is provided in the course GitHub. Download the repository and complete the following section in the script.

```
##################################
# TODO: Complete the parameters #
##################################

TORQUE_PROFILES = [     # Torque profiles to iterate through
      [(0, 0), (phi_peak1, T_peak1), (100, 0)],
      [(0, 0), (phi_peak2, T_peak2), (100, 0)],
      ...
]
DWELL_SECONDS = 20      # time to walk for each k (seconds)
RAMP_UP_STEPS = 1       # ramp-up period in steps
```

Note: T_{peak} must be an integer value between 0 and 23 Nm, and ϕ_{peak} must be a value between 0 and 100%.

Run the Script:

For each torque profile in `TORQUE_PROFILES`:

1. The script applies the corresponding spline-based torque shape: $T_{exo}(\phi) = T_{peak}S(\phi - \phi_{peak})$
2. The user wearing the exoskeleton can walk continuously, and the other group members can notify the user when a new profile is set by the script.
3. Torque and sensor telemetry are logged during the trial.

Your responsibility is to:

1. Observe how torque magnitude differs between high and low peak conditions.
2. Collect the user's subjective feedback about comfort, timing, and perceived assistance.
3. After completing all profiles, use the Spark GUI to save the session data as a .csv file.

Reflect:

1. How did shifting ϕ_{peak} change your perception of assistance?
   (e.g., did earlier/later peaks feel more natural or helpful?)
2. How did changes in the torque-profile shape or peak magnitude affect the applied torque?
   (e.g., did the actual torque closely follow the desired stance-phase curve?)
3. Did modifying the timing or height of the torque peak influence your gait mechanics?
   (e.g., did it alter push-off timing, stance stability, strides, or symmetry? how did you adapt?)

Lab 2.1: Simulating HILO with Gait Phase-Based Torque Control

Now, you will implement CMA on a toy problem so you can see how the algorithm behaves before connecting it to the exoskeleton.

Your goal is to search for the phase-based torque parameters that maximize stride speed, under the assumption that better-timed and appropriately sized plantarflexion assistance will help the user walk slightly faster and more efficiently. CMA handles this by evaluating a batch of candidate torque profiles, observing how each profile affects the measured stride speed during a short walking interval, and then updating the parameter distribution toward those that produce higher speeds. Over several generations, the algorithm should gradually shift its sampling toward profiles with both higher peak torque (within safe limits) and more effective timing, ultimately converging on parameter sets that yield the highest observed stride speed for that user.

In this lab, you will implement CMA on a toy problem that mimics our real goal of maximizing stride speed. Instead of measuring real stride speed, the script uses a synthetic function that rewards high peak torque and specific timing. CMA evaluates batches of candidate torque-timing pairs, observes their (simulated) stride speed, and shifts its sampling toward settings that perform better.

Complete `Lab_2-1.py`**:**
The template script is provided in the course GitHub. Download the repository and complete the following section in the script, given the following pseudocode:

```
############################
#### TODO: Implement CMA ####
############################

initialize mean = [T_peak_mean, phi_peak_mean]
initialize sigma = [sigma_T, sigma_phi]
for each generation:
    for each candidate i in 1.. lambda:
        theta_i = mean + sigma * random_normal_vector()
        theta_i = clamp_to_valid_bounds(theta_i)
        stride_speed_i = run_trial_and_measure_stride_speed(theta_i)
        cost_i = -stride_speed_i
    select mu best candidates (lowest cost)
    mean = average_of_best_candidates()
    sigma = 0.8 * sigma
```

Note: T_{peak} must be an integer value between 0 and 23 Nm, and ϕ_{peak} must be a value between 0 and 100%.

For this script, we will use the following CMA configuration:

Dimension:

$N = 2$

Offspring per generation:

$\lambda = 4 + \lfloor 3\ln(N) \rfloor$

$\lambda = 4 + \lfloor 3\ln(2) \rfloor$

$\lambda = 6$ *candidates per generation*

Parents:

$\mu = \left\lfloor \frac{\lambda}{2} \right\rfloor = 3$

$\mu = 3$ *best candidates used for update*

Generations:

4 generations $\rightarrow 4 \times 6 = 24$ *trials total*

Run the Script:

When you run the CMA script for Lab 2.1:

1. It repeatedly samples candidate torque–timing parameters around the current mean, evaluates and ranks them, updates the mean using the best performers, and gradually shrinks the sampling spread across four generations.
2. Plots how the sampled candidates and the mean move in parameter space across generations.
3. Plots how fitness (-cost ≈ stride speed) and step sizes sigma change over generations.

Your responsibility is to:

1. Verify that the mean parameters move toward high-torque, mid/late-stance values.
2. Confirm that the step sizes sigma decrease over generations, indicating that the search is narrowing.
3. Examine the fitness vs generation plot and check that best and mean fitness improve over time.
4. Save the final evolution plots and record the initial and final mean parameter values for your lab report.

Lab 2.2: Implementing HILO with Gait Phase-Based Torque Control

Finally, once you have confirmed that your CMA algorithm works correctly and a TF has verified your function, you will implement this CMA algorithm on the Spark exoskeleton. Now, you will replace the synthetic test function used in Lab 2.1 with a performance metric measured directly from the user. We will optimize for stride speed, which is estimated using the IMU mounted on the Arduino. The IMU processing script computes a stride-speed value and returns an updated measurement approximately every five seconds during walking. This serves as the input to your CMA cost function.

Setup:

- Follow the setup instructions to enable Scripted Control in the Spark GUI.
- Only allow a group member to wear the Spark exoskeleton when a TF is present for supervision.

Complete `Lab_2-2.py`**:**

The template script is provided in the course GitHub. Download the repository and complete the `T0D0` section in the script by using the function you wrote for Lab 2.1.

Run the Script:

When you run the Lab 2.2 script, the CMA algorithm you implemented will now interact with the Spark exoskeleton in real time. For each generation, the script will:

1. Generate several candidate torque–timing parameter sets using your CMA sampler.
2. Apply each candidate to the Spark by updating T_{peak} and ϕ_{peak} in the phase-based controller.
3. The user wearing the exoskeleton can walk continuously, and the other group members can notify the user when a new profile is set by the script.
4. Allow the user to walk for an interval of 30 seconds and obtain average the stride speed over time.
5. Feed this stride speed measurement into your CMA cost function and record the performance of each candidate.
6. Update the CMA mean and step sizes based on the best-performing candidates, preparing the next generation of parameter samples.

Your responsibility is to:

1. Monitor that the Spark is receiving and applying each new torque profile correctly.
2. Ensure stride-speed values are being returned by the IMU and passed into the CMA objective.
3. Verify that CMA updates behave sensibly (mean parameters gradually shift, search variance shrinks).
4. Save Spark telemetry and IMU logs at the end of the session for later analysis.

Reflect:

1. How did changing the peak torque magnitude and timing influence stride performance (e.g., speed, smoothness, or stability)? (consider whether certain parameter combinations felt more “effective” or biomechanically aligned with your push-off phase)
2. What trends did you observe across generations of the CMA search? (did the optimizer consistently move toward higher torques, later timing, or a narrower region of the parameter space?)
3. Why might CMA-ES be well-suited for human-in-the-loop optimization of exoskeleton control parameters? (reflect on noise in human gait, stride-to-stride variability, and the lack of a known gradient)
4. Did the optimized parameters align with your subjective perception of assistance? (if not, why might the objective metric favor a different profile than your comfort-based intuition?)
5. How might the optimal profile differ for individuals with impaired gait, and what does that imply about the need for personalized control?

Results

Lab 1.1: Proportional Torque Controller

For Lab 1.1, the proportional torque controller worked as expected for different k values: increasing k produced proportionally higher stance-phase torques. The measured torque traces seen in Fig. 5 show clear scaling with the commanded setpoints and confirm that proportional assistance was applied reliably.

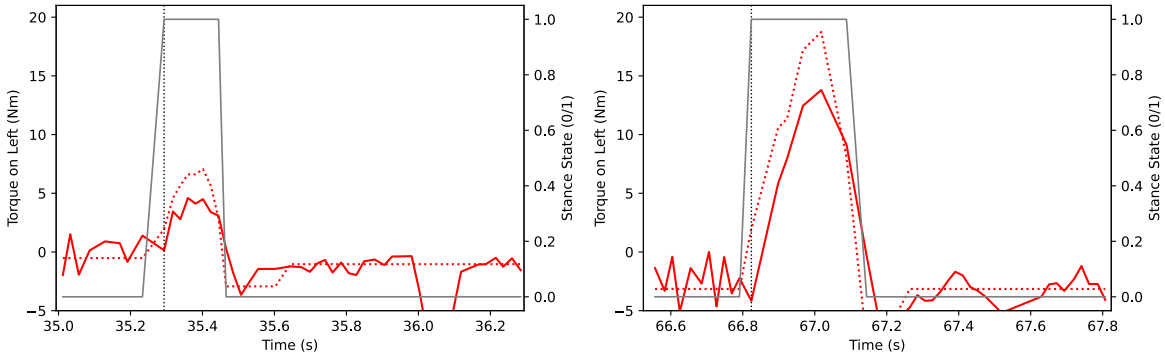

Fig. 5. Torque setpoint (dotted red) and applied torque (solid red) during stance (gray). Left: low k = 0.2; Right: high k = 0.9.

Lab 1.2: Gait Phase-based Torque Controller

When programmed with four input profiles (Table 1 and Fig. 6), the gait phase-based torque controller generated torque waveforms whose timing and general shape matched the programmed three-node profiles, with peaks occurring at the expected stance-phase locations (Fig. 7).

Table 1. Three-node control points defining four spline torque profile.

#	Control Point 1	Control Point 2	Control Point 3
1	0, 0	20, 19	100, 0
2	0, 0	20, 6	100, 0
3	0, 0	80, 19	100, 0
4	0, 0	80, 6	100, 0

(x, y) = (Stance Phase %, Torque Peak [Nm])

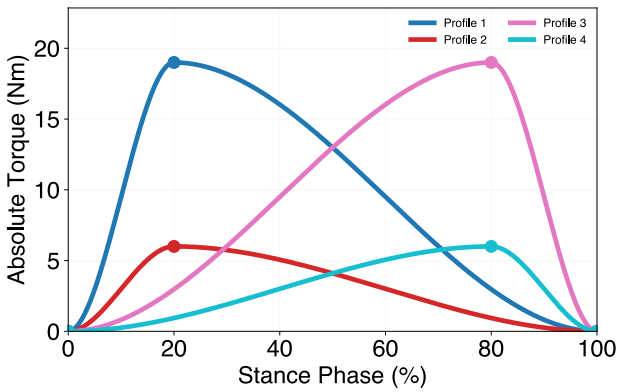

Fig. 6. Right: Resulting peak-preserving cubic torque waveforms generated from these control points.

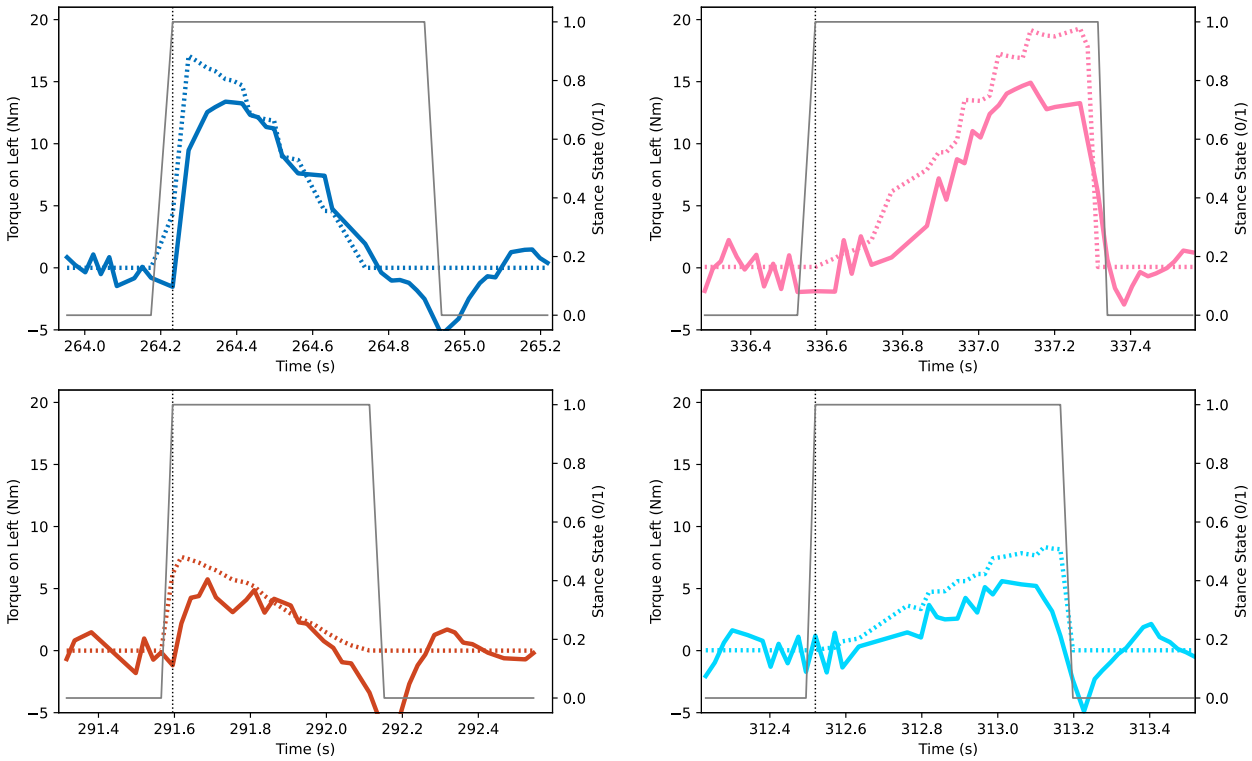

Fig 7. Commanded three-node spline torque profiles (dotted) and measured torque (solid) across four programmed peak locations during stance phase (gray).

Lab 2.1: Simulating HILO with a Gait Phase-based Torque Controller

A simulated CMA optimizer successfully learned to shift the peak ankle torque toward higher magnitudes and later stance phases over four generations (Fig. 8). The mean parameter and step-size trajectories together show that the algorithm both moved the search mean toward the optimum and reduced exploration as it converged, which validates the simulation structure and goals.

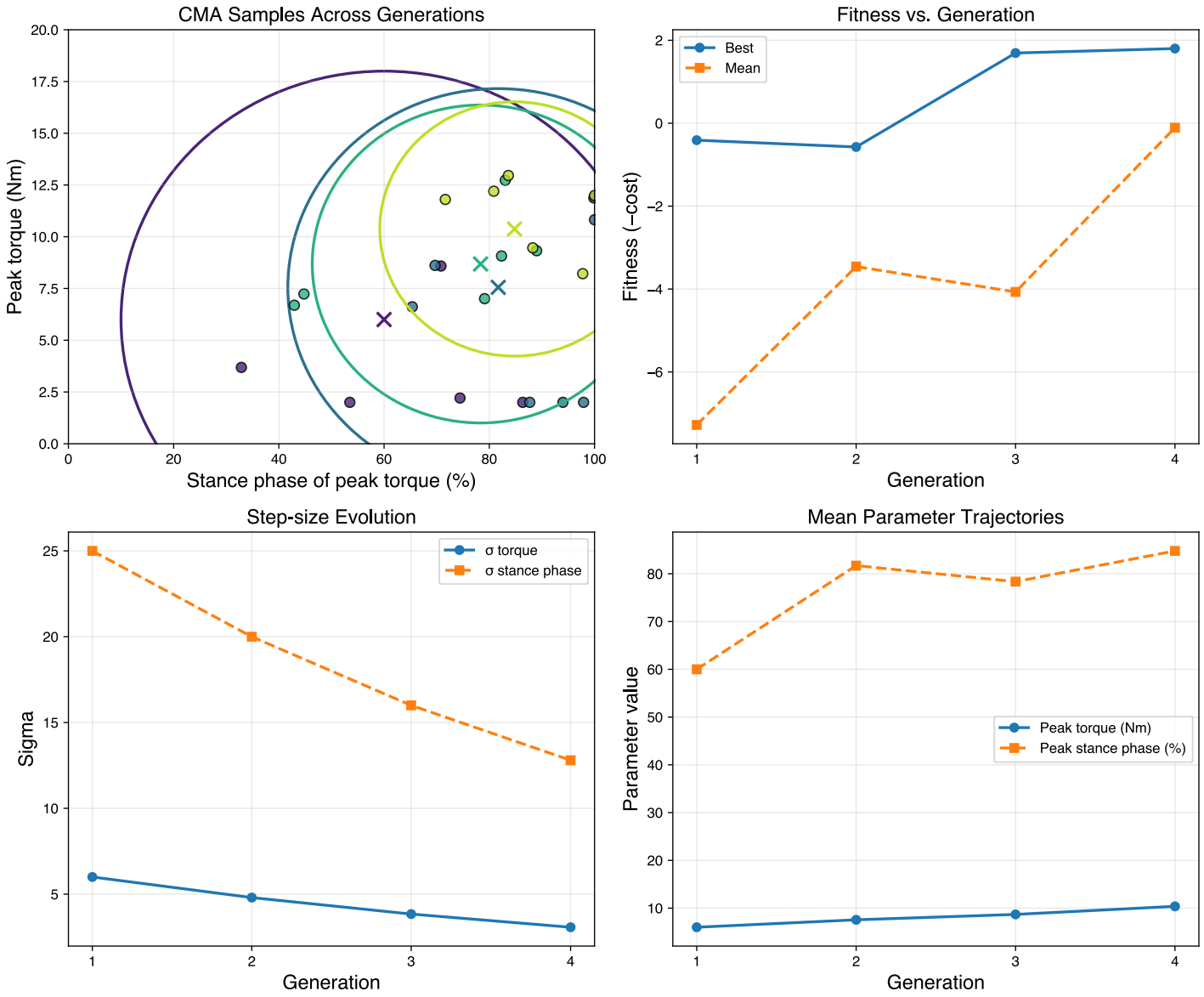

Fig. 8. CMA evolution over four generations in the toy optimization task. Top left: Sampled candidates with 2σ ellipses for each generation. Top right: Best and mean fitness values per generation. Bottom left: Step-size decay for torque and peak-timing parameters. Bottom right: Mean torque and timing parameters across generations.

Future Work

Once the Biomotum team provides updated guidance on correct torque-profile behavior and API functions, the scripts developed here should be revised to match the official implementation, ensuring that spline generation, setpoint timing, and controller execution behave consistently with the intended hardware specifications.