

BIOHYBRID ROBOTICS

MASTER LEARNING GUIDE

What to learn. How deeply. In what order. Why it matters for biohybrid robots.

Who this is for: BSc DSAI graduate entering MTech RAI at IIT Guwahati, targeting a PhD position at ETH Zurich Soft Robotics Lab (Prof. Robert Katzschmann) in biohybrid living systems and musculoskeletal robotics.

What this guide does: It names every specific Python library function, mathematical concept, algorithm, robotics principle and biology idea you need — and explains why you need it, how deeply, and what to build with it.

Companion: PDF 2 is the Trimester 1 exam prep guide (Dec 5-7, 2026).

Daily availability: 8-12 hours. Exam dates: Dec 5-7, 2026. Start date: Sep 23, 2026.

CHAPTER 1

The Honest Diagnosis: What You Need and Why

Your BSc DSAI gave you nine trimesters of broad exposure. That is not the same as the depth you need now. This chapter explains, specifically, what a biohybrid robotics PhD requires and what that means for your learning plan.

What Biohybrid Robotics Actually Demands

A biohybrid robot is a machine in which living biological tissue — usually muscle cells grown on an engineered scaffold — provides actuation, sensing, or both. The engineers who build and study them need to do three things that are genuinely difficult: measure the living system (which behaves unpredictably), build computational models of it (which involves both physics and biology), and close a control loop around it (which requires fast, reliable feedback). These three capabilities each draw on a specific, deep technical foundation.

The measurement problem is primarily a computer vision and signal processing problem. Muscle constructs are typically imaged under microscopes. Displacement is extracted from pixel data. Force requires either a physical sensor or a calibrated mechanical model. Every step in this chain — illumination, optics, pixel intensity, calibration, tracking, filtering, unit conversion — can introduce errors that make downstream analysis meaningless. The tools for this come from SSO5203, OpenCV, NumPy, and probability theory.

The modelling problem requires differential equations, system identification, and parameter estimation. Muscle activation follows a calcium transient triggered by electrical or optical stimulation. That transient has a rise time and a decay time. The mechanical response — how much force the muscle generates at a given length and velocity — is described by the Hill muscle model. Fitting those parameters from noisy measurements, and predicting the system response to new inputs, is a data science and control engineering problem. The tools are SciPy ODE solvers, NumPy least-squares fitting, and Kalman filtering.

The control problem requires understanding feedback, stability, and robustness. A biohybrid actuator is not a motor: its output drifts over time as the tissue fatigues, its response depends on temperature and nutrient levels, and its dynamics are nonlinear. A controller that works for one construct may fail for another. This demands not just a PID controller but an understanding of why PID fails and what alternatives exist. The tools come from SSO5201, SSO5202, and the python-control library.

What ETH Soft Robotics Lab Specifically Looks For

The Katzschmann lab at ETH works on four research directions: soft and musculoskeletal robotics, biohybrid living systems, dexterous manipulation with robot learning, and differentiable simulation

for fabrication. A candidate for the biohybrid direction needs to demonstrate depth in at least one of: (a) closed-loop control of compliant or biological actuators, (b) optical or mechanical measurement of deformable systems, (c) data-driven modelling of systems with biological variability, or (d) tissue engineering and scaffold fabrication. You cannot be deep in all four. The strategy in this guide is to build genuine depth in (a) and (b), with working knowledge of (c), and conceptual literacy in (d).

The lab is explicit that they do not hire against narrow project descriptions. They want to know: what technical area are you deep in, and how does that depth let you contribute to biohybrid research? Your application needs a specific, honest answer to that question backed by a GitHub repository, a research report, and a referee who watched you work.

The Gap You Need to Close

Your BSc DSAI gave you tools. What you are missing is the specific application of those tools to physical systems, the ability to model uncertainty and propagate it correctly, and practical experience with real hardware or real biological data. This guide tells you exactly how to close that gap in the time available.

How to Use This Guide

Each chapter covers one technical area. Within each chapter, topics are ordered from prerequisite to advanced. For each topic you will find: what it is, why it matters for biohybrid research, how deeply you need to learn it, what to read, and what to build. The guide is meant to be read once for orientation and then used as a reference as you work. Do not read it passively. After each section, do the stated exercise before continuing.

The parallel exam companion (PDF 2) covers SSO5201, SSO5202, and SSO5203 for the December 5-7 exams. Those subjects overlap heavily with this guide, so time spent on them is not wasted. The difference is intent: exam preparation is about reliable performance on structured problems. This guide is about building genuine research capability.

CHAPTER 2

Python and Scientific Computing: What You Specifically Need

Python in research is not the same as Python in web development or introductory courses. This chapter specifies exactly which library features, idioms, and practices you need — and explains why each one matters for biohybrid work.

NumPy: The Foundation You Must Own

NumPy is not just a library for fast math. It is the data model underlying the entire scientific Python stack. You need to understand it deeply enough that you can diagnose mistakes, not just use functions correctly. The specific concepts are:

Array Layout and Why It Matters

A NumPy array stores data in a contiguous block of memory with a specific dtype (e.g. float64 = 8 bytes per element). The shape and strides attributes tell NumPy how to interpret that block. A (3, 4) array with strides (32, 8) means: to move one row, jump 32 bytes; to move one column, jump 8 bytes. This is row-major (C) order. Understanding this explains why transposing a large array is cheap (just swap the strides) but why iterating over a transposed array in a loop is slow (memory access jumps around). For biohybrid work, you will process video frames (arrays of shape [T, H, W] or [T, H, W, C]) and sensor timeseries ([T, N_sensors]). Knowing the memory layout lets you write code that is fast without resorting to Cython or C.

EXERCISE — NumPy layout

Create a (1000, 1000) float64 array. Compare the time for `a.sum(axis=0)` versus `a.T.sum(axis=1)`. They compute the same thing. Measure the difference with `%timeit` in a Jupyter notebook and explain why they differ using the strides argument.

Broadcasting Rules

Broadcasting allows NumPy to perform operations on arrays of different shapes without copying data. The rules: align shapes from the right; dimensions must match or one must be 1; size-1 dimensions are virtually expanded. For example, subtracting the mean of a (T, N) sensor timeseries: (T, N) - (N,) works because the (N,) array is broadcast across T rows. This matters constantly in signal processing and calibration code.

EXERCISE — Broadcasting

You have a (5000, 6) array of IMU data (5000 time steps, 6 channels). Normalise each channel to zero mean and unit variance in one line using broadcasting. Verify that the result has mean 0 and std 1 per column. Do not use a loop.

Vectorisation vs Loops

A Python loop over a NumPy array is slow because of interpreter overhead at each step. Vectorised operations (those that operate on whole arrays) run in compiled C or Fortran code and

are typically 10-100x faster. For biohybrid work, you will process tens of thousands of video frames and long sensor timeseries. Code that takes 10 seconds on a small dataset may take hours on a full experiment if written with loops. The test: if you find yourself writing `for i in range(len(signal))`, look for a NumPy operation that does the same thing.

EXERCISE — Vectorisation

Implement a causal moving average (window=5) on a 1D array of length 10000. Do it first with a Python for loop, then with `np.convolve`, then with a stride trick using `np.lib.stride_tricks.sliding_window_view`.

Measure runtime for all three. Understand why they differ.

SciPy: Solving Scientific Problems

SciPy adds numerical algorithms on top of NumPy. The modules you specifically need are:

scipy.integrate — ODE Solving

Muscle activation and mechanical dynamics are described by ordinary differential equations. The activation ODE for a first-order model is $da/dt = (u - a) / \tau$, where u is the stimulation input (0 or 1), a is the activation level (0 to 1), and τ is the time constant (typically 10-50 ms). `scipy.integrate.solve_ivp` solves this given initial conditions and a time vector. You must understand the difference between stiff and non-stiff ODEs (muscle models are often stiff when τ is small), the meaning of `rtol` and `atol` tolerances, and how to interpret the solver's output. A second-order mechanical model couples the activation ODE to a force-displacement equation.

EXERCISE — ODE solving

Implement the two-state muscle model: $da/dt = (u-a)/\tau_{act}$ if $u>a$ else $(u-a)/\tau_{deact}$, and $dx/dt = v$, $dv/dt = (F_{active}(a,x) - F_{passive}(x) - b*v) / m$. Use $F_{active} = a * F_{max} * f_l(x)$ where $f_l(x) = \exp(-((x-x_0)/w)^2)$ is a Gaussian force-length curve. Stimulate with a 1 Hz square wave (50% duty cycle). Plot activation, force, and displacement. This model will be the basis of your T1-C project.

scipy.signal — Signal Processing

Sensor signals from force transducers, accelerometers, or image analysis pipelines are noisy. You need: `butter` (design a Butterworth filter), `filtfilt` (zero-phase filtering — critical: `scipy.signal.lfilter` introduces phase delay which matters for control), `find_peaks` (detect contraction events), `welch` (power spectral density to identify noise sources and signal bandwidth). For biohybrid imaging, you will also use `convolve2d` for spatial filtering of microscopy frames.

scipy.optimize — Parameter Fitting

Given experimental data and a model, you need to find model parameters that best fit the data. `scipy.optimize.minimize` with `method="L-BFGS-B"` handles this for continuous, differentiable objective functions. `scipy.optimize.curve_fit` fits parameterised functions with uncertainty estimates (it returns parameter values AND their covariance matrix — use that covariance to report confidence intervals, not just point estimates). `scipy.optimize.differential_evolution` is a gradient-free method useful when your objective function has many local minima, which is common in muscle model fitting.

EXERCISE — Parameter fitting

Generate noisy data from the muscle force-length curve $f_l(x) = \exp(-((x-x_0)/w)^2)$ with $x_0=1.0$, $w=0.3$, plus Gaussian noise with `std=0.05`. Use `curve_fit` to recover x_0 and w . Print the estimated parameters AND

their 95% confidence intervals ($2 * \text{sqrt}$ of the diagonal of the covariance matrix). Repeat with 10 noisy datasets and show the distribution of recovered parameters.

OpenCV and Image Analysis

This is where most biohybrid measurement work happens. The specific OpenCV functions you need to understand internally (not just call) are:

Camera Calibration: `cv2.calibrateCamera`

This function takes a set of 3D world points and their 2D image projections (from multiple checkerboard images) and returns the camera matrix K and distortion coefficients. K has the form $\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$. f_x and f_y are focal lengths in pixels. c_x , c_y is the principal point. You need to understand what each parameter means physically and how to use `cv2.undistort` to correct images before measuring anything. The reprojection error (RMS pixel distance between projected and observed points) tells you how good your calibration is. For biohybrid work, a reprojection error above 1 pixel means your subsequent displacement measurements are unreliable.

Optical Flow: `cv2.calcOpticalFlowPyrLK`

Lucas-Kanade sparse optical flow tracks feature points between consecutive frames. The algorithm solves a 2×2 linear system at each feature point using the image gradient and the assumption that nearby pixels have the same motion. You need to understand: what happens when that assumption fails (large motions, occlusion, texture-free regions), what the eigenvalues of the 2×2 matrix tell you about tracking reliability (both eigenvalues should be large; if either is small, the point is on an edge or in a flat region and cannot be tracked reliably), and how to interpret the status output (1 = tracked, 0 = lost). For biohybrid imaging, features placed on the muscle construct boundary will be lost during large contractions; your pipeline must handle this gracefully.

Contour Analysis and Region Properties

Segmenting a muscle construct in each frame and measuring its area, length, or centroid is a common first measurement. `cv2.findContours` followed by `cv2.contourArea` and `cv2.boundingRect` gives you the basic region properties. For more precise measurements, `skimage.measure.regionprops` gives you the equivalent of MATLAB's `regionprops`. The critical point: your segmentation must be validated against a known reference. Do not report a displacement in microns without showing that your pixel-to-micron scale factor was measured from a calibration target with known dimensions.

EXERCISE — Image pipeline

Write a Python function that takes a video of a moving object with a known scale bar, segments the object in each frame using a threshold, tracks its centroid using Lucas-Kanade optical flow, and returns a timeseries of (time, x_{mm} , y_{mm}) where mm values are calibrated from the scale bar. Plot the result. Compute the displacement amplitude and compare it to the known motion amplitude.

Data Structures and Algorithms for Robotics

This is the part most DSAI graduates underestimate. You studied algorithms in courses. Now you need to know them well enough to implement them correctly in a research context where there is

no test suite to catch your bugs.

Graph Representations

A robot's configuration space — the set of all possible joint angle combinations — is often modelled as a graph. A path planning algorithm searches that graph. In Python, you can represent a graph as a dictionary of adjacency lists: `graph[node] = [(neighbour, cost), ...]`. For a 2D grid world (common in robotics demos), each cell is a node and the edges connect it to its 4 or 8 neighbours. For a higher-dimensional configuration space, nodes are sampled configurations and edges are valid transitions. The choice of representation affects the algorithm's speed: a dictionary is fine for sparse graphs; a NumPy adjacency matrix is better for dense graphs where you need to do matrix operations.

BFS, DFS, and Uniform-Cost Search — from Scratch

BFS uses a queue (`collections.deque`; appending to the right and popping from the left is $O(1)$). DFS uses a stack (Python list with `append` and `pop`, both $O(1)$). Uniform-cost search uses a priority queue (`heapq` in Python; `heappush` and `heappop`, both $O(\log n)$). You must be able to implement all three from scratch, not just trace them on a diagram. The differences matter: BFS finds the path with fewest edges (good for unweighted graphs). UCS finds the path with minimum total cost (good for weighted graphs). DFS finds some path but not necessarily the shortest (useful when you just need any solution fast).

EXERCISE — Implement UCS

Implement uniform-cost search from scratch in Python. Input: a graph as `{node: [(neighbour, cost)]}`, a start node, and a goal node. Output: the optimal path and its total cost. Use `heapq` for the priority queue. Test it on the Romania map from AIMA (available in the AIMA Python code repository). Verify that it finds the same path as the textbook. Then profile your implementation on a grid graph of size 100x100 to see where time is spent.

A* Search and Heuristic Design

A* extends UCS by adding a heuristic $h(n)$ that estimates the cost from node n to the goal. The heuristic must be admissible: it must never overestimate the true cost. The straight-line distance heuristic is admissible for any shortest-path problem where the actual path cannot be shorter than the straight line. For a robot arm configuration space, the heuristic might be the joint-angle distance to the goal configuration. A bad heuristic (one that sometimes overestimates) can cause A* to return a suboptimal path. You must understand why: a node expanded with an overestimated f -value may prevent the optimal path from being explored.

The Kalman Filter as an Algorithm

The Kalman filter is often presented as a formula. It is more useful to think of it as an algorithm on two data structures: the state estimate (a vector $\hat{x}$) and the state covariance (a matrix P). The predict step propagates both forward in time using the system dynamics. The update step corrects both using a new measurement. Implement it as a Python class with `predict` and `update` methods. The class stores $\hat{x}$ and P as instance variables. This implementation will be the core of your T1-B project.

EXERCISE — Kalman filter class

Implement a linear Kalman filter as a Python class: `__init__(self, F, H, Q, R, x0, P0)` where F is the state transition matrix, H is the observation matrix, Q is process noise covariance, R is measurement noise covariance. Methods: `predict()` and `update(z)`. Test it on a 1D constant-velocity model: state is [position, velocity], observation is position only. Simulate 100 steps with true trajectory $x(t)=t$, noisy measurements $z(t)=x(t)+\text{noise}$. Plot true trajectory, measurements, and Kalman estimate with uncertainty bands (± 2 sigma).

Writing Research-Quality Python

Code in a research context has a specific requirement that code in industry or coursework does not always have: it must be reproducible by someone else from scratch, possibly months later. This requires specific practices:

Environment management: Use a `requirements.txt` or `environment.yml` file that specifies exact library versions. A result that depends on `numpy 1.24` but cannot be reproduced with `numpy 2.0` is a problem. Use `conda` or `venv` to isolate environments per project.

Seeding random number generators: Any analysis that uses randomness must set a seed at the top: `np.random.seed(42)` and (if using PyTorch) `torch.manual_seed(42)`. Without this, your results change between runs, which makes debugging and reproducing results impossible.

Function signatures and units: Every function that takes physical quantities should name the units in the parameter name or docstring. `def kalman_update(z_mm, R_mm2)` is better than `def kalman_update(z, R)`. A measurement pipeline that silently mixes millimetres and pixels will produce wrong results that are very hard to find.

Testing: Write at least one test for each numerical function. Use known analytical results: the Kalman filter for a zero-noise case should converge to the true state in one step. The optical flow tracker on a rigidly translating image should return the exact translation vector. These tests catch the majority of implementation bugs.

Specific Python Skills You Must OWN Before Applying to ETH

NumPy: array indexing, broadcasting, vectorisation. SciPy: `solve_ivp`, `curve_fit`, `butter+filtfilt`. OpenCV: `calibrateCamera`, `calcOpticalFlowPyrLK`, `findContours`. Algorithms: BFS, UCS, A*, Kalman filter — implemented from scratch, not called from a library. git: branching, committing with meaningful messages, writing a README that lets someone else reproduce your results.

CHAPTER 3

Mathematics: The Specific Topics and How Deeply You Need Them

Mathematics for biohybrid robotics is not a collection of formulas to memorise. Each area of mathematics gives you a specific lens for understanding a specific aspect of the problem. This chapter explains what that lens is for each area.

Linear Algebra: Geometry, Transformation and Uncertainty

The reason linear algebra is central to robotics is that it is the language of spatial relationships. A rotation matrix R describes how one coordinate frame relates to another. A covariance matrix P describes the uncertainty in a state estimate. A Jacobian matrix J describes how a small change in input produces a small change in output. These are not abstract objects: each has a specific physical meaning that you need to understand, not just a computational procedure to follow.

What a Matrix Column Means

The columns of a rotation matrix R are the unit vectors of the rotated frame expressed in the original frame. If R transforms a vector from frame B to frame A , then the first column of R is where frame B 's x-axis points, expressed in frame A 's coordinates. This is not a trick — it is the definition. Understanding this means you can check whether a matrix is a valid rotation matrix (each column should have unit length; columns should be mutually orthogonal; determinant should be $+1$) without memorising a formula.

The Null Space and What It Means for Robots

The null space of a matrix A is the set of vectors x such that $Ax = 0$. For a robot Jacobian J , the null space is the set of joint velocities that produce zero end-effector velocity. This is not a pathological case — it is a resource. A redundant robot (more joints than task space dimensions) has a non-trivial null space, meaning it can move its joints without moving its end-effector. You can use that null space to avoid joint limits, avoid obstacles, or adjust posture. For biohybrid systems, the analogous question is: what stimulation patterns produce no observable change in the measurement? Those are the unobservable directions.

Eigenvalues and Stability

The eigenvalues of a square matrix A tell you the long-term behaviour of the linear dynamical system $dx/dt = Ax$. If all eigenvalues have negative real parts, the system is stable (perturbations decay). If any eigenvalue has a positive real part, the system is unstable (perturbations grow). For a discretised system $x[k+1] = Fx[k]$, stability requires all eigenvalues of F to have magnitude less than 1. This is the mathematical foundation of control stability analysis. For the covariance matrix P in the Kalman filter, eigenvalues are always non-negative (P is positive semidefinite), and their magnitudes tell you the size of the uncertainty ellipse in different directions.

Singular Value Decomposition

SVD decomposes any matrix A as $A = U S V^T$, where U and V are orthogonal matrices and S is diagonal with non-negative entries (the singular values). The singular values tell you how well-conditioned the matrix is: if the largest singular value is much larger than the smallest, the matrix is ill-conditioned, and numerical operations (like solving $Ax = b$) will amplify errors. For camera calibration, ill-conditioning means your estimated camera parameters are unreliable. For system identification, ill-conditioning means you cannot distinguish between different parameter values from your data. The pseudoinverse of A is $V S_{\text{inv}} U^T$, where S_{inv} replaces each non-zero singular value with its reciprocal and zeros remain zero.

EXERCISE — SVD for calibration

Generate 10 3D-2D point correspondences for a pinhole camera with known parameters ($f_x=500$, $f_y=500$, $c_x=320$, $c_y=240$). Add Gaussian noise with $\text{std}=1$ pixel to the 2D points. Set up the linear system $Ap=0$ where p contains the flattened camera matrix (the DLT method). Solve using SVD (`np.linalg.svd`). Compare your recovered parameters to the ground truth. Now increase the noise to 5 pixels and repeat. Report how the estimation error changes and which parameter is most sensitive to noise.

Probability and Statistics: The Language of Uncertainty

Every measurement has noise. Every model has error. Every biological experiment has variability between constructs. Probability theory is the only language precise enough to reason about these uncertainties without making them disappear or ignoring them.

Bayes Theorem — What It Actually Means

Bayes theorem states: $P(\text{hypothesis} \mid \text{data})$ is proportional to $P(\text{data} \mid \text{hypothesis})$ times $P(\text{hypothesis})$. In words: your updated belief in a hypothesis is proportional to how likely the data is under that hypothesis times how likely the hypothesis was before you saw the data. For state estimation: your updated belief in the system state is proportional to how likely your measurement is given that state times how likely that state was before the measurement. This is the foundation of the Kalman filter, the particle filter, and Bayesian parameter estimation.

Gaussian Distributions and Why They Dominate

The Gaussian distribution $N(\mu, \sigma^2)$ is characterised by its mean μ and variance σ^2 . It appears everywhere in practice for three reasons: (1) the central limit theorem says that the sum of many independent random variables approaches a Gaussian, (2) Gaussian noise is a reasonable model for many physical noise sources (thermal noise, quantisation noise, small perturbations), and (3) Gaussian distributions are closed under linear operations — if x is Gaussian, then $Ax + b$ is Gaussian. This last property is what makes the Kalman filter exact: if the state and noise are Gaussian and the dynamics are linear, the state estimate is always a Gaussian whose mean and covariance you can compute exactly.

When Gaussian Fails

Muscle activation has a non-Gaussian distribution in the sense that the activation state can switch between distinct modes (resting, partially activated, fatigued, damaged). A Gaussian distribution cannot represent this multimodal structure. This is precisely why particle filters are needed: they represent the state distribution as a set of weighted samples, which can be multimodal.

Understanding when to use a Kalman filter versus a particle filter is not a matter of algorithmic preference — it is a matter of matching the algorithm's assumptions to the physical reality of your system.

Independent Replication and Why Frames Are Not Samples

This is a statistical mistake that appears constantly in biohybrid research papers and must not appear in yours. A single muscle construct filmed for 1000 frames gives you 1000 images. Those 1000 images are NOT 1000 independent samples. They are 1000 observations of the same construct over time, all affected by the same batch of cells, the same culture conditions, the same handling. The independent unit is the construct. If you have 5 constructs, you have 5 independent samples, not 5000. Your error bars must reflect construct-level variability, not frame-level variability. Any statistical test must use construct-level measurements, not frame-level measurements.

Differential Equations: Modelling Dynamics

Physical systems change over time. Differential equations are the mathematical language for describing how they change. For biohybrid robotics, you need to work with three types:

First-Order Systems: Activation Dynamics

The simplest muscle activation model is $da/dt = (u - a) / \tau$. This is a first-order linear ODE with a forcing term. The solution for a step input $u=1$ starting from $a=0$ is $a(t) = 1 - \exp(-t/\tau)$. The time constant τ determines how fast the activation rises. In practice, $\tau_{\text{activation}}$ is different from $\tau_{\text{deactivation}}$ (calcium is pumped back into the sarcoplasmic reticulum more slowly than it is released). You will use this model in your T1-C project and identify τ from experimental data in T2.

Second-Order Systems: Mechanical Dynamics

A spring-mass-damper model for scaffold mechanics gives $m \ddot{x} + b \dot{x} + kx = F_{\text{input}}$. In state-space form with state $[x, \dot{x}]$, this becomes a 2x2 system. The eigenvalues of the system matrix determine stability and oscillation frequency. Overdamped ($b^2 > 4mk$): two real negative eigenvalues, smooth response. Critically damped ($b^2 = 4mk$): fastest non-oscillatory response. Underdamped ($b^2 < 4mk$): complex eigenvalues, oscillatory response. Biohybrid scaffolds are typically overdamped or critically damped because the hydrogel provides substantial viscous damping.

Nonlinear Systems: Why Linearisation Matters

Most real systems are nonlinear. The muscle force-length relationship is nonlinear (it peaks at the optimal fibre length and decreases on both sides). The activation-to-force mapping is nonlinear (at low activations, a small increase produces a large force change; near maximum activation, the same increase produces little additional force). You analyse nonlinear systems by linearising them around an operating point: compute the Jacobian of the system vector field, which gives you a linear approximation valid near that point. The Jacobian for the EKF is this linearisation applied to the measurement function.

EXERCISE — ODE and linearisation

Consider the pendulum ODE: $\ddot{\theta} = -(g/l)\sin(\theta) - (b/m)\dot{\theta} + (1/ml^2)\tau$. Set $g=9.81$, $l=1$, $b=0.2$, $m=1$. (1) Write this as a 2D state-space system with state $[\theta, \dot{\theta}]$. (2) Find the equilibrium point at $\theta=0$. (3) Linearise around this equilibrium (the linearised system has state matrix $A = \begin{bmatrix} 0 & 1 \\ -g/l & -b/m \end{bmatrix}$). (4) Simulate both the nonlinear and linearised systems from initial condition $\theta=0.1$ rad for 5 seconds. (5) Determine the angular range over which the linearised model is accurate to within 5% in peak amplitude.

CHAPTER 4

Robotics: Beyond the Syllabus, Toward Biohybrid Systems

SSO5201 covers the mechanics of rigid-body robots. This chapter extends that foundation to the specific topics you need for biohybrid research — and explains where rigid-body assumptions break down.

Spatial Transformations: What You Must Really Understand

The homogeneous transformation matrix T is a 4×4 matrix that encodes both a rotation R (upper-left 3×3) and a translation t (upper-right 3×1). The key insight that most students miss: t is the position of the child frame origin expressed in the parent frame coordinates. This means that when you write T_{AB} (transform from B to A), the translation part is the position of the origin of B, expressed in A's coordinate system. Inverting T is not done by matrix inversion (expensive); it is done analytically: $T_{BA} = [[R^T, -R^T t], [0, 0, 0, 1]]$.

For biohybrid imaging, you need this to transform between: the camera frame (where pixel coordinates live), the world frame (where physical dimensions live), and the stage frame (where the microscope stage position is recorded). A calibration procedure establishes $T_{\text{camera_world}}$. Every subsequent measurement computation must apply this transform correctly.

The Jacobian: Connecting Joint Motion to Task Motion

The Jacobian J relates infinitesimal joint angle changes dq to infinitesimal end-effector pose changes dx : $dx = J(q) dq$. This relationship is local — J depends on the current configuration q . Three things you need to understand about J :

Singularities: When J loses rank, some end-effector motions require infinite joint velocity. For a 2-link arm, this happens when the arm is fully extended (elbow at 180 degrees). At that configuration, the end-effector cannot move along the arm's axis using finite joint velocities. Physically, the mechanism has lost one degree of freedom. For biohybrid systems, the analogous situation is when the stimulation input produces no measurable change in the output — the system is at a measurement singularity.

Force mapping: The transpose of the Jacobian maps end-effector forces to joint torques: $\tau = J^T F$. This is useful when you want to estimate the force a muscle construct is producing from a measurement of its displacement and a model of the scaffold's stiffness. If $F_{\text{scaffold}} = k * x_{\text{displacement}}$, then the estimated input force from the muscle is $J^T * F_{\text{scaffold}}$.

Redundancy: When J has more columns than rows (more joint angles than task space dimensions), the inverse kinematics problem has infinitely many solutions. The minimum-norm solution is $\dot{q} = J^+ dx$ where J^+ is the pseudoinverse. Any vector in the null space of J can be added without changing the task motion.

Control: Why PID Is Not Enough

PID control is the right starting point for any control problem. You must understand it before moving to anything else. A PID controller computes the control input $u(t) = K_p * e(t) + K_i * \text{integral}(e) + K_d * de/dt$, where $e(t)$ is the error between desired and actual output. The parameters K_p , K_i , K_d determine the response: K_p determines proportional response (higher K_p = faster response but more overshoot), K_i eliminates steady-state error but can cause integral windup, K_d damps oscillations but amplifies noise.

For biohybrid control, PID has specific failure modes: (1) the actuator dynamics change over time as the tissue fatigues — the controller tuned on a fresh construct may be unstable on a fatigued one; (2) the actuator has a significant dead band (stimulation below a threshold produces no response); (3) the output measurement has delay (the camera frame rate limits how fast you can close the loop). Each of these failure modes motivates a specific extension: adaptive control for time-varying dynamics, threshold-compensating control for dead bands, and predictor-based control for delay.

What You Need to Study Beyond PID

Model Predictive Control (MPC): At each time step, solve an optimisation problem over a finite future horizon to find the optimal control sequence. Apply only the first step, then repeat. MPC handles constraints explicitly (e.g., stimulation amplitude must stay between 0 and maximum safe level). It is computationally more expensive than PID but much more principled for constrained systems. For biohybrid control, the constraints include: maximum safe stimulation frequency (to avoid damage), minimum rest time between stimulation trains (to allow recovery), and maximum displacement (to avoid tearing the scaffold).

System Identification: Before you can design a model-based controller, you need a model. System identification is the process of finding model parameters from experimental input-output data. The simplest approach is least-squares fitting: apply a known input (a sinusoidal sweep or a pseudo-random binary sequence), measure the output, and fit a parametric model (e.g., a first-order transfer function) to the input-output relationship. The fitted model is then used to design the controller or as the prediction model in MPC.

The Biohybrid Control Challenge

Biohybrid actuators are non-stationary systems: their properties change over the course of an experiment as cells consume nutrients, produce metabolic waste, and become fatigued. A controller designed and tuned at time 0 will behave differently at time 2 hours. You need either: (a) a controller that is robust to parameter variations (gain-scheduled PID, or a controller with wide stability margins), or (b) a controller that adapts its model online as the system changes (adaptive control or learning-based control). This is a specific, open research question that your projects can address.

Where Rigid-Body Robotics Fails for Biohybrid Systems

The entire first chapter of SSO5201 assumes rigid bodies: link dimensions do not change, joint angles fully determine the configuration. This assumption breaks down completely for biohybrid

systems. A muscle construct deforms as it contracts. The scaffold compliance means that applied forces produce distributed deformation, not rigid displacement. The tissue-scaffold interface can peel or delaminate under repeated loading.

This does not mean SSO5201 is useless for biohybrid work — it is the foundation. But you need to know where the foundation ends and new ground begins. Specifically: use rigid-body kinematics for the measurement apparatus (camera rig, stage, manipulators). Use continuum mechanics models for the deforming biological construct. The interface between these two models is where the most interesting biohybrid research happens.

CHAPTER 5

Artificial Intelligence: State Estimation, Planning and Learning

SSO5202 covers a broad range of AI topics. For biohybrid research, three areas are specifically important: state estimation (Kalman, particle), decision-making under uncertainty (MDPs, POMDPs), and learning-based methods (when and how to use them).

State Estimation: The Hidden State Problem

In a biohybrid system, you cannot directly measure activation level, fatigue state, or internal force. These are hidden states. You can measure displacement (from a camera) or force (from a transducer) or electrical activity (from electrodes). State estimation is the problem of inferring the hidden state from these noisy, indirect observations.

The Kalman Filter: When It Works and When It Does Not

The Kalman filter is the optimal estimator for systems where: (1) the dynamics are linear ($x[k+1] = Fx[k] + Bu[k] + \text{noise}$), (2) the observation is linear ($z[k] = Hx[k] + \text{noise}$), (3) all noise is Gaussian. Under these conditions, the Kalman filter gives you the minimum mean-squared-error estimate of the state. You need to understand the predict and update steps not just as formulas but as geometric operations: the predict step propagates the uncertainty ellipse forward ($P = FPF^T + Q$), and the update step shrinks the uncertainty ellipse in the direction of the measurement (using the Kalman gain $K = PH^T(HPH^T + R)^{-1}$).

For biohybrid work, the linear Kalman filter works well for: tracking displacement of a marker under small-amplitude oscillation, estimating velocity from noisy position measurements, and fusing estimates from two sensors with different noise levels.

The Extended Kalman Filter: Linearisation and Its Limits

When the dynamics or observation model are nonlinear, the EKF linearises them using the Jacobian at the current state estimate. The linearisation is valid only locally: if the true state is far from the estimate (e.g., after a large perturbation or a missed measurement), the Jacobian may be computed at the wrong point, and the filter may diverge. Divergence means the filter's covariance matrix collapses (becomes too small) while the true error grows. This is a real failure mode, not just a theoretical concern. In practice, you detect it by monitoring the innovation sequence (the difference between the predicted and actual measurement): if innovations are consistently larger than the innovation covariance predicts, the filter has diverged.

Particle Filters: Handling Multimodal Uncertainty

The particle filter represents the state distribution as a set of N weighted samples (particles). Each particle is one hypothesis about the current state. The update step reweights particles by how likely

the measurement is under each hypothesis. The resample step eliminates low-weight particles and duplicates high-weight ones. The advantage is that the distribution can be any shape — bimodal, heavy-tailed, truncated. The disadvantage is computational cost: a 2D state might need 1000 particles, a 10D state might need millions. For biohybrid work, the particle filter is appropriate when the actuator can be in qualitatively different states (e.g., healthy vs. fatigued vs. damaged) that cannot be represented by a single Gaussian.

MDPs and the Stimulation Decision Problem

Consider the problem of deciding how to stimulate a biohybrid actuator to achieve a target displacement while avoiding tissue damage. This is a sequential decision problem under uncertainty: the current state (activation level, fatigue, displacement) is partially observed; each action (stimulation pattern) leads to a new state; the goal is to maximise total reward (displacement achieved) minus total cost (tissue damage, energy consumed). This is precisely the formulation of a Markov Decision Process.

The Bellman equation gives the optimal value function $V^*(s) = \max_a [R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')]$. For a biohybrid stimulation problem, s is the discretised state (activation, fatigue, current length), a is the stimulation command, R is the reward (progress toward target minus damage cost), and $P(s'|s,a)$ is the transition model (how the state changes in response to stimulation). Computing V^* by value iteration gives you the optimal policy: for each state, which stimulation command produces the best expected outcome.

The POMDP extends this to partial observability: you maintain a belief distribution $b(s)$ over states, and the optimal policy is a function of the belief rather than the state. For biohybrid systems, this is the right formulation: you do not directly observe fatigue or activation; you observe displacement and perhaps force. The belief updates using Bayes theorem, exactly like the state estimation algorithms above.

When and How to Use Machine Learning

Machine learning for biohybrid research is not about training large neural networks. It is about using data-driven methods when the physics-based model is inadequate. Specifically:

Use classical ML (Gaussian processes, random forests, linear regression) when: you have a small dataset (fewer than 1000 samples), you need uncertainty estimates, or you need interpretability. Gaussian process regression gives you a mean prediction and a confidence interval, which is essential for research applications. A random forest can tell you which features are most important, helping you understand the system.

Use deep learning when: you have a large dataset, the input has high dimensionality (images), and prediction accuracy is more important than interpretability. A CNN for segmenting muscle constructs in microscopy images is a reasonable use of deep learning. A multi-layer fully connected network for predicting force from a 5-feature activation model is not — a linear model or Gaussian process would perform better and be more interpretable.

Use residual learning (physics + ML) when: you have a physics-based model that captures most of the dynamics but leaves a systematic residual error. Train an ML model to predict that residual. This is currently the most credible approach for biohybrid modelling because it preserves interpretability of the physics component while improving accuracy.

CHAPTER 6

Biology and Materials: What an Engineer Must Know

You do not need to become a biologist. You need to understand the living system well enough to design experiments, interpret results, and identify the biological phenomena that make your computational models fail.

Muscle Physiology: The Specific Concepts

Skeletal muscle generates force through a process called excitation-contraction coupling. An electrical stimulus triggers an action potential that propagates along the cell membrane and into the T-tubule system. This releases calcium ions from the sarcoplasmic reticulum. Calcium binds to troponin, which allows myosin heads to bind to actin filaments. The myosin heads undergo a conformational change (the power stroke), pulling the actin filaments and shortening the sarcomere. This is the molecular basis of muscle contraction.

For a control engineer, the important quantities are: the time from stimulation to peak force (the activation delay, 10-50 ms depending on the construct), the force-length relationship (force is maximum at the optimal sarcomere length and decreases on both sides), the force-velocity relationship (force decreases as shortening velocity increases, and increases above isometric during lengthening), and the fatigue dynamics (repeated stimulation causes calcium accumulation and then depletion, leading to progressive force decline).

For your models: the activation delay motivates a first-order activation ODE. The force-length relationship gives you the $f_l(x)$ term. The force-velocity relationship gives you the $f_v(v)$ term. The fatigue dynamics motivate an additional state variable in your model, with its own time constant (typically tens of seconds to minutes, much slower than activation). Ignoring any of these while claiming your model is physiologically grounded is a scientific error.

The Optogenetics Approach: What It Is and Why It Matters

Conventional electrical stimulation requires electrodes in contact with the tissue, which introduces infection risk and limits the spatial precision of stimulation. Optogenetics is an approach where light-sensitive ion channels (opsins, most commonly channelrhodopsin) are genetically expressed in the muscle cells. Shining light of the appropriate wavelength directly activates these channels, triggering an action potential without physical contact. This allows spatially patterned stimulation (illuminate one region of the construct to make it contract more than another), fibre-optic delivery through a bioreactor, and non-invasive repeated stimulation.

The ETH SRL has published work on optogenetically controlled biohybrid robots (the 2016 Raman et al. paper is the canonical reference). Understanding this approach is important for your PhD application because it represents the state of the art in biohybrid stimulation. When you describe

your research interests, you should be able to explain what optogenetics enables that electrical stimulation does not.

Scaffold Materials and Their Mechanical Properties

The scaffold is the engineered structure on which the muscle cells are grown. Its mechanical properties must be carefully chosen: if the scaffold is too stiff, the cells cannot remodel it and will not mature properly; if it is too compliant, it deforms too much under the cells' resting tension and does not provide a stable anchor for force transmission.

The most common scaffold materials for skeletal muscle biohybrid robots are fibrin gels and Matrigel-collagen composites. Their stiffness is in the range of 1-10 kPa (for comparison, steel is 200 GPa and soft brain tissue is 0.1-1 kPa). You characterise scaffold mechanics using rheology (measuring storage and loss moduli G' and G'') or tensile testing (measuring the force-displacement curve and fitting a constitutive model). The simplest constitutive model is the neo-Hookean hyperelastic model, which has a single parameter (the shear modulus G) and is appropriate for small strains in isotropic hydrogels.

For your research: you do not need to fabricate scaffolds. You need to be able to read a methods section that describes the scaffold and extract the relevant mechanical parameters for your model. You need to understand why a 1 kPa scaffold produces different results than a 5 kPa scaffold even if everything else is the same.

Experimental Design and Biological Variability

This is where many engineering papers on biohybrid systems fail. The key concept: biological variability is not noise; it is a feature of the system that must be characterised and reported.

Two constructs made from the same cell line on the same day using the same protocol will produce different forces. This is because the number of cells that successfully differentiated into contractile fibres varies, the alignment of the fibres varies, and the degree of maturation varies. Your statistical analysis must reflect this: you cannot compute the mean and standard deviation of 1000 displacement measurements from one construct and report them as if they describe construct-to-construct variability. You need multiple independent constructs (typically $n \geq 3$, and for anything published, $n \geq 6$).

Specifically: compute your summary statistics (mean force, mean displacement amplitude, contraction frequency) separately for each construct. Then compute the mean and standard deviation of these per-construct statistics. This correctly propagates the biological variability.

CHAPTER 7

Six Concrete Projects Across Six Trimesters

Each project has a specific technical question, a specific implementation, specific validation criteria, and a specific connection to biohybrid research. Projects build on each other: T1 projects feed into T2, and so on. Start T1-A in your first week.

Trimester 1 Projects (Sep-Dec 2026)

All three T1 projects can run on your laptop with no biological materials. They use synthetic data, standard libraries, and your own implementations. The goal is to build the measurement, estimation, and modelling pipeline that will later be applied to real experimental data.

T1-A: Optical Flow Pipeline for Muscle Contraction

The question: Can you measure the contraction dynamics of a simulated muscle construct from a synthetic video, and report the displacement, shortening fraction, and contraction frequency with calibrated uncertainty?

What to build: Generate a 200-frame synthetic video (30 fps) of a rectangular object that oscillates sinusoidally at 1 Hz with amplitude 10 pixels. Add Gaussian noise, motion blur, and a simulated scale bar (a horizontal line of known length). Implement the full measurement pipeline: (1) camera calibration using the scale bar (a scale factor, not a full camera matrix, since this is 2D), (2) Lucas-Kanade tracking of corner features placed on the object, (3) computation of displacement timeseries in millimetres from calibrated pixels, (4) frequency estimation using Welch's power spectral density, (5) amplitude estimation from the PSD peak. Report the measurement error as $(\text{measured} - \text{true}) / \text{true}$ for both frequency and amplitude.

Validation criteria: Frequency estimation error below 1%, amplitude estimation error below 5%, with noise std=2 pixels. If you cannot meet these criteria, your pipeline has a bug or a calibration error. Fix it before claiming the project is done.

Where it leads: This pipeline is the measurement front-end for all subsequent projects. In T2, you will apply it to real experimental video. In T3, it will feed into the closed-loop controller.

T1-B: Kalman Filter for Actuator State Estimation

The question: Can you estimate the position and velocity of a simulated actuator from noisy displacement measurements, with realistic handling of missing measurements and sensor dropouts?

What to build: Implement the Kalman filter class described in Chapter 2. Use a constant-velocity state model: $\text{state} = [\text{position}, \text{velocity}]$, $F = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}$, $Q = \sigma_{\text{process}}^2 * \begin{bmatrix} dt^4/4 & dt^3/2 \\ dt^3/2 & dt^2 \end{bmatrix}$. Apply it to the displacement timeseries from T1-A. Add dropouts (randomly set 10% of measurements to missing). Compare three estimators: raw measurement, causal moving

average (window=5 frames), and your Kalman filter. Metrics: RMS position error, RMS velocity error (velocity estimated as finite difference of true position), and coverage of the 95% confidence interval (should be approximately 95%). A confidence interval that covers 30% of true values is not a 95% confidence interval: your R matrix is wrong.

Critical detail: Handle missing measurements correctly. When a measurement is missing, run the predict step but skip the update step. The uncertainty (covariance) should grow during missing-measurement periods and shrink when measurements return.

T1-C: Muscle Activation Model and Control

The question: How do the parameters of a first-order muscle activation model affect the force response, and can a simple PID controller track a target displacement in simulation?

What to build: Implement the two-state muscle model from Chapter 2's SciPy exercise. Add three explicit parameters: `tau_act` (activation time constant), `tau_deact` (deactivation time constant), and `F_max` (maximum isometric force). Write the model as a Python class with a `step` method that takes the current state and stimulation input and returns the new state and force output. Then add a PID controller that closes the loop: the controller reads the current displacement and outputs a stimulation command (clamped to $[0, 1]$). Run step-response and sinusoidal tracking experiments. Show what happens when you increase K_p beyond the stability limit. Show what integral windup looks like and how anti-windup prevents it.

Label everything honestly: This is a simulation with hypothetical parameters. The parameters are not calibrated to any specific construct. The model does not include biological variability. These limitations must be stated in your project report.

Trimester 2 Projects (Jan-Apr 2027)

T2 projects extend T1 work toward real data and calibrated hardware. By T2 you should have access to a physical compliant object (a rubber strip, a piece of hydrogel, or any elastomeric material) if not to actual biological tissue.

T2-A: System Identification from Experimental Data

The question: Given input-output data from a physical compliant system, can you fit the parameters of the muscle model from T1-C and predict its response to a new input?

What to build: Film a rubber strip being stretched and released repeatedly with known timing. Apply the T1-A pipeline to extract displacement timeseries. Then fit the two-state model parameters (`tau_act`, `tau_deact`, `F_max`, `k`, `b`) using `scipy.optimize.minimize` with an L-BFGS-B method. Split your data: use 70% for fitting, 30% for validation. Report the fit quality on the training data and the prediction quality on the validation data. A model that fits perfectly but predicts poorly is overfit: add regularisation or reduce the number of free parameters.

T2-B: Cross-Construct Variability

The question: How much do the fitted model parameters vary across repeated trials of the same system, and which parameters are most consistent?

What to build: Repeat the T2-A identification on 5 different trials (or 5 different stretching amplitudes). Fit the model independently for each trial. Report the mean and standard deviation of each parameter. Identify which parameters are tightly constrained (low CV = coefficient of variation) and which are poorly identified (high CV). Plot the distribution of each parameter across trials using a strip plot or violin plot. This quantifies the biological/experimental variability and tells you which parameters you can treat as fixed across experiments versus which need to be re-identified each time.

Trimester 3 Projects (May-Aug 2027)

T3-A: Closed-Loop Control on a Physical Compliant System

The question: Can you implement and validate a feedback controller that maintains a target displacement in a physical compliant system in real time?

What to build: Connect the T1-A measurement pipeline to the T1-C controller in a real-time loop. The camera acquires a frame, the optical flow pipeline estimates displacement, the Kalman filter smooths it, the PID controller computes the control action, and a servo or actuator applies the action. This is a real-time control loop with the camera as the sensor. The key engineering challenge is latency: the camera has a frame delay, the optical flow computation takes time, and the Kalman filter adds no delay (it is causal). Measure the total latency from physical motion to control action and quantify its effect on tracking performance.

Trimester 4-6: Research Quality Projects

By T4 you should have access to experimental data from a biological construct (either your own institution's lab or through a collaboration). T4-A involves reproducing one specific measurement from a published biohybrid paper (e.g., the displacement timeseries from Figure 2 of a specific paper, using the described stimulation pattern). T4-B involves applying your T2 identification pipeline to real biological data and comparing parameter estimates across constructs from the same cell line. T5-A involves implementing and validating an adaptive controller that updates its model estimate online as the construct fatigues. T6 is the thesis synthesis and ETH application preparation.

CHAPTER 8

The ETH PhD Path: What to Build and How to Present It

Getting a PhD call from ETH SRL requires demonstrating genuine technical depth, relevant research experience, and a clear understanding of what the lab does and how you would contribute. This chapter tells you what that looks like specifically.

What ETH SRL Wants to See in an Application

The lab works on four directions. For the biohybrid direction, they need candidates who can either: build and characterise tissue-integrated sensing and actuation systems (wet lab + instrumentation), or develop computational models and controllers for biohybrid actuators (dry lab + programming). You are building toward the second profile. That means your application needs to demonstrate:

- 1. A working implementation that does something real.** Not a course assignment, not a tutorial reproduction. A project where you defined the question, implemented the solution, validated it with quantitative metrics, identified failure modes, and documented everything reproducibly. Your T1-A pipeline, rigorously validated, is more impressive than a half-finished neural network.
- 2. Mathematical understanding, not just code.** In the application or interview, you need to explain your Kalman filter from first principles. You need to explain why your PID controller became unstable at a certain gain. You need to explain what the singular values of your system identification data matrix tell you about parameter identifiability. If you can only say "I used the library function," you have not demonstrated the depth they need.
- 3. Biological literacy.** You do not need to have grown muscle cells. You need to demonstrate that you understand the system you are trying to control. In your research statement, explain: what the muscle force-length curve is, why it matters for your controller design, and what biological variability means for the system identification approach you are proposing.
- 4. A specific, honest research proposal.** Not "I want to use AI for biohybrid robots." Specifically: "I propose to develop an adaptive Kalman filter for estimating the activation state of a muscle-actuated biohybrid system from optical displacement measurements, with online model updating to track fatigue-induced parameter changes. I will validate this on a published experimental dataset and extend it to real-time closed-loop control in simulation." That is a proposal that demonstrates you understand the problem and have a plan.

Topics You Must Study That Are Outside the Official Syllabus

The MTech RAI syllabus covers fundamentals. For the ETH PhD direction, you additionally need:

Continuum mechanics basics: stress, strain, constitutive models (neo-Hookean, Kelvin-Voigt, Maxwell). Not to derive them from scratch but to read a materials science paper and understand

what the parameters mean. Recommended: Ogden "Non-Linear Elastic Deformations" Chapter 1, or the simpler Cowin and Doty "Tissue Mechanics" Chapter 2.

Fluorescence microscopy: the difference between brightfield, phase contrast, and fluorescence imaging; what GFP is and why it is used; what DAPI staining shows; what confocal imaging adds. You need this to read biohybrid papers that describe their imaging setup and understand what they are actually measuring. No lab access required: read the Nikon MicroscopyU tutorials online.

Robust control basics: H-infinity control and the small gain theorem at a conceptual level. The specific idea: if your model has bounded uncertainty (e.g., the time constant can vary by plus or minus 20%), you can design a controller that is guaranteed to be stable for all models within that uncertainty bound. This is the mathematical foundation for controllers that remain stable as the biohybrid actuator parameters drift.

Experimental design for biological systems: Latin square designs, randomised block designs, sample size calculation. Specifically: before running an experiment, you should be able to calculate how many constructs you need to detect a 20% improvement in force output with 80% statistical power. This shows methodological sophistication that distinguishes engineering papers from biology papers.

Scientific writing: The structure of a methods section, a results section, and a discussion section. Specifically: a methods section must be detailed enough that another researcher could reproduce your experiment. A results section presents data without interpretation. A discussion section compares your results to prior work, explains unexpected findings, and acknowledges limitations honestly. Practice this by writing one-page summaries of your T1 projects in this format.

The Most Important Thing

ETH SRL does not primarily hire based on what you know. They hire based on evidence of what you can do independently. A GitHub repository with clean, documented, tested code for your T1-A pipeline is more convincing than a list of courses you took. A one-page project report that honestly states what you measured, what you found, and what the limitations are, is more convincing than a portfolio of unfinished demos.

Your Preparation Timeline

Now to Dec 7: Run the T1-A pipeline. Implement the Kalman filter class. Complete the T1-C model and control simulation. Pass the three exams with strong understanding of SSO5201, SSO5202, SSO5203.

Dec 8 to Apr 2027: Apply T1-A to real video (film a rubber strip or any moving object with a scale bar). Complete T2-A system identification. Start reading one biohybrid paper per week (start with Cvetkovic et al. 2014 and Raman et al. 2016). Identify a supervisor at your institution or a collaborating institution who has access to biological tissue.

May to Dec 2027: Complete T3-A closed-loop control. Begin T4-A reproduction study. Build a GitHub portfolio with clear README files for each project. Write one 4-page research report in conference paper format.

Jan to Aug 2028: Complete thesis research. Write the MTech thesis. Prepare ETH application: research statement, CV, two referee letters, transcript. Monitor ETH SRL open positions page monthly.

Key References

R E F	TITLE	ACCESS	PURPOSE
[1]	Cvetkovic et al. PNAS 2014	pmc.ncbi.nlm.nih.gov/articles/PMC4104884/	READ FIRST — foundational bio-bot paper
[2]	Raman et al. PNAS 2016	pmc.ncbi.nlm.nih.gov/articles/PMC4822586/	READ SECOND — optogenetic bio-bot
[3]	Lynch and Park Modern Robotics	hades.mech.northwestern.edu/index.php/Modern_Robotics	Kinematics, dynamics, control
[4]	Russell and Norvig AIMA	aima.cs.berkeley.edu/	Search, probabilistic inference, MDPs
[5]	Szeliski Computer Vision	szeliski.org/Book/	Image formation, calibration, motion
[6]	OpenStax Anatomy and Physiology	openstax.org/books/anatomy-and-physiology-2e	Muscle contraction chapter
[7]	Welch and Bishop Kalman Tutorial	cs.unc.edu/~welch/media/pdf/kalman_intro.pdf	Readable Kalman derivation
[8]	Python Control Library docs	python-control.readthedocs.io	Control systems in Python
[9]	ETH SRL Open Positions	srl.ethz.ch/open-positions.html	Monitor monthly from T4 onward
[10]	IITG MTech RAI Syllabus	iitg.ac.in/oes/odp/mtech/rai/MTech_RAI_Curriculum_Syllabus.pdf	Official course scope