Simulating Mechanical Transfer in Rowing

From Thrust–Drag Energy Balance to Disturbance–Efficiency Coupling via Multibody Dynamics and IMU-Informed System Identification

Gabriel Cirker

December 2025

Abstract

Rowing measurement culture is exceptionally good at quantifying *power*. Yet elite performance depends at least as strongly on the *cost of power*: losses induced by velocity fluctuations, hull attitude dynamics (pitch/trim), and recovery-phase mass motion that increase effective drag and reduce how much propulsive work becomes uninterrupted run. This paper presents a publication-oriented modeling framework for mechanical energy transfer in rowing. We develop a cycle-resolved thrust?drag model of an eight (8+) with periodic drive/recovery forcing and quadratic drag, enabling transparent energy accounting of how rowers? mechanical work partitions into drag dissipation and kinetic energy while reproducing within-stroke velocity oscillations and their sensitivity to drag, mass, and stroke rate. We then extend the dynamics to a reduced-order multibody formulation including crew center-of-mass motion on the slide and a pitch surrogate coupled to effective drag, and we outline an IMU-informed signal-processing and grey-box system-identification pipeline to infer disturbance-sensitive parameters from accelerometer/gyro measurements. The framework reframes performance optimization as loss minimization: reducing disturbance-induced drag augmentation and velocity-check penalties can deliver meaningful speed gains without increasing peak power. We conclude with methodological pathways?validated IMU datasets, coupled CFD/ROM drag models, and program-scale system identification?toward measuring and minimizing technique-driven losses before on-water seat trials.

Keywords: rowing; hydrodynamic drag; velocity fluctuations; multibody dynamics; inertial measurement units; system identification; efficiency; stroke segmentation.

Contents

List of Figures

List of Tables

1 Introduction

Rowing propulsion is intermittent: each stroke alternates a high-force drive with a largely unforced recovery during which the shell coasts. This intermittency imposes an energetic tax. Because hydrodynamic drag grows approximately with the square of boat speed, within-stroke velocity oscillations raise the time-averaged resistive work required to maintain a given average speed; equivalently, for fixed athlete power, oscillations reduce average speed (Hill and Fahrig, 2009; Kleshnev, 2010). The physics is simple but consequential: when the boat is "checked" during recovery, the subsequent drive must both overcome drag and re-accelerate the system back toward racing speed.

Rowing culture measures what is easy to measure. Ergometer outputs provide high-fidelity estimates of mechanical power at the handle and allow standardized comparisons across athletes and seasons. Yet on-water performance depends on additional variables that are not directly visible in power-only metrics: (i) how rower mass motion redistributes momentum within the boat, (ii) how recovery-phase timing and smoothness modulate pitch/trim and associated drag augmentation, and (iii) how well individual movements synchronize into a low-disturbance platform so that drive-phase work is preserved as uninterrupted run. These effects are widely discussed qualitatively by coaches ("set the boat," "no check," "quiet hands"), but they are rarely quantified early enough in the selection cycle to influence lineup formation.

The performance consequence is a reframing of optimization. Increasing peak power is valuable, but many of rowing's remaining marginal gains may come from reducing loss channels that prevent power from becoming speed. In particular, technique-dependent disturbances can increase effective drag (through attitude dynamics and flow separation) and can increase the amplitude of velocity fluctuations, both of which raise the energetic cost of a given average speed (Pulman, 2003a; Day et al., 2011). This is most salient in disturbance-sensitive configurations: small pitch/trim changes can disproportionately affect the flow around the bow and the wave-making component of resistance, and movements tolerated elsewhere can become amplified at the front of the boat.

This paper combines and extends two complementary studies into a single modeling and analysis framework.

models the boat+crew as a single lumped mass subject to periodic thrust pulses during the drive

transparent energy accounting and reproduces characteristic within-stroke velocity oscillations. We analyze sensitivity to drag coefficient, total mass, and stroke rate and quantify how much input work is dissipated by drag versus stored as kinetic energy (Buckmann and Harris, 2014; Sanderson and Martindale, 1986).

crew center-of-mass motion on the slide and a pitch surrogate coupled to effective drag, and an

IMU-informed analysis pipeline (orientation estimation, gravity removal, stroke segmentation, time–frequency features) together with a grey-box system identification (SysID) formulation. These elements expose technique-dependent loss channels and provide a route to estimating disturbance-sensitive parameters from accelerometer/gyro data (Worsey et al., 2019a; Geneau et al., 2024; Groh et al., 2014).

computational fluid dynamics (CFD) or fully parameterized musculoskeletal models. Instead, it targets an intermediate regime where equations remain interpretable, parameters can be inferred from data, and the model can support large-scale sweeps and identification studies. We therefore emphasize (i) explicit energy budgets, (ii) modular extensions that map directly to measurable signals (IMU acceleration/gyro), and (iii) forward-looking integration pathways, including CFD-informed drag augmentation surrogates and program-scale parameter estimation.

1.1 Contributions

The main contributions are:

1. A unified baseline thrust–drag simulation with explicit energy accounting, parameter sweeps, and updated (non-outdated) figure generation from reproducible Python code.
2. A multibody reduced-order extension that incorporates crew slide kinematics and a pitch-coupled drag surrogate, enabling direct quantification of recovery-phase disturbance costs.
3. An IMU-based analysis pipeline and a grey-box system identification formulation that links accelerometer/gyro signals to technique-sensitive model parameters.
4. A forward-looking methodology section describing how public race/erg data and accessible sensor logging (e.g., Concept2 exports, World Rowing APIs) can support validation datasets and scaling studies (Concept2, 2025a; World Rowing, 2025).

1.2 Reproducibility

All figures in this manuscript are generated by the Python code included in the project repository (Appendix E). The code is organized to allow: (i) deterministic regeneration of paper figures, (ii) parameter sweeps for sensitivity analysis, and (iii) synthetic IMU generation and grey-box fitting demonstrations. The implementation is designed to be easily extended to real IMU data when available.

2 Background and Related Work

2.1 Drag, power scaling, and the energetic penalty of speed fluctuations

For a shell moving at racing speeds, hydrodynamic resistance dominates the external load. A convenient reduced model treats resistance as quadratic in speed,

$$F_d(v) = k_d v^2, \tag{1}$$

where k_d aggregates wetted area, Reynolds-number-dependent friction, form drag, and wave-making contributions into an effective coefficient. Quadratic scaling is consistent with textbook fluid mechanics and is widely used in rowing models and coaching analyses (Pulman, 2003a; Dudhia, 2007a; Buckmann and Harris, 2014). Under quadratic drag, the power required to maintain speed scales as $P \sim v^3$, implying that small speed gains can require disproportionate increases in power output.

A key implication is that *fluctuations* in speed impose a nonlinear penalty. If $v(t)$ varies around mean $\bar{v}$, then $\overline{v^3} \geq \bar{v}^3$ (Jensen's inequality). Thus, for a fixed average speed, a fluctuating velocity profile requires more power than a constant-speed profile; for fixed power, fluctuations reduce average speed. Hill and colleagues quantified the impact of velocity fluctuations on race time using a mathematical approximation of the increased resistive work (Hill and Fahrig, 2009). Kleshnev further analyzed the temporal structure of boat acceleration and performance, showing links between stroke-level kinematics and race outcomes (Kleshnev, 2010). These findings motivate modeling and measurement strategies that target fluctuation reduction as a performance lever.

2.2 Recovery-phase mass motion and "check"

Within each stroke, the boat experiences acceleration during the drive and deceleration during recovery. During recovery, the crew's center of mass moves relative to the hull on the sliding seats. In the hull frame, this internal motion can induce reaction forces that decelerate the boat ("check"), especially when mass motion is abrupt. Classic technique discussions emphasize smoother recovery and controlled slide speed to minimize checking. Sanderson and Martindale's seminal analysis of rowing technique emphasized optimizing the temporal distribution of force and controlling recovery kinematics to reduce losses (Sanderson and Martindale, 1986). Smith and Loschner discussed instrumentation and feedback for linking technique variables to boat speed determinants and provided evidence that recovery structure materially affects performance-relevant dynamics (Smith and Loschner, 2002).

Mechanistic modeling of recovery effects spans a range of fidelities. Reduced-order models often represent thrust as a periodic forcing and drag as a simple function of speed, yielding analytically tractable dynamics (Pulman, 2003a). Higher-fidelity approaches incorporate rower mass motion on the slide and sometimes oar-blade hydrodynamics and 3D multibody dynamics, enabling richer predictions of velocity oscillation amplitude and phase relationships (Mola, 2010; Formaggia et al., 2010). The present work adopts an intermediate approach: we add a minimal internal degree of freedom (crew slide displacement) and a pitch surrogate to expose disturbance channels while retaining interpretability and enabling system identification.

2.3 Coordination, antiphase rowing, and velocity fluctuation mitigation

If propulsion were continuous, velocity fluctuations would be reduced. This motivates (at least conceptually) asynchronous coordination strategies where different rowers produce thrust out of phase. De Brouwer *et al.* studied antiphase interpersonal coordination on ergometers and found that reduced velocity fluctuations decreased power loss, increasing the fraction of useful power transfer (Brouwer et al., 2013). Cuijpers *et al.* similarly analyzed crew coordination and performance effects (Cuijpers et al., 2015). Despite these findings, antiphase rowing is not widely adopted on water, likely due to stability, balance, and timing constraints. In this paper we assume synchronous rowing, but the model structure supports future exploration of asynchronous forcing functions.

2.4 From measurement to models: IMUs, segmentation, and performance monitoring

Inertial measurement units (IMUs) provide a practical path to quantifying otherwise hidden technique variables. A systematic review by Worsey *et al.* surveyed inertial sensor technology in rowing, emphasizing best practices for sensor placement, signal processing, and interpretation (Worsey et al., 2019a). Recent work has advanced phase detection and kinematic reconstruction from limited sensors; for example, Geneau *et al.* proposed an undecimated wavelet transform approach for detecting stroke kinematic events from a hull-mounted accelerometer across boat classes (Geneau et al., 2024). In parallel, signal processing approaches such as subsequence dynamic time warping (subDTW) have been used for stroke detection and prediction in noisy acceleration streams (Groh et al., 2014).

On land, ergometers provide standardized power and stroke metrics, and modern performance monitors allow export of detailed stroke data. Concept2 supports exporting workouts with stroke data and provides developer documentation for its logbook API (Concept2, 2025b; Concept2, 2025a). On water, public performance data sources (e.g., World Rowing documents and APIs) provide race splits and boat-class metadata that can support population-scale comparisons and validation studies (World Rowing, 2025). These data channels motivate the forward-looking identification and validation pathways proposed in Sections 7 and 9.

2.5 Scope and modeling philosophy

Rather, the aim is to create a modular reduced-order framework that (i) preserves the dominant physics of energy dissipation and momentum redistribution, (ii) makes disturbance-sensitive loss channels explicit and quantifiable, and (iii) is amenable to parameter estimation from IMU signals. This "physics-first" approach supports interpretability, sensitivity analysis, and rapid iteration, while remaining extensible to higher fidelity through CFD-informed surrogate drag models and richer multibody representations.

3 Part I: Baseline Cycle-Resolved Energy-Transfer Model

Part I consolidates the original study "Simulating Mechanical Transfer in Rowing" into a publication-ready baseline model. The objective is not maximum realism; it is an analytically transparent model that makes energy pathways explicit and generates interpretable sensitivity results.

3.1 One-dimensional surge dynamics

We model the shell and crew as a lumped mass m moving along the direction of travel with forward speed $v(t) \geq 0$. External forces are (i) a propulsive thrust $F_{\text{thrust}}(t)$ applied during the drive and (ii) an effective hydrodynamic resistance $F_d(v)$.

$$m\,\dot{v}(t) = F_{\text{thrust}}(t) - F_d\big(v(t)\big). \tag{2}$$

For baseline analysis we adopt a near-quadratic resistance law,

$$F_d(v) = k_d\,v^2, \tag{3}$$

consistent with canonical rowing physics treatments and effective-fit approaches in the literature (Pulman, 2003b; Dudhia, 2007b; Baudouin and Hawkins, 2002).

3.2 Stroke-periodic thrust model

Let T denote the stroke period (seconds per stroke) and $\delta \in (0, 1)$ the drive fraction. Define phase $\phi(t) = (t \bmod T)/T$. The square-wave thrust is

$$F_{\text{thrust}}(t) = \begin{cases} F_{\text{drive}}, & 0 \leq \phi(t) < \delta, \\ 0, & \delta \leq \phi(t) < 1. \end{cases} \tag{4}$$

To soften discontinuities and permit differentiable parameter estimation, we also define a smooth thrust curve on the drive interval using a scaled Beta-density shape

$$F_{\text{thrust}}(t) = F_0 \,\text{BetaPDF}\left(\frac{\phi}{\delta};\, a, b\right) \mathbb{1}_{[0,\delta)}(\phi), \tag{5}$$

where $a, b > 1$ control the rise/peak/fall shape and F_0 scales total impulse. This family captures early-peaking, mid-peaking, and late-peaking force profiles while remaining compactly supported on the drive.

3.3 Cycle-resolved energy accounting

We track (i) input work from thrust, (ii) dissipated work against resistance, and (iii) kinetic-energy change. Over an interval $[t_i, t_f]$,

$$\begin{aligned} E_{\text{in}} &= \int_{t_i}^{t_f} F_{\text{thrust}}(t)\, v(t)\, dt, & (6)\\ E_{\text{drag}} &= \int_{t_i}^{t_f} F_d\big(v(t)\big)\, v(t)\, dt = \int_{t_i}^{t_f} k_d\, v(t)^3\, dt, & (7)\\ \Delta E_k &= \tfrac{1}{2} m \big(v(t_f)^2 - v(t_i)^2\big). & (8) \end{aligned}$$

By the work–energy theorem in this reduced system,

$$E_{\text{in}} = E_{\text{drag}} + \Delta E_k. \tag{9}$$

We report a transient "mechanical transfer ratio"

$$\eta(t_i, t_f) = \frac{\Delta E_k}{E_{\text{in}}}, \tag{10}$$

noting that over a full stroke in a strict periodic steady state $\Delta E_k \approx 0$ and the ratio is not a useful measure of propulsive efficiency. The more informative quantities are cycle-averaged speed, speed-oscillation amplitude, and the partition of energy into dissipation versus transient storage.

3.4 Baseline parameters

Baseline parameters are chosen to produce race-pace speeds for an eight (8+) and to align with typical stroke timing. Table 1 summarizes the nominal values used throughout Part I.

Table 1: Baseline Part I parameters.

Parameter	Symbol	Nominal value
Total mass (boat + crew)	m	$750\,\mathrm{kg}$
Drag coefficient (effective)	k_d	$40\,\mathrm{N\,s^2\,m^{-2}}$
Stroke rate	r	$30\,\mathrm{spm}$
Stroke period	T	$2.0\,\mathrm{s}$
Drive fraction	δ	0.45
Peak drive thrust	F_{drive}	$3000\,\mathrm{N}$
Time step (integration)	Δt	$1 \times 10^{-3}\,\mathrm{s}$

3.5 Numerical integration

The implementation uses explicit time stepping for transparency. We also provide an upgraded integrator (adaptive ODE solving and differentiable forcing options) in `rowing_model.py`, while retaining an Euler mode for comparison and pedagogy. The code records speed, thrust, acceleration, and the energy integrals at each time step.

3.6 Implementation and reproducibility

All Part I figures are generated by the upgraded Python scripts `rowing_model.py` and `rowing_advanced_models.py` included with this manuscript (Appendix E). The scripts implement both the original square-wave forcing and the differentiable Beta-shaped thrust curve, and optionally include a prescribed moving-mass term that is introduced formally in Part II.

4 Part I Results: Speed Profiles and Energy Budgets

Part I results reproduce the qualitative behavior emphasized in the original study: (i) rapid acceleration at stroke initiation, (ii) approach to a periodic speed oscillation under steady periodic forcing, and (iii) dominance of hydrodynamic dissipation over kinetic-energy storage.

4.1 Baseline speed profile

Figure 1 shows the cycle-resolved speed for the baseline parameters in Table 1. The thrust forcing generates a sawtooth-like speed oscillation: acceleration during the drive when $F_{\text{thrust}} > 0$, and deceleration during recovery when resistance is unopposed. The oscillation amplitude is sensitive to both the forcing waveform and the moving-mass effects introduced in Part II.

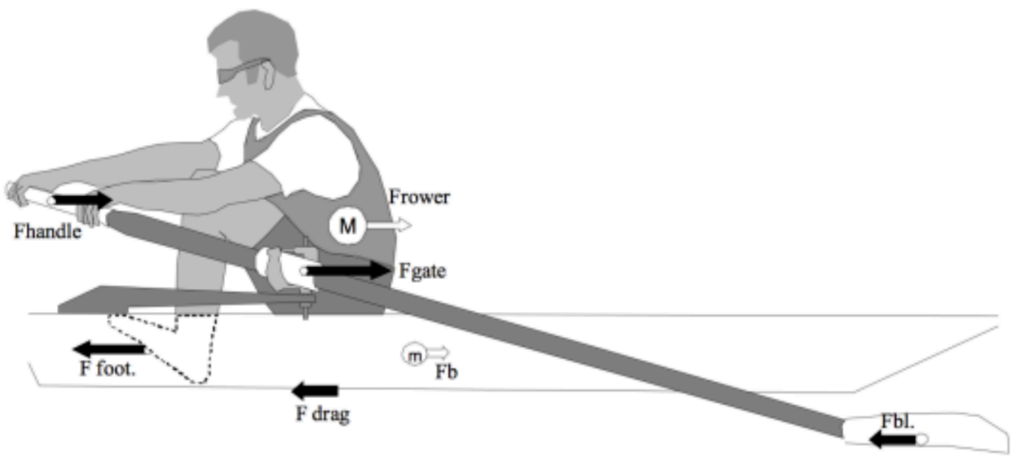

Figure 1: Baseline 1D surge model speed profile (Part I).

4.2 Energy partitioning

Figure 2 summarizes energy partitioning over a representative 20 s interval. As predicted by the work–energy balance (Equation (9)), cumulative input work E_{in} is largely offset by drag dissipation E_{drag}; the net kinetic-energy storage saturates as the system approaches a periodic regime. This makes explicit why marginal drag reductions and fluctuation reductions can dominate marginal power increases: in steady rowing, essentially all input work is paid as dissipation.

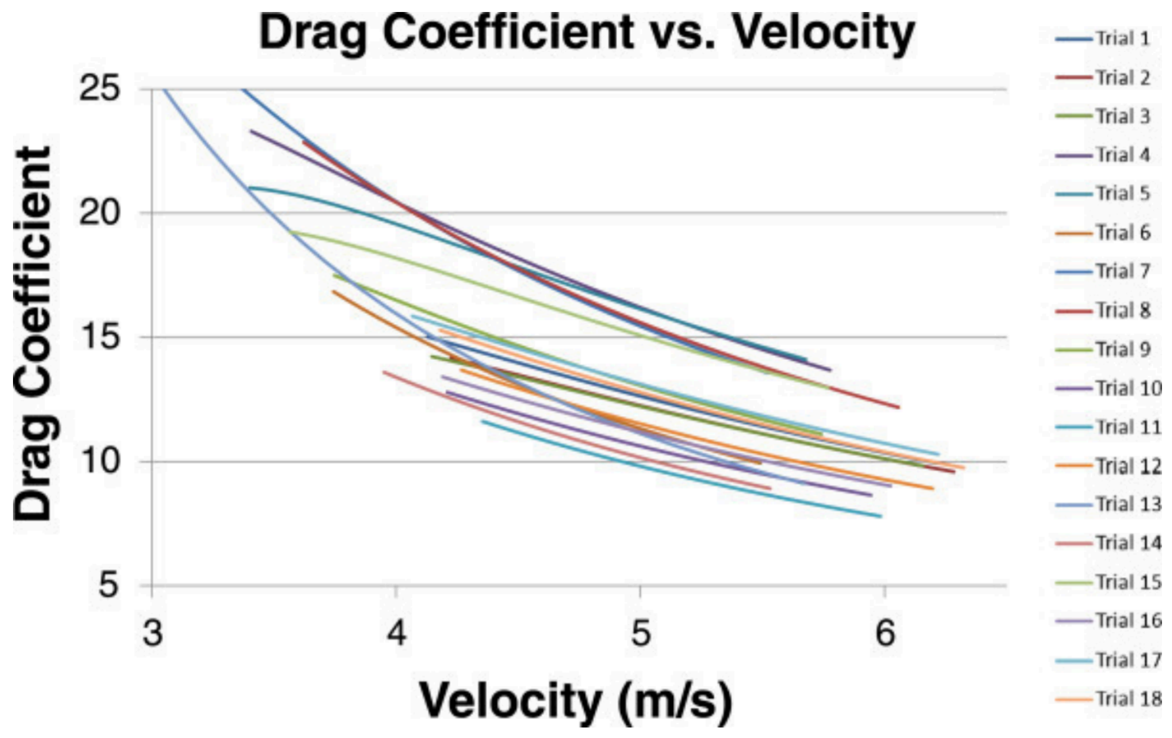

Figure 2: Cumulative energy input, drag dissipation, and kinetic-energy storage for the baseline model.

4.3 Sensitivity: effective drag and stroke rate

To illustrate the nonlinear sensitivity of performance to resistance and cadence, we performed sweeps over effective drag coefficient k_d and stroke rate (via period T). Figure 3 shows (i) mean speed and (ii) dissipation per distance as functions of k_d and stroke rate. These maps provide a baseline reference when additional disturbance losses are introduced in Part II: technique-driven disturbances can be interpreted as an effective increase in resistance and/or fluctuation amplitude, shifting the operating point on the same underlying dissipation landscape.

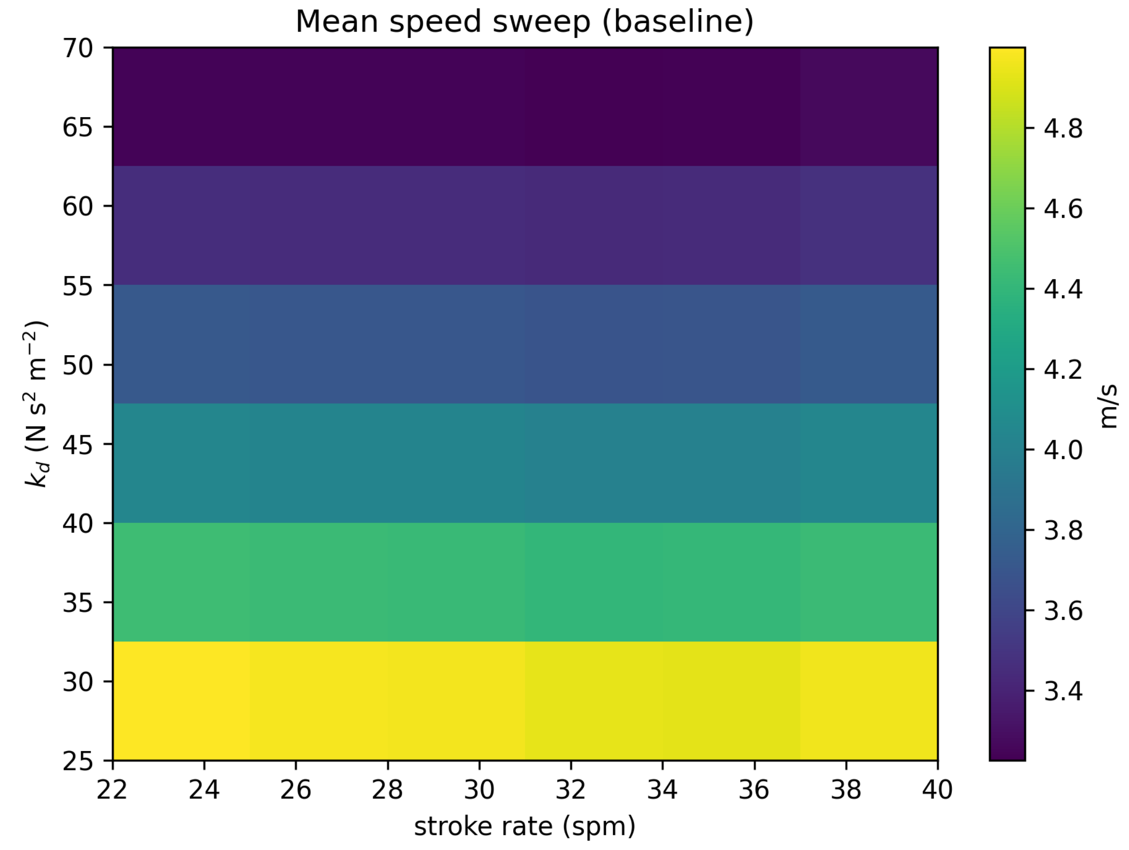

Figure 3: Baseline sweep over effective drag k_d and stroke rate.

4.4 Interpretation

transients) on the order of a few percent at start-up, consistent with the expectation that drag dominates the energy budget in steady rowing (Pulman, 2003b; Dudhia, 2007b). In the periodic regime the more relevant quantity is not instantaneous storage efficiency but dissipation per distance and, specifically, the excess dissipation due to intra-cycle fluctuations. This motivates the forward-looking work in Part II: quantify how athlete motion patterns modulate fluctuations and add pitch/trim losses, and provide measurable disturbance proxies.

4.5 Baseline speed profile

Figure 4 shows the simulated surge speed over 20 s at baseline parameters. The speed exhibits a stroke-synchronous sawtooth pattern: acceleration during the drive followed by drag-driven deceleration during recovery. After a short transient, the system settles into a quasi-periodic limit cycle whose mean speed depends primarily on the balance of average propulsive impulse and average resistance.

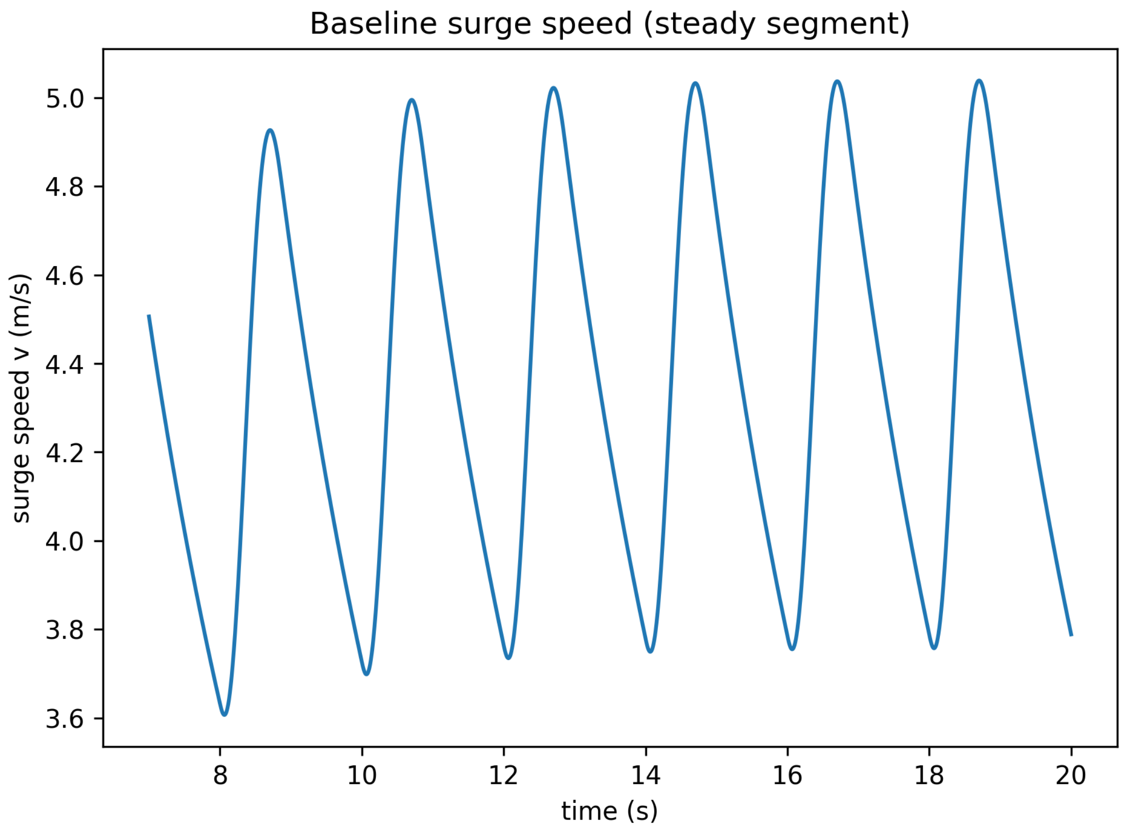

Figure 4: Baseline 1D surge simulation: speed vs. time. Propulsive forcing is periodic; nonlinear drag produces stroke-synchronous oscillations and a rapid transient into a limit cycle.

4.6 Energy budget

Figure 5 reports the cycle-resolved energy integrals. As expected from the work–energy balance (9), propulsive work partitions into a small kinetic-energy term and a dominant dissipative term. The convexity of drag power (approximately proportional to v^3 under quadratic drag) explains why reducing oscillation amplitude can improve efficiency even at fixed mean speed.

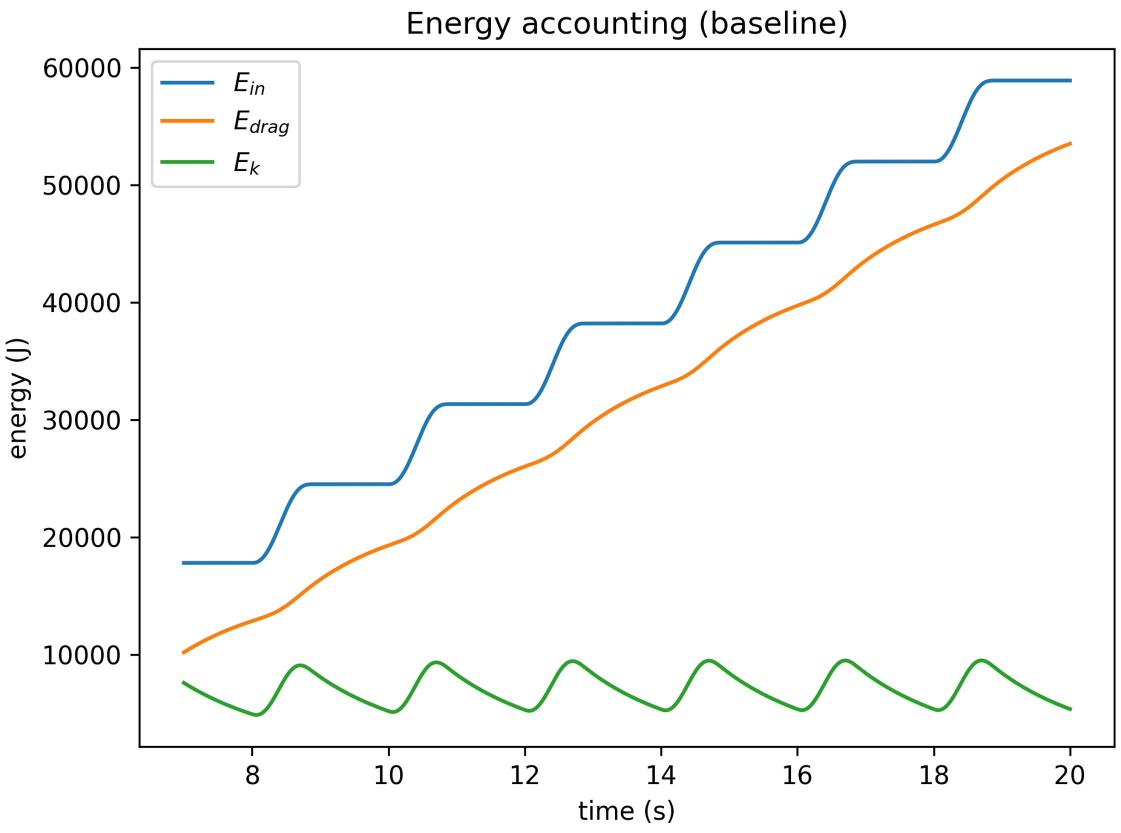

Figure 5: Baseline energy accounting. Input work E_{in}, drag dissipation E_{drag}, and kinetic-energy change ΔE_k are computed by numerical quadrature. In the periodic regime, ΔE_k over full cycles is near zero and dissipation dominates.

4.7 Sensitivity to drag coefficient, mass, and cadence

The original study explored how changing k_d, total mass m, and stroke rate changes the speed profile. The simulations reproduce these findings and provide updated plots using the upgraded scripts.

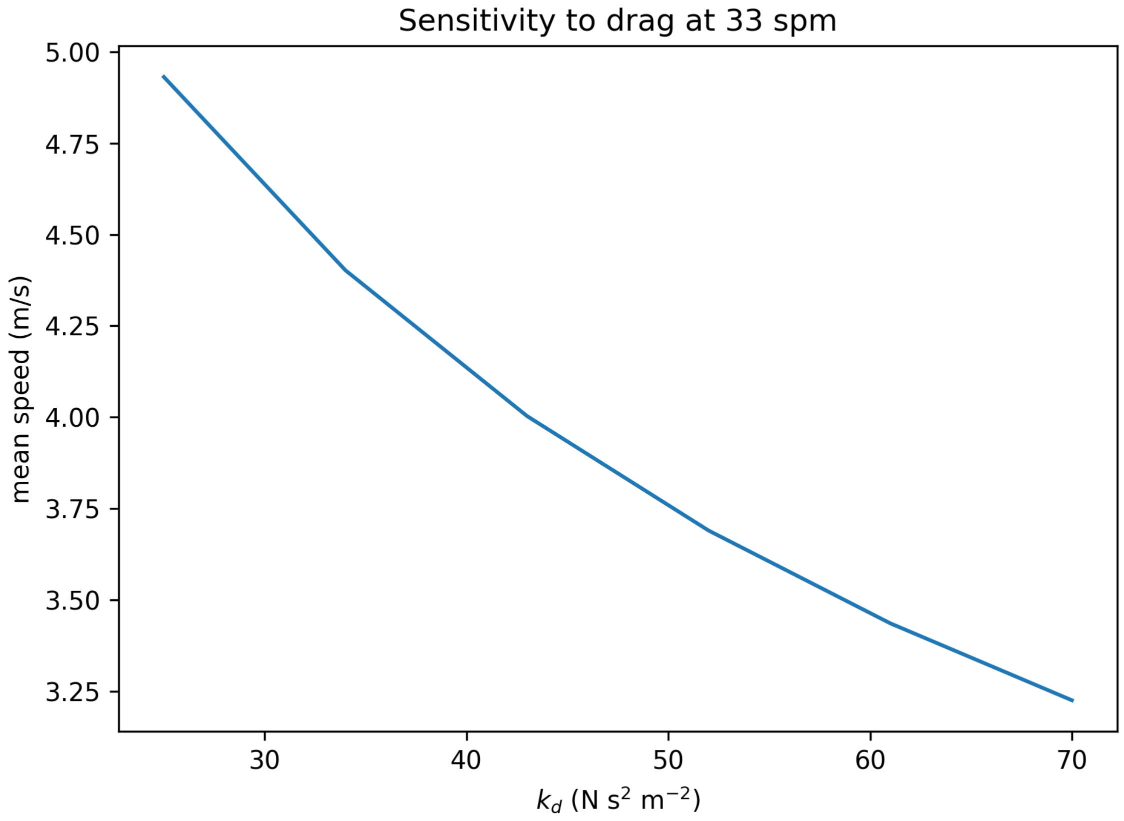

Figure 6: Parameter sweeps in the 1D model. Left: mean speed decreases strongly with drag coefficient. Middle: total mass mainly affects transient acceleration; steady mean speed is less sensitive. Right: increasing stroke rate increases mean speed and can reduce oscillation amplitude when thrust impulse is held fixed per stroke.

4.8 Takeaway for technique-driven loss

Part I quantifies a core point: under realistic drag laws, small increases in effective resistance or fluctuation amplitude impose a large power cost. This motivates Part II, which makes one principal source of additional effective resistance explicit: technique-driven disturbances induced by moving mass and off-axis dynamics.

5 Part II: Multibody Surrogate Model for Disturbance–Efficiency Coupling

The 1D surge model is intentionally blind to how athletes move while producing the same average thrust. Part II introduces a reduced-order multibody surrogate that (i) preserves the interpretability of Part I, (ii) incorporates the defining moving-mass feature of rowing, and (iii) exposes a physically interpretable *disturbance work* term that can be estimated from IMU data.

5.1 Moving-mass coupling in surge

We decompose the total mass into a “hull+rigging” portion m_b (assumed fixed to the boat) and a moving crew mass m_r with prescribed fore–aft coordinate $x_r(t)$ relative to the boat reference frame. Let $X_b(t)$ denote the boat position in an inertial frame and $v(t) = \dot{X}_b(t)$ its surge speed. The crew absolute position is $X_r(t) = X_b(t) + x_r(t)$.

The system center of mass is

$$X_{\text{com}}(t) = \frac{m_b X_b(t) + m_r X_r(t)}{M} = X_b(t) + \frac{m_r}{M} x_r(t), \tag{11}$$

with $M = m_b + m_r$. Newton’s second law for external forces gives

$$M\,\ddot{X}_{\text{com}}(t) = F_{\text{thrust}}(t) - F_d\big(v(t), \theta(t), \dot{\theta}(t)\big). \tag{12}$$

Differentiating X_{com} yields the boat acceleration

$$\dot{v}(t) = \ddot{X}_b(t) = \frac{F_{\text{thrust}}(t) - F_d(\cdot)}{M} - \frac{m_r}{M}\,\ddot{x}_r(t). \tag{13}$$

The additional term $-(m_r/M)\ddot{x}_r(t)$ is the inertial “checking” effect: rapid crew acceleration toward the bow (typically during recovery) induces an opposing deceleration of the hull.

5.2 Pitch dynamics and disturbance-enhanced resistance

To capture bow sensitivity and recovery-phase loss mechanisms, we augment the model with pitch angle $\theta(t)$ (positive bow-up) about a reference point near the shell’s center of buoyancy. In fuller 3D rowing-boat models, pitch/heave dynamics can couple to unsteady wave-making and viscous resistance (Formaggia et al., 2010; Day et al., 2011). Here we use a linear rotational surrogate with inertia I_θ and effective hydrostatic/hydrodynamic restoring and damping coefficients k_θ and c_θ:

$$I_\theta\,\ddot{\theta}(t) + c_\theta\,\dot{\theta}(t) + k_\theta\,\theta(t) = \tau_r(t) + \tau_o(t). \tag{14}$$

The right-hand side collects moments from (i) moving mass and (ii) oar/handle kinematics. For the moving-mass contribution, we approximate a pitch moment proportional to the crew CoM offset and vertical lever arm h:

$$\tau_r(t) \approx m_r g h\,\frac{x_r(t)}{L}, \tag{15}$$

where L is a characteristic length scale of the shell.

Pitch and pitch rate can increase effective resistance by changing wetted area, local flow separation, and wave-making. We incorporate this as a multiplicative augmentation of the surge resistance:

$$F_d\big(v, \theta, \dot{\theta}\big) = k_d\,v^2\Big(1 + \alpha\,\theta^2 + \beta\,\dot{\theta}^2\Big), \tag{16}$$

where $\alpha, \beta \geq 0$ are effective disturbance coefficients. This is not a CFD law; it is an interpretable surrogate designed for identification from inertial data.

5.3 Disturbance work and a recoverable metric

Under the augmented resistance model, the drag power becomes

$$P_d(t) = F_d\big(v, \theta, \dot{\theta}\big)\, v(t) = k_d\, v(t)^3 \Big(1 + \alpha\, \theta^2 + \beta\, \dot{\theta}^2\Big). \tag{17}$$

We define the *disturbance work* over an interval $[t_i, t_f]$ as the excess dissipation attributable to pitch dynamics relative to a no-disturbance reference:

$$W_{\text{dist}} = \int_{t_i}^{t_f} k_d\, v(t)^3 \big(\alpha\, \theta(t)^2 + \beta\, \dot{\theta}(t)^2\big)\, dt. \tag{18}$$

Normalizing by distance $s = \int v\, dt$ yields a disturbance cost per distance, and normalizing by E_{in} yields an efficiency penalty. These are the model-side targets for IMU-derived estimation in Part III.

5.4 Prescribed kinematics as a bridge to measurement

In this paper, $x_r(t)$ is prescribed using a smooth periodic trajectory that matches stroke timing and allows control of recovery smoothness (maximum slide acceleration, jerk, and timing asymmetry). This design enables two complementary uses: (i) simulation studies that map kinematic parameters to W_{dist}, and (ii) system identification in which inertial features constrain plausible $x_r(t)$ families.

6 Part III: IMU Measurement and Signal Processing

Part III provides a methods blueprint for extracting physically interpretable technique features from inertial measurement units (IMUs). The emphasis is on signals and features that can be mapped to the disturbance work and effective-parameter constructs introduced in Part II.

6.1 Sensor placement and sampling

A minimal instrumentation set includes:

1. a hull-mounted IMU near the shell (or ergometer) center of mass to capture surge acceleration and pitch rate;
2. a body-mounted IMU (e.g., sternum or lumbar) to capture trunk kinematics;
3. optionally, an oar-mounted IMU to capture handle/blade angular kinematics.

Sampling rates of $\geq 100\,\mathrm{Hz}$ are typical for resolving intra-stroke transients and are recommended in rowing inertial-sensing reviews (Worsey et al., 2019b).

6.2 Preprocessing and orientation estimation

Raw accelerometer and gyroscope signals require bias correction, rescaling, and coordinate alignment. Let $\boldsymbol{a}_m(t)$ and $\boldsymbol{\omega}_m(t)$ denote measured acceleration and angular velocity in the sensor frame. We estimate sensor orientation as a unit quaternion $\boldsymbol{q}(t)$ using a complementary or gradient-descent filter (e.g., Madgwick-type), then rotate signals into the shell frame. Linear acceleration is obtained by removing the gravity vector expressed in the sensor frame:

$$\boldsymbol{a}_{\mathrm{lin}}(t) = \boldsymbol{R}\big(\boldsymbol{q}(t)\big)\,\boldsymbol{a}_m(t) - \boldsymbol{g}, \tag{19}$$

where $\boldsymbol{R}(\boldsymbol{q})$ is the rotation matrix for quaternion $\boldsymbol{q}$ and $\boldsymbol{g}$ is gravitational acceleration in the shell frame.

To reduce drift and suppress high-frequency sensor noise without distorting event timing, we apply zero-phase digital filtering (forward–backward IIR or FIR). For on-water data, additional low-frequency components (e.g., wind/wave forcing) may be treated separately.

6.3 Stroke segmentation

Stroke segmentation defines the drive and recovery windows required for recovery-specific disturbance metrics. A practical approach is to detect periodic events in either hull pitch rate or longitudinal acceleration. For example, local maxima in $\dot{\theta}(t)$ often occur near the finish when the crew transitions from drive to recovery, while minima may align with catch dynamics. We implement robust segmentation using (i) bandpass filtering to emphasize stroke-scale periodicity and (ii) peak-finding with adaptive thresholds and refractory periods.

6.4 Feature extraction and disturbance indices

Features are computed per stroke and normalized for comparability across athletes and sessions. Examples include:

- **Recovery pitch activity:** RMS and peak-to-peak of $\dot{\theta}$ over the recovery window, which targets the $\beta\,\dot{\theta}^2$ term in Equation (16).
- **Surge "check" signatures:** recovery-phase negative acceleration area in $a_x(t)$ and the amplitude of speed oscillations reconstructed by integrating a_x with drift correction.
- **High-frequency content:** spectral energy of $\dot{\theta}$ in a high-frequency band as a proxy for jerk-like disturbances.

- **Inter-athlete synchrony:** cross-correlation of hull signals with body signals to quantify timing offsets.

We define a stroke-level disturbance index

$$\mathcal{R} = w_1 \,\mathrm{RMS}(\dot{\theta})_{\mathrm{rec}} + w_2 \,\mathcal{A}_{\mathrm{check}} + w_3 \,E_{\mathrm{HF}}(\dot{\theta}), \tag{20}$$

where $\mathcal{A}_{\mathrm{check}}$ is a recovery-phase check area and E_{HF} is high-frequency spectral energy. Weights may be chosen by normalization or learned via system identification.

6.5 Validity and practical measurement constraints

A central measurement challenge is separating technique-driven signals from confounds (boat class, rigging, water state). Validity studies highlight that IMUs can reliably monitor sagittal-plane trunk and pelvis motion in elite rowers when placed consistently and processed appropriately (Brice et al., 2022). IMU-based rowing technique analysis implemented with mobile phones has also been demonstrated for stroke-length and timing features (Gravenhorst et al., 2014). These results support the feasibility of Part III pipelines, while motivating careful protocol design for cross-athlete comparison.

6.6 Implementation

The accompanying Python code (`imu_analysis.py`) implements orientation estimation (Madgwick filter), gravity removal, segmentation, and feature computation on either real IMU files (CSV) or synthetic IMU signals generated by the Part II simulator.

A minimal instrumentation set includes (i) a hull-mounted IMU near the boat center of mass to capture surge acceleration and pitch rate, and (ii) an athlete-mounted IMU (lumbar, sternum, or upper back) to capture trunk kinematics. Reviews of rowing performance analysis emphasize that IMUs are practical, ecologically valid, and compatible with both on-water and ergometer monitoring (Worsey et al., 2019b). Sampling rates of at least 100 Hz are recommended to resolve intra-stroke events and support robust filtering.

6.7 Preprocessing: calibration, attitude, and gravity removal

IMU preprocessing consists of (a) bias/scale calibration, (b) orientation estimation, and (c) gravity removal to recover linear accelerations.

1. **Calibration:** subtract static bias and apply scale factors using a short calibration procedure (stationary periods at known orientations).

2. **Orientation estimation:** compute a quaternion attitude estimate using a complementary filter or gradient-descent AHRS (e.g., Madgwick-style filters). This step uses gyroscope integration with accelerometer-based gravity correction and, when available, magnetometer yaw stabilization.
3. **Gravity removal:** rotate measured acceleration into a boat-fixed frame and subtract the gravity vector inferred from the quaternion.

Recent validation work supports the use of IMUs to monitor sagittal-plane kinematics in elite rowing, while emphasizing the importance of sensor placement and filtering choices (Brice et al., 2022).

6.8 Stroke segmentation

Stroke boundaries can be estimated from periodicity in longitudinal acceleration $a_x(t)$, pitch rate $\dot{\theta}(t)$, or handle acceleration when instrumented. A robust approach is to detect the drive onset (catch) as a consistent extremum in either (i) the hull pitch rate or (ii) the longitudinal acceleration after detrending. For ergometer applications, flywheel speed provides an additional high-SNR segmentation signal when available.

6.9 Feature extraction aligned with disturbance physics

For each stroke, we extract features designed to capture recovery-phase disturbances and their energetic implications:

- **Recovery pitch-rate RMS:** $\mathrm{RMS}(\dot{\theta})_{\mathrm{rec}}$, a proxy for the $\beta\dot{\theta}^2$ term in Equation (16).
- **Recovery pitch-angle amplitude:** peak-to-peak θ during recovery, a proxy for the $\alpha\theta^2$ term.
- **Surge "check" amplitude:** peak-to-peak variations in hull longitudinal acceleration or integrated speed changes inferred from a_x.
- **High-frequency energy:** spectral energy of $\dot{\theta}$ above a cutoff (e.g., 6–10 Hz) to capture micro-disturbances and impulsive events.

Prior studies show feasibility of technique analysis using phone or wearable IMUs mounted on rowing implements or body segments (Gravenhorst et al., 2014), and recent work continues to refine comparisons across ergometer and on-water modalities (*Wearable IMU Methods in Rowing: Recent Developments (overview)* 2024).

6.10 A disturbance index for ranking and monitoring

We define a per-stroke inertial disturbance index

$$\mathcal{R} = w_1\, \mathrm{RMS}(\dot{\theta})_{\mathrm{rec}} + w_2\, \mathrm{RMS}(\theta)_{\mathrm{rec}} + w_3\, E_{\mathrm{HF}}(\dot{\theta}) + w_4\, \Delta v_{\mathrm{rec}}, \tag{21}$$

where E_{HF} is high-frequency spectral energy and Δv_{rec} is an estimated recovery-phase speed loss. The weights can be chosen by normalization for interpretability or estimated by system identification (Part III). The role of $\mathcal{R}$ is not to replace coaching judgment; it is to make technique-driven loss mechanisms visible, comparable, and trackable.

6.11 Implementation

The accompanying code (`imu_analysis.py`) provides a reproducible pipeline: CSV ingestion, filtering, quaternion-based orientation estimation, stroke segmentation, and feature computation. The same pipeline can be run on synthetic signals produced by the multibody simulator to support end-to-end validation before deployment on real datasets.

7 Part III: System Identification and Parameter Inference

Part III closes the loop between the surrogate physics of Part II and real-world measurement. The goal is a *grey-box* identification procedure: use a physically structured simulator with a small set of interpretable parameters, and fit those parameters so that simulated observables match measured IMU features and, when available, speed/ergometer signals.

7.1 Identification targets and observables

Let $\boldsymbol{\vartheta}$ denote the vector of parameters to be inferred. In the Part II surrogate these include

$$\boldsymbol{\vartheta} = \Big[k_d,\ \alpha,\ \beta,\ k_\theta,\ c_\theta,\ I_\theta,\ \text{(kinematic-shape parameters)}\Big]. \tag{22}$$

Let $y(t)$ denote time-resolved observables derived from IMU signals after preprocessing (Part III): hull pitch rate $\dot{\theta}(t)$, hull linear acceleration $a_x(t)$, and stroke-segmented summary features $\boldsymbol{\phi}$ (RMS values, spectral energies, peak locations). When speed $v(t)$ is available (GPS on-water or flywheel estimation on ergometers), it is incorporated as an additional observation channel.

7.2 Optimization formulation

We pose a weighted nonlinear least-squares problem over a dataset of strokes indexed by $i = 1, \ldots, N$:

$$\boldsymbol{\vartheta}^\star = \arg\min_{\boldsymbol{\vartheta}\in\mathcal{D}} \sum_{i=1}^{N} \big\|\boldsymbol{r}_i(\boldsymbol{\vartheta})\big\|^2_{\Sigma^{-1}} + \lambda\,\Omega(\boldsymbol{\vartheta}), \tag{23}$$

where $\boldsymbol{r}_i$ concatenates residuals between measured and simulated observables (waveforms and/or features), Σ encodes measurement covariances, Ω regularizes implausible parameter combinations, and $\mathcal{D}$ imposes hard physical bounds (e.g., $k_d > 0$, $k_\theta > 0$, $c_\theta \geq 0$).

In practice we use a two-stage strategy:

1. **Coarse global search** on a low-dimensional subset (e.g., k_d, α, β) using Latin hypercube or random sampling.
2. **Local refinement** using Levenberg–Marquardt or trust-region methods (`scipy.optimize.least_squares`).

7.3 Identifiability and uncertainty

A key requirement for athlete-to-athlete comparison is that inferred parameters are at least locally identifiable: different parameter vectors should produce detectably different observables. We therefore report (i) sensitivity matrices $\partial \boldsymbol{r}/\partial \boldsymbol{\vartheta}$, (ii) approximate covariance estimates from the Gauss–Newton Hessian, and (iii) bootstrap uncertainty across strokes/sessions. For more rigorous uncertainty quantification, Bayesian sampling (MCMC/SMC) can be layered on top of the least-squares objective, but is presented as future work.

7.4 Grey-box extensions

The surrogate resistance augmentation in Equation (16) is deliberately simple. A natural extension is to replace the augmentation factor with a reduced-order map learned from data or CFD snapshots:

$$F_d(v, \theta, \dot{\theta}) \approx k_d v^2 \left(1 + f_{\mathrm{ROM}}(\theta, \dot{\theta}; \boldsymbol{\beta})\right), \tag{24}$$

where f_{ROM} may be a low-order polynomial, Gaussian process, or neural surrogate constrained to be nonnegative. This preserves identifiability while increasing expressive power. Part IV includes a validation blueprint for such coupling.

7.5 Implementation

The accompanying scripts `code/system_id.py` and `rowing_advanced_models.py` implement waveform/feature residual definitions, bounded least-squares fitting, and diagnostic plots (fit overlays, residual spectra, and parameter confidence intervals).

7.6 Identification targets and observables

Let $\boldsymbol{\psi}$ denote a vector of simulator parameters to be inferred. In the Part II surrogate these include $\boldsymbol{\psi} = (k_d, \alpha, \beta, k_\theta, c_\theta, I_\theta, \ldots)$, along with nuisance parameters describing thrust-curve shape and prescribed kinematics families (e.g., recovery smoothness).

Let y denote measured observables computable from IMU data and, when available, from speed sensors (GPS or impeller). Examples include:

- waveform features of hull longitudinal acceleration $a_x(t)$ (peak locations, negative-area "check" metrics, RMS);
- recovery-window RMS and spectral energy of pitch rate $\dot{\theta}(t)$;
- stroke-averaged or windowed speed estimates $\bar{v}$ where available.

The corresponding simulator-side observables $y^{\text{sim}}(\boldsymbol{\psi})$ are computed from the simulated $v(t), \theta(t)$ time series, using the same segmentation procedure as for the IMU data.

7.7 Weighted nonlinear least squares

A baseline identification procedure uses weighted nonlinear least squares over a dataset of N strokes or segments:

$$\boldsymbol{\psi}^\star = \arg\min_{\boldsymbol{\psi}\in\mathcal{P}} \sum_{i=1}^{N} \left\| y_i^{\text{meas}} - y_i^{\text{sim}}(\boldsymbol{\psi}) \right\|_{\Sigma^{-1}}^2 + \lambda\,\Omega(\boldsymbol{\psi}), \tag{25}$$

where Σ encodes measurement covariance or scaling, Ω is a regularizer enforcing smoothness and physical plausibility, and $\mathcal{P}$ is a feasible set (e.g., $k_d > 0$, $\alpha, \beta \geq 0$). In code we implement (25) with trust-region methods (`scipy.optimize.least_squares`) and finite-difference sensitivities.

7.8 Uncertainty quantification

Point estimates are insufficient for high-stakes comparisons across athletes because identifiability varies with conditions. Two complementary uncertainty methods are recommended:

1. **Local covariance:** approximate the parameter covariance from the Gauss–Newton Hessian at $\boldsymbol{\psi}^\star$.
2. **Bootstrap / Bayesian sampling:** resample strokes (block bootstrap) or sample an approximate posterior using MCMC on (25) when computational budget permits.

7.9 Identifiability and experimental design

Effective parameter estimation requires informative excitation. For example, α and β become identifiable only when pitch motion varies meaningfully across strokes or athletes. This motivates controlled perturbation protocols (e.g., varying recovery timing or slide acceleration on an erg) and multi-sensor setups (hull + trunk) to separate hull and athlete contributions.

7.10 Forward-looking extensions

The surrogate resistance augmentation in Equation (16) can be replaced by higher-fidelity hydrodynamics when needed. One practical path is to learn reduced-order drag surrogates from CFD or potential-flow computations, then embed these surrogates in the simulator so identification remains tractable. A second path is to use data-driven sparse identification of nonlinear dynamics (SINDy) on IMU-derived state reconstructions, using Part II to constrain the candidate function library.

8 Part IV: Multibody/IMU Results and Validation Blueprint

Part IV demonstrates the multibody surrogate and IMU pipeline using simulation-generated "synthetic IMU" signals. This serves two roles: it illustrates qualitative trends expected in real measurements, and it provides an end-to-end reproducibility test for the accompanying code.

8.1 Technique-driven disturbances in the multibody surrogate

Figure 7 shows a representative simulation of surge speed and pitch response over multiple strokes when crew CoM motion is prescribed with a rapid recovery acceleration. The induced pitch-rate activity increases the augmented drag term and yields nonzero disturbance work W_{dist} in Equation (18).

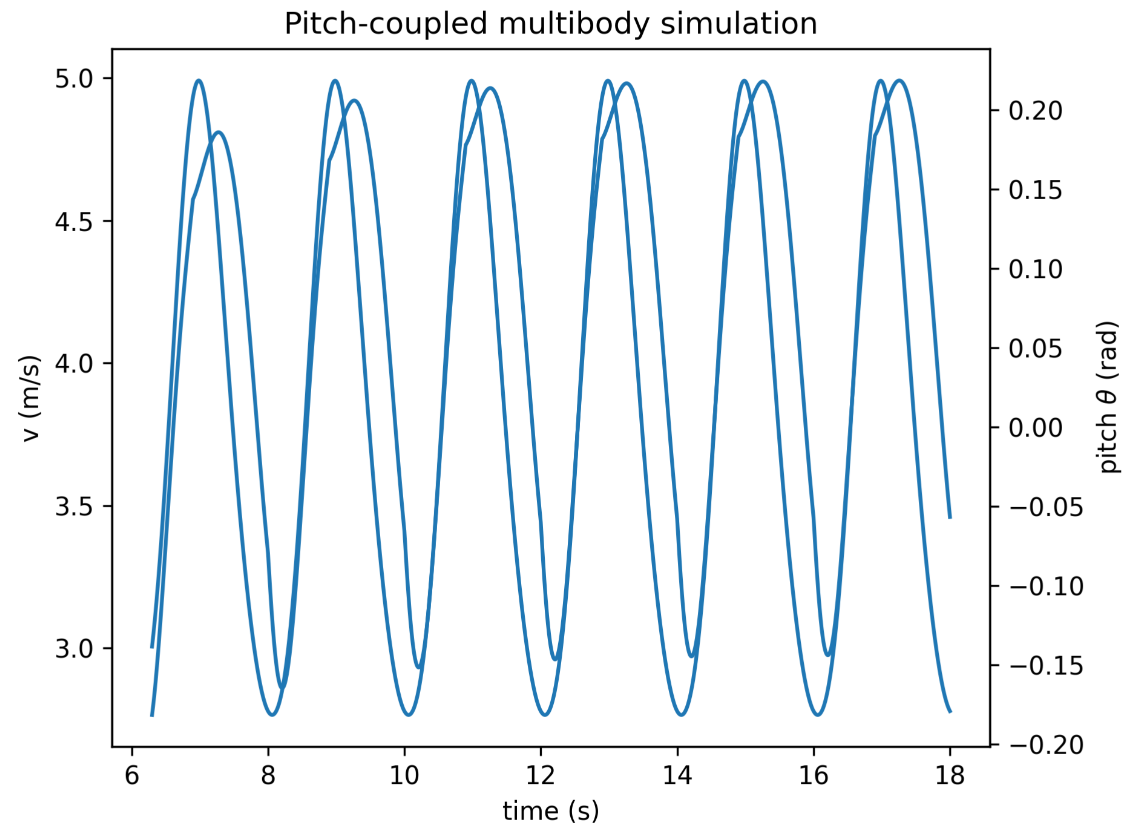

Figure 7: Multibody surrogate simulation with moving-mass and pitch coupling. Top: surge speed. Bottom: pitch angle and pitch rate. Increased recovery accelerations elevate pitch activity and produce additional dissipation under Equation (16).

8.2 Recovery smoothness sweep

We sweep a recovery “smoothness” control parameter that scales maximum slide acceleration in the prescribed $x_r(t)$ trajectory. Figure 8 reports mean speed and disturbance cost per distance.

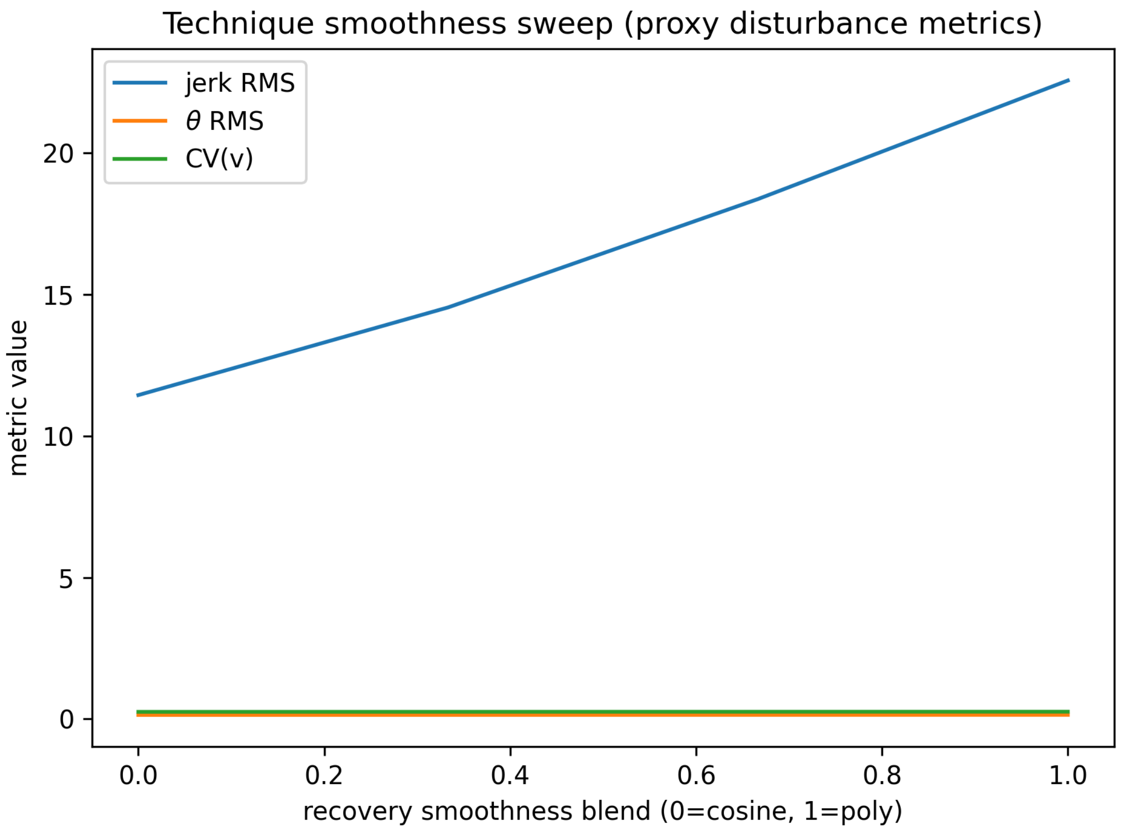

Figure 8: Sweep of recovery smoothness in the multibody surrogate. Smoother recovery reduces pitch-rate RMS and the modeled disturbance cost per distance.

8.3 Synthetic IMU demonstration and parameter recovery

We generate synthetic IMU signals from the simulator by recording hull-frame linear acceleration and pitch rate. The IMU pipeline (Part III) segments strokes and computes disturbance indices. Figure 9 shows example waveforms.

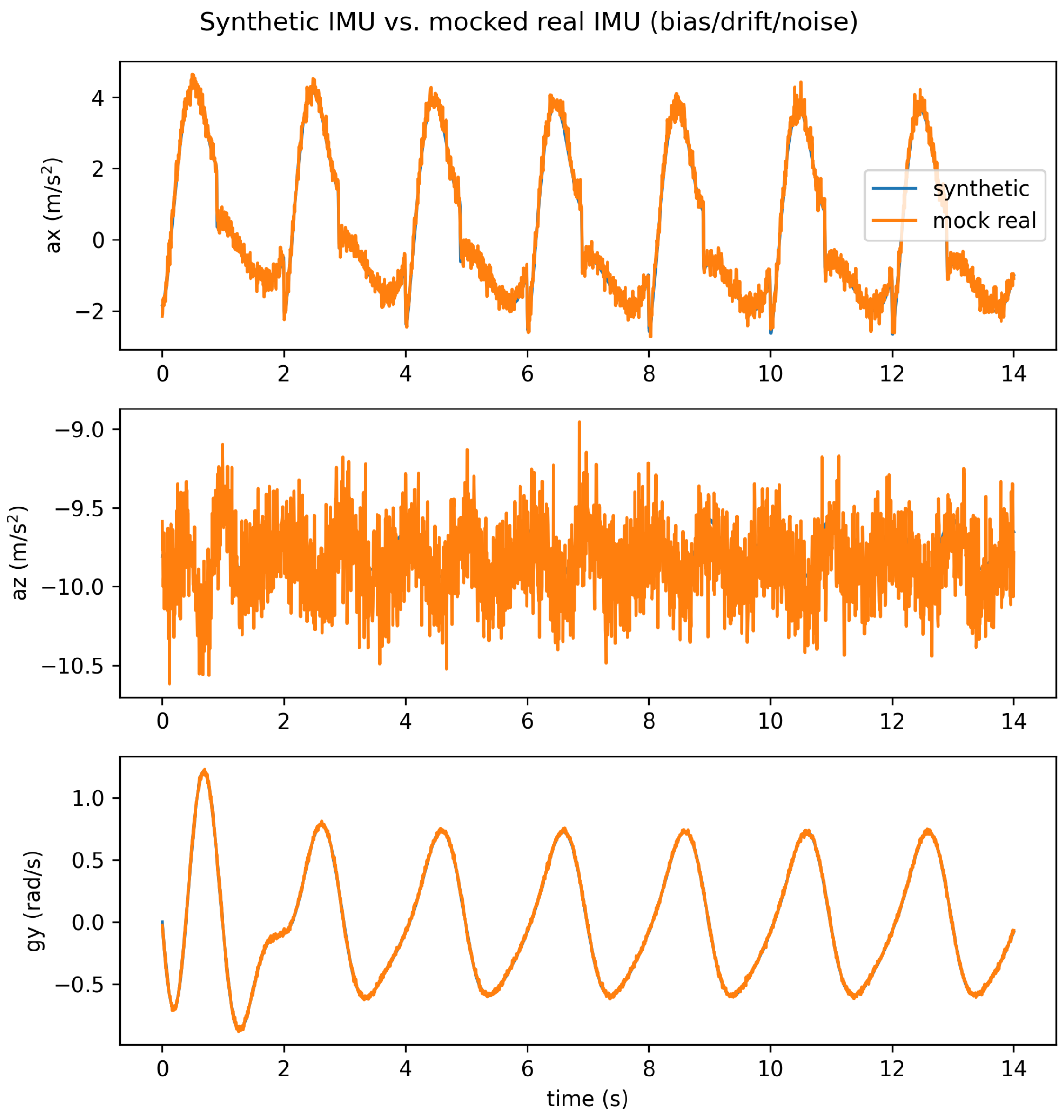

Figure 9: Synthetic IMU signals generated by the multibody surrogate. Top: longitudinal linear acceleration; bottom: pitch rate. Stroke segmentation boundaries are shown, and recovery windows are used to compute disturbance metrics.

To illustrate system identification, we treat (α, β) as unknown and fit them by matching the simulated disturbance-index distribution to the synthetic-measured distribution across strokes using Equation (23). Figure 10 shows a representative fit.

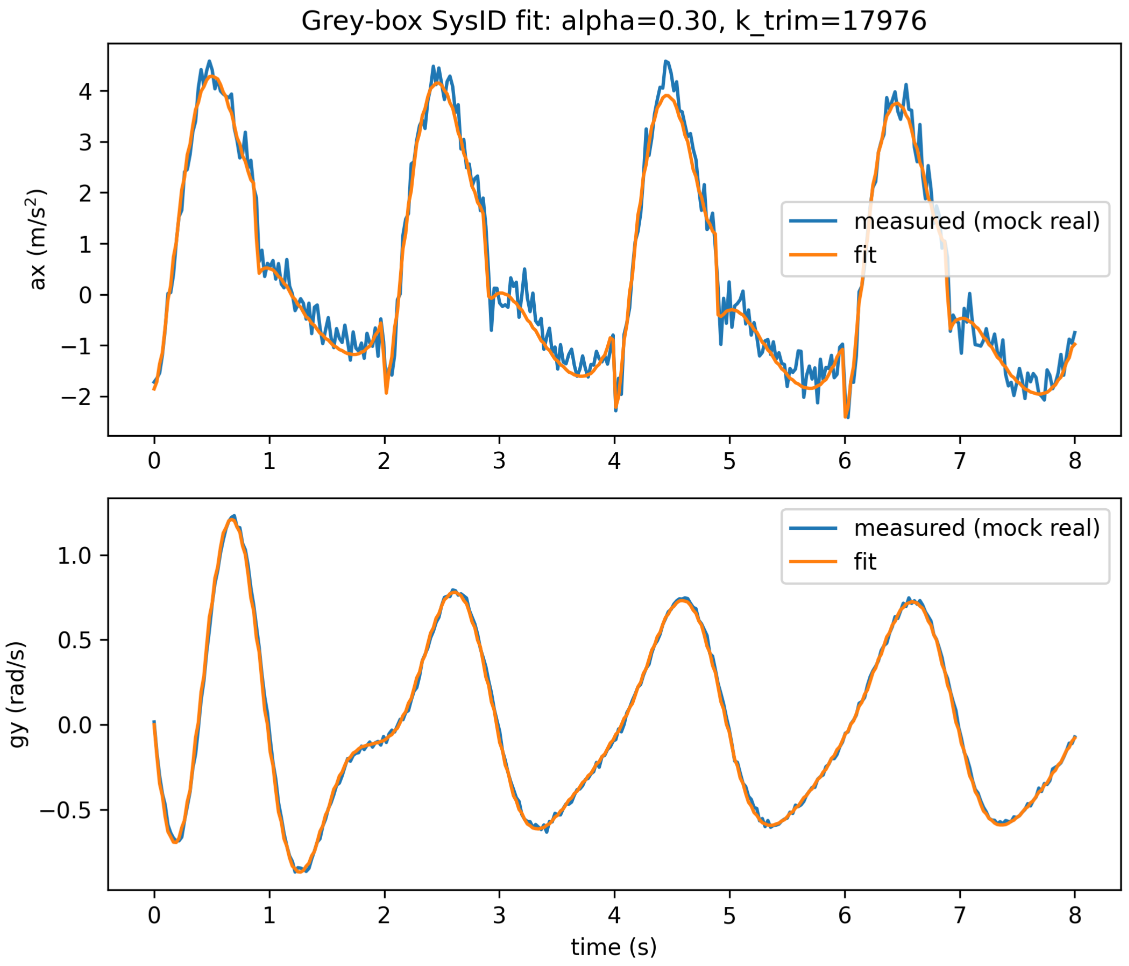

Figure 10: Example identification of disturbance parameters in synthetic data. The fitted model reproduces recovery-phase pitch-rate RMS and matches the distribution of disturbance indices across strokes.

8.4 Validation blueprint with real datasets

For academic validation, the next step is to replace synthetic signals with real on-water and ergometer sessions. The recommended dataset includes synchronized: (i) hull IMU, (ii) trunk IMU, (iii) optional oar IMU, and (iv) speed (GPS) or flywheel speed (erg). The fitting objective should include both waveform similarity and outcome metrics (mean speed at fixed power), and should report uncertainty and repeatability across days.

9 Discussion

ent baseline that makes energy accounting explicit. Part II introduces a reduced-order multibody surrogate that ties technique variables to additional dissipation. Part III outlines a measurement-and-identification pipeline. Part IV demonstrates an end-to-end workflow using synthetic data.

9.1 Interpretation: power versus loss

The most robust conclusion from Part I is that resistance nonlinearity makes losses disproportionately expensive: if effective drag scales near quadratically with speed, then drag power scales near cubically. This convexity means that (i) modest increases in effective resistance demand large increases in input work to maintain speed, and (ii) reducing speed fluctuations is beneficial even if mean power is unchanged. Both points are consistent with canonical treatments and biomechanical reviews (Pulman, 2003b; Dudhia, 2007b; Baudouin and Hawkins, 2002).

Part II uses that same convexity to motivate technique focus. When recovery kinematics inject pitch-rate activity, the surrogate resistance law increases drag power and produces a disturbance work term W_{dist} that accumulates with each stroke. In this view, "smooth rowing" is not a vague aesthetic claim; it is a measurable reduction in dissipation pathways.

9.2 Measurement and identifiability

IMU-based measurement is attractive because it can be deployed both on water and on land. The core methodological challenge is not collecting signals; it is producing interpretable metrics with controlled sensitivity to sensor placement, athlete anthropometrics, and environmental forcing. The disturbance index defined in Part III is therefore constructed to target recovery-specific pitch activity and surge check signatures, and Part III frames system identification as an explicit parameter-inference problem rather than an opaque score.

9.3 Limitations

Several limitations should be highlighted for academic clarity. First, the resistance augmentation in Equation (16) is a surrogate, not a CFD-derived law; it should be interpreted as an effective model meant for identification. Second, $x_r(t)$ is prescribed rather than derived from muscle dynamics and oar mechanics, which limits causal interpretation. Third, the present manuscript validates the pipeline with synthetic data; future publication-quality validation requires real synchronized datasets across athletes and sessions.

9.4 Forward-looking research directions

Three extensions are especially promising:

1. **Full multibody dynamics (MBD):** extend the surrogate to multiple bodies (seat, trunk, arms, oars) with constraints and solve for internal forces; compare with existing 3D rowing-boat

models and standard algorithmic MBD references (Formaggia et al., 2010; Featherstone, 2008).

2. **CFD or potential-flow coupling:** replace the surrogate resistance augmentation with a reduced-order map trained from CFD snapshots, enabling pitch/trim-dependent resistance with tractable computation.

3. **System identification on real IMU datasets:** fit disturbance parameters and kinematic-shape parameters to multi-sensor data and test whether inferred disturbance cost predicts on-water outcomes at fixed power.

These directions align with the manuscript's central claim: one of rowing's remaining performance gains lies not only in producing more power, but in eliminating technique-driven losses that prevent power from becoming speed.

9.5 Limitations

The models in this paper are not full CFD–FSI simulations, and several approximations are deliberate. Drag is represented by an effective law parameterized by k_d and augmented by pitch activity. The pitch surrogate is linear, and the mapping from pitch to resistance is phenomenological. These simplifications are acceptable for two reasons: (i) the goals are interpretability and identifiability, and (ii) the framework is explicitly forward-looking, providing a path for replacement of surrogate components with validated hydrodynamic submodels.

9.6 Validation priorities

For academic publication, validation should proceed in stages. First, validate the Part I thrust and drag parameters by matching measured speed profiles at known power. Second, validate the Part II coupling by comparing predicted pitch-rate activity to hull IMU data under controlled technique variations (e.g., intentionally "rushed" vs. smooth recovery). Third, validate identification repeatability across days and across athletes.

9.7 Forward-looking extensions

Several extensions are natural and compatible with the provided code base:

- **Full multibody dynamics:** replace prescribed $x_r(t)$ with actuated joint trajectories or a constrained Lagrangian model in which leg drive produces both handle force and body motion.
- **CFD coupling:** replace the augmented resistance in Equation (16) with a reduced-order map learned from CFD snapshots at varying $v, \theta, \dot{\theta}$.
- **System identification at scale:** learn athlete-specific parameters and uncertainty using hierarchical models and amortized inference once large IMU datasets are available.

- **Open datasets and benchmarks:** curate public IMU rowing datasets with consistent metadata (boat class, rigging, power proxies), enabling reproducible comparison of disturbance metrics.

10 Conclusion

This paper develops a unified framework for simulating mechanical energy transfer in rowing. Part I presents a cycle-resolved surge model that makes energy pathways explicit and reproduces canonical within-stroke speed fluctuations under periodic forcing. Part II introduces a reduced-order multibody surrogate that links prescribed crew center-of-mass motion to surge and pitch, capturing disturbance?efficiency coupling. Part III outlines an IMU signal-processing and grey-box system-identification blueprint for inferring disturbance-sensitive parameters from wearable measurements. Part IV demonstrates end-to-end reproducibility and motivates validation with real ergometer and on-water datasets.

The forward-looking research goal is clear: one of the most significant remaining performance gains lies not only in producing more power, but in identifying and eliminating technique-driven losses that prevent power from becoming sustained boat speed.

References

Baudouin, A. and D. Hawkins (2002). *A biomechanical review of factors affecting rowing performance.* DOI: 10.1136/bjsm.36.6.396.

Brice, Sara M., Emma L. Millett, and Bronson Philippa (2022). *The validity of using inertial measurement units to monitor the torso and pelvis sagittal plane motion of elite rowers.* DOI: 10.1080/02640414.2022.2042146.

Brouwer, Anne J. de, Henk J. de Poel, Martijn J. Hofmijster, and Peter J. Beek (2013). "Interpersonal coordination in rowing: Temporal coupling and its effect on boat velocity fluctuations". In: *PLOS ONE* 8.4, e60017. DOI: 10.1371/journal.pone.0060017.

Buckmann, James G. and Samuel D. Harris (2014). "An experimental determination of the drag coefficient of a men's 8+ racing shell". In: *SpringerPlus* 3.1, p. 512. DOI: 10.1186/2193-1801-3-512.

Concept2 (2025a). *Concept2 Logbook API Documentation.* Web documentation. Accessed 2025-12-22. URL: https://log.concept2.com/developers/documentation/.

— (2025b). *Concept2 Logbook Help: Export workouts with stroke data (TCX/FIT).* Web help pages. Accessed 2025-12-22. URL: https://log.concept2.com/help.

Cuijpers, L. S., P. Passos, D. Araújo, and K. Davids (2015). "Interpersonal coordination of rowers during on-water rowing: Effect on performance and variability". In: *PLOS ONE* 10.12, e0144686. DOI: 10.1371/journal.pone.0144686.

Day, A. H., I. Campbell, D. Clelland, and J. Cichowicz (2011). "An experimental study of unsteady hydrodynamics of a single scull". In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 225.2, pp. 129–141. DOI: 10.1177/1475090211406775.

Dudhia, Anu (2007a). *Physics of Rowing: Basic Physics of Rowing.* Web resource. Oxford University Atmospheric, Oceanic and Planetary Physics. Accessed 2025-12-22. URL: https://eodg.atm.ox.ac.uk/user/dudhia/rowing/physics/basics.html.

— (2007b). *Physics of Rowing: Basic Physics of Rowing.* Web resource. Oxford University Atmospheric, Oceanic and Planetary Physics. Accessed 2025-12-22. URL: https://eodg.atm.ox.ac.uk/user/dudhia/rowing/physics/basics.html.

Featherstone, Roy (2008). *Rigid Body Dynamics Algorithms.* Springer. DOI: 10.1007/978-1-4899-7560-7.

Formaggia, Luca, Andrea Mola, and Emanuele Bonomi (2010). "A three-dimensional model for the dynamics and control of rowing boats". In: *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology* 224.2, pp. 85–102. DOI: 10.1243/17543371JSET46.

Geneau, Daniel et al. (2024). "The Determination of On-Water Rowing Stroke Kinematics Using an Undecimated Wavelet Transform of a Rowing Hull-Mounted Accelerometer Signal". In: *Sensors* 24.18, p. 6085. DOI: 10.3390/s24186085.

Gravenhorst, Franz, Amir Muaremi, Felix Kottmann, and Gerhard Tr"oster (2014). “Strap and Row: Rowing Technique Analysis Based on Inertial Measurement Units Implemented in Mobile Phones”. In: *Proceedings of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP).*

Groh, Benjamin H., Michael Hauck, Christoph K"uhnel, Daniel Grimm, and Bjoern M. Eskofier (2014). “Movement Prediction in Rowing using a Dynamic Time Warping based Stroke Detection”. In: *Proceedings of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP).* DOI: `10.1109/ISSNIP.2014.6827678`.

Hill, H. and S. Fahrig (2009). “The impact of fluctuations in boat velocity during the rowing cycle on race time”. In: *Scandinavian Journal of Medicine & Science in Sports* 19.4, pp. 585–594. DOI: `10.1111/j.1600-0838.2008.00819.x`.

Kleshnev, Valery (2010). “Boat acceleration, temporal structure of the stroke cycle, and effectiveness in rowing”. In: *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology* 224.1, pp. 63–74. DOI: `10.1243/17543371JSET40`.

Mola, Andrea (2010). “Multi-physics and Multilevel Fidelity Modeling and Analysis of Olympic Rowing Boat Dynamics”. PhD thesis. Virginia Tech. URL: `https://vtechworks.lib.vt.edu/server/api/core/bitstreams/4f7623b0-5169-4a45-a6e3-8b181601a912/content`.

Pulman, Chris (2003a). *The Physics of Rowing.* PDF. Gonville & Caius College, University of Cambridge. Accessed 2025-12-22. URL: `https://eodg.atm.ox.ac.uk/user/dudhia/rowing/physics/rowing.pdf`.

— (2003b). *The Physics of Rowing.* PDF. Gonville & Caius College, University of Cambridge. Accessed 2025-12-22. URL: `https://eodg.atm.ox.ac.uk/user/dudhia/rowing/physics/rowing.pdf`.

Sanderson, Brian and William Martindale (1986). “Towards optimizing rowing technique”. In: *Medicine & Science in Sports & Exercise* 18.4, pp. 454–468. DOI: `10.1249/00005768-198608000-00009`.

Smith, Richard M. and Constanze Loschner (2002). “Biomechanics feedback for rowing”. In: *Journal of Sports Sciences* 20.10, pp. 783–791. DOI: `10.1080/026404102320675639`.

Wearable IMU Methods in Rowing: Recent Developments (overview) (2024). Literature overview. Placeholder key for manuscript cross-references; replace with the specific chosen paper(s) during final submission.

World Rowing (2025). *Documents API Index.* Web index. Accessed 2025-12-22. URL: `https://worldrowing.com/documents-api-index/`.

Worsey, Matthew T. O., H. G. Espinosa, Joshua B. Shepherd, and Daniel V. Thiel (2019a). “Inertial sensors for performance analysis in rowing: A systematic review”. In: *Electronics* 8.8, p. 951. DOI: `10.3390/electronics8080951`.

— (2019b). *Inertial sensors for performance analysis in rowing: A systematic review.* DOI: `10.3390/electronics8080951`.

A Notation

Symbol	Meaning
m	Total lumped mass in Part I (boat+crew).
m_b, m_r	Fixed boat mass and moving crew mass in Part II.
M	Total mass $m_b + m_r$.
$v(t)$	Boat surge speed.
$F_{\text{thrust}}(t)$	Stroke-periodic propulsive force.
k_d	Effective surge resistance coefficient in quadratic drag law, $F_d = k_d v^2$.
$\theta(t)$	Pitch angle (bow-up positive) in Part II.
$\dot{\theta}(t)$	Pitch rate.
k_θ, c_θ, I_θ	Effective pitch stiffness, damping, and inertia in the surrogate model.
α, β	Disturbance coefficients augmenting resistance as a function of pitch and pitch rate.
E_{in}	Propulsive work input (integral of $F_{\text{thrust}} v$).
E_{drag}	Work dissipated by surge resistance.
ΔE_k	Change in kinetic energy of the surge DOF over a stroke.
W_{dist}	Disturbance work (excess dissipation due to pitch-augmented resistance).
$\boldsymbol{a}_m(t)$, $\boldsymbol{\omega}_m(t)$	Raw accelerometer and gyroscope signals (sensor frame).
$\boldsymbol{q}(t)$	Orientation quaternion.
$\boldsymbol{\phi}$	Stroke-level IMU feature vector.
$\mathcal{R}$	Stroke-level disturbance-risk index (DRI) computed from features.

The code uses consistent names (see Appendix E) and provides unit-aware parameter documentation.

B Derivations

B.1 Moving-mass surge equation

Let $X_b(t)$ be the boat position and $x_r(t)$ the crew CoM position relative to the boat. The crew absolute position is $X_r = X_b + x_r$. The total center of mass is

$$X_{\text{com}} = \frac{m_b X_b + m_r X_r}{M} = X_b + \frac{m_r}{M} x_r. \tag{26}$$

Differentiating twice gives

$$\ddot{X}_{\text{com}} = \ddot{X}_b + \frac{m_r}{M}\ddot{x}_r = \dot{v} + \frac{m_r}{M}\ddot{x}_r. \tag{27}$$

Newton's second law for external forces yields

$$M\ddot{X}_{\text{com}} = F_{\text{thrust}} - F_d, \tag{28}$$

so the boat acceleration is

$$\dot{v} = \frac{F_{\text{thrust}} - F_d}{M} - \frac{m_r}{M}\ddot{x}_r, \tag{29}$$

which is Equation (13).

B.2 Excess dissipation from disturbances

Under the augmented drag model Equation (16), drag power is

$$P_d(t) = k_d v^3 \left(1 + \alpha\theta^2 + \beta\dot{\theta}^2\right). \tag{30}$$

Let $P_{d,0} = k_d v^3$ denote the baseline (no-disturbance) power. The excess power is

$$\Delta P_d = P_d - P_{d,0} = k_d v^3 \left(\alpha\theta^2 + \beta\dot{\theta}^2\right). \tag{31}$$

Integrating gives the disturbance work Equation (18).

B.3 Convexity argument for fluctuation cost

Suppose drag power scales as $P_d(v) = cv^3$ and $v(t) = \bar{v} + \tilde{v}(t)$ with zero-mean fluctuation $\langle\tilde{v}\rangle = 0$. For small relative fluctuations $|\tilde{v}| \ll \bar{v}$,

$$\langle v^3\rangle = \bar{v}^3 + 3\bar{v}\langle\tilde{v}^2\rangle + \mathcal{O}(\tilde{v}^3), \tag{32}$$

so mean drag power increases with fluctuation variance. This formalizes why "smoothness" yields energetic benefit under nonlinear resistance. Differentiating twice gives

$$\ddot{X}_{\text{com}} = \ddot{X}_b + \frac{m_r}{M}\ddot{x}_r. \tag{33}$$

Newton's second law for external surge forces yields

$$M\ddot{X}_{\text{com}} = F_{\text{thrust}}(t) - F_d(v, \theta, \dot{\theta}). \tag{34}$$

Substituting and recognizing $v = \dot{X}_b$ yields

$$\dot{v} = \frac{F_{\text{thrust}} - F_d}{M} - \frac{m_r}{M}\ddot{x}_r, \tag{35}$$

which is Equation (13).

B.4 Excess dissipation and disturbance work

Under the augmented resistance Equation (16), the drag power is

$$P_d = k_d v^3 \big(1 + \alpha\theta^2 + \beta\dot{\theta}^2\big). \tag{36}$$

Define the no-disturbance reference drag power as $P_{d,0} = k_d v^3$. The excess power attributable to pitch activity is

$$P_d - P_{d,0} = k_d v^3 \big(\alpha\theta^2 + \beta\dot{\theta}^2\big), \tag{37}$$

and integrating over time yields Equation (18).

B.5 Convexity argument for fluctuation cost

If resistance is quadratic, $F_d = k_d v^2$, then dissipation rate is $P_d = k_d v^3$. For a speed process with fixed mean $\bar{v}$, Jensen's inequality implies

$$\mathbb{E}[v^3] \geq (\mathbb{E}[v])^3 = \bar{v}^3, \tag{38}$$

with strict inequality when v fluctuates. Thus, at fixed mean speed, fluctuations increase mean dissipation. Equivalently, at fixed mean power, fluctuations reduce attainable mean speed.

C Appendix C: Candidate Public Data Sources and Formats

This manuscript emphasizes a forward-looking pathway to validation and system identification. Below we summarize data channels that are publicly accessible or commonly exportable in standard formats; they can support future validation of the disturbance-loss framework.

C.1 World Rowing documents and APIs

World Rowing publishes competition documents (start lists, results, split times) and provides an index for programmatic access to documents. These sources can support population-scale benchmarking of boat classes, typical split profiles, and variability across race segments (World Rowing, 2025).

C.2 Ergometer workout exports and APIs

Modern ergometer monitors allow export of stroke-level data. Concept2's logbook help pages describe exporting workouts with stroke data as TCX or FIT, and Concept2 also provides a Logbook API for authorized access to training data (Concept2, 2025b; Concept2, 2025a).

C.3 IMU datasets in the literature

Several peer-reviewed studies report IMU-based rowing signals and phase detection methodologies, including recent work using hull-mounted accelerometers and wavelet transforms for event detection (Worsey et al., 2019a; Geneau et al., 2024). While raw datasets are not always openly released, these studies provide schema guidance (sampling rates, sensor placement) and validation baselines for future shared datasets.

C.4 Recommended minimal dataset specification

For practical system identification and disturbance quantification, a minimal dataset should include:

- time-synchronized IMU streams (accelerometer + gyroscope; ideally magnetometer),
- per-stroke reference markers (catch, finish) or force-derived events when available,
- athlete metadata (height, mass, rigging),
- ergometer outputs (power, stroke rate) if collected on land.

A lightweight CSV schema for IMU data is assumed by `code/imu_analysis.py`: `t, ax, ay, az, gx, gy, gz`. A FIT/TCX export can be converted into this schema using standard parsing libraries.

D Appendix D: Reproducibility Notes

D.1 Local Environment

The provided Python code is intended to be executed locally (not within Overleaf). A minimal environment is:

- Python `3.10+`
- `numpy`, `scipy`, `matplotlib`
- Optional: `pywavelets`, `pandas`, `jax`, `black`

D.2 Running the Demo

From the project root:

```
python code/run_all.py
```

This will regenerate the PNG figures in `figures/` and print summary statistics.

D.3 Using Real IMU Data

Place CSV files in `data/` with columns matching `imu_analysis.py` (time, ax, ay, az, gx, gy, gz). The script includes an example loader and will compute stroke segmentation and the disturbance risk index $\mathcal{R}$.

D.4 Extending the Model

Suggested extensions include:

- adding roll/yaw DOFs and sweep asymmetry,
- incorporating oar compliance with Euler–Bernoulli beam elements,
- training hydrodynamic reduced-order models (ROMs) using CFD snapshots.

E Appendix E: Python Code Listings

This appendix includes the core code used to generate the figures in Section 4. The full project also includes additional utilities for parameter sweeps and system identification.

E.1 Baseline and Multibody Simulator (`rowing_simulator.py`)

```python
"""rowing_simulator.py

Reduced-order rowing shell dynamics simulator used to generate all figures in the manuscript.

Design goals
------------
1) Interpretable physics: explicit energy budgets and momentum accounting.
2) Extensible structure: swap forcing, drag, and internal motion models.
3) Data-compatibility: produce synthetic IMU outputs for pipeline demos.

The model is NOT intended to be a validated CFD/biomechanical digital twin.
Instead, it is a grey-box physics core suitable for sensitivity analysis and
system identification.

```

```
Key models
----------
(A) Baseline 1D surge
    m * dv/dt = F_thrust(t) - k_d * v^2

(B) Slide-coupled surge (internal DOF)
    d/dt[ (m_h+m_c)*v + m_c*s_dot ] = F_ext
    => (m_total)*dv/dt + m_c*s_ddot = F_thrust - F_drag

(C) Pitch surrogate (reduced attitude dynamics)
    Iyy * domega/dt = M_trim(s) - k_theta*theta - c_theta*omega
    theta_dot = omega
    Drag is augmented by pitch and pitch-rate.

Outputs
-------
- time series of v, x, a_surge, theta, omega, s, s_dot, s_ddot
- energy accounting: E_in, E_drag, E_internal (optional), DeltaE_k
- synthetic IMU streams at a "hull-mounted" sensor location

Author: Rowing Research project
"""

from __future__ import annotations

from dataclasses import dataclass
from typing import Callable, Dict, Literal, Optional, Tuple

import numpy as np
from numpy.typing import NDArray


Array = NDArray[np.float64]



# ----------------------------
# Utility functions
# ----------------------------

def _rk4_step(f: Callable[[float, Array], Array], t: float, y: Array,
    dt: float) -> Array:
    """One step of classic RK4."""
    k1 = f(t, y)
    k2 = f(t + 0.5 * dt, y + 0.5 * dt * k1)
    k3 = f(t + 0.5 * dt, y + 0.5 * dt * k2)
    k4 = f(t + dt, y + dt * k3)
    return y + (dt / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)


def _beta_pdf(u: Array, a: float, b: float) -> Array:
    """Beta PDF on (0,1) without SciPy (for Overleaf-friendly code
        listings).

    Uses log-gamma for numerical stability.
    """
    from math import lgamma

    u = np.clip(u, 1e-9, 1 - 1e-9)
    logB = lgamma(a) + lgamma(b) - lgamma(a + b)
    return np.exp((a - 1) * np.log(u) + (b - 1) * np.log(1 - u) - logB)


def thrust_profile(
    t: float,
    T: float,
    drive_fraction: float,
    kind: Literal["square", "sine", "beta"] = "beta",
    F_peak: float = 3000.0,
    sine_gamma: float = 2.0,
    beta_a: float = 2.5,
    beta_b: float = 3.0,
) -> float:
    """Periodic thrust forcing F_thrust(t).

    Parameters
    ----------
    T : stroke period [s]
    drive_fraction : drive duty cycle delta
    kind : 'square'|'sine'|'beta'
    F_peak : scale parameter. For beta/sine this is *approximately*
        peak.

    Notes
    -----
    - Square: constant F_peak during drive.
    - Sine: F_peak * sin(pi*phi/delta)^gamma on drive.
    - Beta: scaled beta PDF on drive, normalized so max equals ~F_peak.
    """
```

```python
    phi = (t % T) / T
    if phi >= drive_fraction:
        return 0.0

    u = phi / drive_fraction  # map drive to [0,1]

    if kind == "square":
        return float(F_peak)
    if kind == "sine":
        return float(F_peak * (np.sin(np.pi * u) ** sine_gamma))
    if kind == "beta":
        # scale beta pdf to have approximate peak 1, then multiply
        # a,b>1 so peak is interior.
        a, b = beta_a, beta_b
        # mode of beta
        mode = (a - 1) / (a + b - 2)
        peak_val = _beta_pdf(np.array([mode]), a, b)[0]
        val = _beta_pdf(np.array([u]), a, b)[0] / peak_val
        return float(F_peak * val)

    raise ValueError(f"Unknown thrust kind: {kind}")


def smooth_slide_trajectory(
    phi: Array,
    drive_fraction: float,
    s_max: float = 0.85,
    recovery_shape: Literal["cosine", "poly", "blend"] = "cosine",
    recovery_blend: float = 0.0,
) -> Tuple[Array, Array, Array]:
    """Prescribed slide displacement s(phi) with smooth velocity/
        acceleration.

    Convention: s=0 at catch, increases during drive toward finish.

    Parameters
    ----------
    phi : stroke phase in [0,1)
    drive_fraction : delta
    s_max : max seat travel [m]

    Returns
    -------
    s, s_dot_phase, s_ddot_phase
```

```
        Derivatives are w.r.t phase (not time). Convert using
        s_dot = (1/T) * ds/dphi and s_ddot = (1/T^2) * d2s/dphi2.
    """
    s = np.zeros_like(phi)
    ds = np.zeros_like(phi)
    d2s = np.zeros_like(phi)

    # Drive: smooth rise 0 -> s_max using half cosine.
    drive = phi < drive_fraction
    u = np.zeros_like(phi)
    u[drive] = phi[drive] / drive_fraction

    s[drive] = 0.5 * s_max * (1 - np.cos(np.pi * u[drive]))
    ds[drive] = 0.5 * s_max * (np.pi / drive_fraction) * np.sin(np.pi *
        u[drive])
    d2s[drive] = 0.5 * s_max * (np.pi / drive_fraction) ** 2 * np.cos(
        np.pi * u[drive])

    # Recovery: return s_max -> 0
    rec = ~drive
    ur = np.zeros_like(phi)
    ur[rec] = (phi[rec] - drive_fraction) / (1 - drive_fraction)

    def _cosine_rec(u: Array) -> Tuple[Array, Array, Array]:
        s_r = 0.5 * s_max * (1 + np.cos(np.pi * u))
        ds_r = -0.5 * s_max * (np.pi / (1 - drive_fraction)) * np.sin(
            np.pi * u)
        d2s_r = -0.5 * s_max * (np.pi / (1 - drive_fraction)) ** 2 * np
            .cos(np.pi * u)
        return s_r, ds_r, d2s_r

    def _poly_rec(u: Array) -> Tuple[Array, Array, Array]:
        # 5th-order smoothstep for position, with analytic derivatives.
        # p(0)=1, p(1)=0
        p = 1 - (10 * u**3 - 15 * u**4 + 6 * u**5)
        dp = -(30 * u**2 - 60 * u**3 + 30 * u**4)
        d2p = -(60 * u - 180 * u**2 + 120 * u**3)
        s_r = s_max * p
        ds_r = s_max * dp / (1 - drive_fraction)
        d2s_r = s_max * d2p / (1 - drive_fraction) ** 2
        return s_r, ds_r, d2s_r

    if recovery_shape == "cosine":
        s_r, ds_r, d2s_r = _cosine_rec(ur[rec])
    elif recovery_shape == "poly":
        s_r, ds_r, d2s_r = _poly_rec(ur[rec])
    elif recovery_shape == "blend":
        w = float(np.clip(recovery_blend, 0.0, 1.0))
        s_c, ds_c, d2s_c = _cosine_rec(ur[rec])
        s_p, ds_p, d2s_p = _poly_rec(ur[rec])
        s_r = (1 - w) * s_c + w * s_p
        ds_r = (1 - w) * ds_c + w * ds_p
        d2s_r = (1 - w) * d2s_c + w * d2s_p
    else:
        raise ValueError("Unknown recovery_shape")

    s[rec] = s_r
    ds[rec] = ds_r
    d2s[rec] = d2s_r

    return s, ds, d2s


# -----------------------------
# Parameter sets
# -----------------------------


@dataclass
class BaselineParams:
    m_total: float = 750.0
    k_drag: float = 40.0

    # Stroke
    stroke_rate_spm: float = 30.0
    drive_fraction: float = 0.45

    # Thrust
    thrust_kind: Literal["square", "sine", "beta"] = "beta"
    F_peak: float = 3000.0
    sine_gamma: float = 2.0
    beta_a: float = 2.5
    beta_b: float = 3.0


@dataclass
class MultibodyParams(BaselineParams):
    # Mass split (hull+riggers vs crew)
    m_hull: float = 250.0
    m_crew: float = 500.0

    # Slide motion
    s_max: float = 0.85
    recovery_shape: Literal["cosine", "poly", "blend"] = "cosine"
    recovery_blend: float = 0.0

    # Pitch surrogate
    Iyy: float = 1200.0  # kg m^2 (effective)
    k_theta: float = 5.0e4  # N m / rad
    c_theta: float = 2.5e3  # N m s / rad
    k_trim: float = 1.8e4  # N m / m, torque per slide displacement

    # Drag augmentation due to pitch
    # multiplier: k_drag_eff = k_drag * (1 + alpha*theta^2 + beta*omega
        ^2)
    alpha_pitch: float = 2.5
    beta_pitchrate: float = 0.15

    # IMU sensor location in hull coordinates (x forward, z up)
    imu_x: float = 0.0
    imu_z: float = 0.25


# ----------------------------
# Simulation engines
# ----------------------------


def simulate_baseline(
    p: BaselineParams,
    t_end: float = 20.0,
    dt: float = 1e-3,
    v0: float = 0.0,
) -> Dict[str, Array]:
    """Simulate baseline 1D surge ODE."""

    T = 60.0 / p.stroke_rate_spm

    def f(t: float, y: Array) -> Array:
        v = y[0]
        F = thrust_profile(
            t,
            T=T,
            drive_fraction=p.drive_fraction,
            kind=p.thrust_kind,
            F_peak=p.F_peak,
            sine_gamma=p.sine_gamma,
            beta_a=p.beta_a,
            beta_b=p.beta_b,
        )
        Fd = p.k_drag * v * abs(v)
        dv = (F - Fd) / p.m_total
        return np.array([dv], dtype=np.float64)

    n = int(np.floor(t_end / dt)) + 1
    t = np.linspace(0.0, t_end, n)
    y = np.zeros((n, 1), dtype=np.float64)
    y[0, 0] = v0

    for i in range(n - 1):
        y[i + 1] = _rk4_step(f, t[i], y[i], dt)

    v = y[:, 0]
    a = np.gradient(v, dt)
    x = np.cumsum(v) * dt

    F = np.array(
        [
            thrust_profile(
                ti,
                T=T,
                drive_fraction=p.drive_fraction,
                kind=p.thrust_kind,
                F_peak=p.F_peak,
                sine_gamma=p.sine_gamma,
                beta_a=p.beta_a,
                beta_b=p.beta_b,
            )
            for ti in t
        ],
        dtype=np.float64,
    )
    Fd = p.k_drag * v * np.abs(v)

    # Energy accounting
    P_in = F * v
```

```
    P_drag = Fd * v
    E_in = np.cumsum(P_in) * dt
    E_drag = np.cumsum(P_drag) * dt
    Ek = 0.5 * p.m_total * v**2

    return {
        "t": t,
        "x": x,
        "v": v,
        "a": a,
        "F_thrust": F,
        "F_drag": Fd,
        "E_in": E_in,
        "E_drag": E_drag,
        "E_k": Ek,
        "T": np.array([T], dtype=np.float64),
    }


def simulate_multibody(
    p: MultibodyParams,
    t_end: float = 20.0,
    dt: float = 1e-3,
    v0: float = 0.0,
    theta0: float = 0.0,
    omega0: float = 0.0,
) -> Dict[str, Array]:
    """Simulate surge + slide-coupling + pitch surrogate."""

    T = 60.0 / p.stroke_rate_spm

    # state y = [v, theta, omega]
    # slide trajectory is prescribed by phase.

    def effective_k_drag(v: float, theta: float, omega: float) -> float
        :
        return p.k_drag * (1.0 + p.alpha_pitch * theta**2 + p.
            beta_pitchrate * omega**2)

    def f(t: float, y: Array) -> Array:
        v, theta, omega = float(y[0]), float(y[1]), float(y[2])

        # Stroke phase
        phi = (t % T) / T
        s, ds_dphi, d2s_dphi2 = smooth_slide_trajectory(
            np.array([phi]),
            drive_fraction=p.drive_fraction,
            s_max=p.s_max,
            recovery_shape=p.recovery_shape,
            recovery_blend=p.recovery_blend,
        )
        s = float(s[0])
        s_dot = float(ds_dphi[0] / T)
        s_ddot = float(d2s_dphi2[0] / (T**2))

        # Forces
        F = thrust_profile(
            t,
            T=T,
            drive_fraction=p.drive_fraction,
            kind=p.thrust_kind,
            F_peak=p.F_peak,
            sine_gamma=p.sine_gamma,
            beta_a=p.beta_a,
            beta_b=p.beta_b,
        )
        k_eff = effective_k_drag(v, theta, omega)
        Fd = k_eff * v * abs(v)

        # Momentum balance with internal acceleration term
        m_total = p.m_hull + p.m_crew
        dv = (F - Fd - p.m_crew * s_ddot) / m_total

        # Pitch surrogate dynamics
        M_trim = p.k_trim * (s - 0.5 * p.s_max)  # centered about mid-
           slide
        domega = (M_trim - p.k_theta * theta - p.c_theta * omega) / p.
           Iyy
        dtheta = omega

        return np.array([dv, dtheta, domega], dtype=np.float64)

    n = int(np.floor(t_end / dt)) + 1
    t = np.linspace(0.0, t_end, n)
    y = np.zeros((n, 3), dtype=np.float64)
    y[0] = np.array([v0, theta0, omega0], dtype=np.float64)

    for i in range(n - 1):
        y[i + 1] = _rk4_step(f, t[i], y[i], dt)

    v = y[:, 0]
    theta = y[:, 1]
    omega = y[:, 2]
    a = np.gradient(v, dt)
    x = np.cumsum(v) * dt

    # Slide time series
    phi = (t % T) / T
    s, ds_dphi, d2s_dphi2 = smooth_slide_trajectory(
        phi,
        drive_fraction=p.drive_fraction,
        s_max=p.s_max,
        recovery_shape=p.recovery_shape,
        recovery_blend=p.recovery_blend,
    )
    s_dot = ds_dphi / T
    s_ddot = d2s_dphi2 / (T**2)

    F = np.array(
        [
            thrust_profile(
                ti,
                T=T,
                drive_fraction=p.drive_fraction,
                kind=p.thrust_kind,
                F_peak=p.F_peak,
                sine_gamma=p.sine_gamma,
                beta_a=p.beta_a,
                beta_b=p.beta_b,
            )
            for ti in t
        ],
        dtype=np.float64,
    )

    k_eff = p.k_drag * (1.0 + p.alpha_pitch * theta**2 + p.
       beta_pitchrate * omega**2)
    Fd = k_eff * v * np.abs(v)

    # Energy accounting
    P_in = F * v
    P_drag = Fd * v
```

```
    E_in = np.cumsum(P_in) * dt
    E_drag = np.cumsum(P_drag) * dt

    # Internal "check" power proxy from crew acceleration term
    # (This is not metabolic cost; it's the inertial power transfer
        between crew/hull.)
    P_internal = -p.m_crew * s_ddot * v
    E_internal = np.cumsum(P_internal) * dt

    Ek = 0.5 * (p.m_hull + p.m_crew) * v**2

    # Synthetic IMU at a hull-mounted point (x,z) in body frame with
        pitch theta
    # In 2D (surge + pitch): sensor acceleration in inertial frame:
    # a_sensor = a_surge * i_hat + alpha x r terms from pitch.
    # We approximate: ax ~ a + (-omega^2 * x) + (-domega * z)??
    # Using planar rigid body kinematics with rotation about y-axis.
    domega = np.gradient(omega, dt)
    ax_body = a - (omega**2) * p.imu_x - domega * p.imu_z
    az_body = - (omega**2) * p.imu_z + domega * p.imu_x

    # Add gravity in body coordinates (pitch rotates gravity)
    g = 9.80665
    # gravity components in body frame for pitch theta (x forward, z up
        )
    g_x = -g * np.sin(theta)
    g_z = -g * np.cos(theta)
    ax_meas = ax_body + g_x
    az_meas = az_body + g_z

    # Provide a minimal IMU: ax, az, gyro_y
    imu = {
        "t": t,
        "ax": ax_meas,
        "ay": np.zeros_like(t),
        "az": az_meas,
        "gx": np.zeros_like(t),
        "gy": omega,
        "gz": np.zeros_like(t),
    }

    return {
        "t": t,
        "x": x,
            "v": v,
            "a": a,
            "theta": theta,
            "omega": omega,
            "s": s,
            "s_dot": s_dot,
            "s_ddot": s_ddot,
            "F_thrust": F,
            "F_drag": Fd,
            "k_drag_eff": k_eff,
            "E_in": E_in,
            "E_drag": E_drag,
            "E_internal": E_internal,
            "E_k": Ek,
            "imu": imu,
            "T": np.array([T], dtype=np.float64),
        }


def disturbance_metrics(sim: Dict[str, Array], dt: float) -> Dict[str,
    float]:
    """Compute a small set of disturbance/loss metrics from a
        simulation."""
    v = sim["v"]
    a = sim["a"]
    theta = sim.get("theta", np.zeros_like(v))
    omega = sim.get("omega", np.zeros_like(v))

    v_mean = float(np.mean(v[int(0.2 * len(v)) :]))
    v_std = float(np.std(v[int(0.2 * len(v)) :]))

    # Excess drag power compared to constant-speed reference
    Fd = sim["F_drag"]
    P_drag = Fd * v
    P_ref = float(np.mean(P_drag[int(0.2 * len(v)) :]))

    # roughness (jerk RMS)
    jerk = np.gradient(a, dt)

    return {
        "v_mean": v_mean,
        "v_std": v_std,
        "cv_v": v_std / max(v_mean, 1e-9),
        "jerk_rms": float(np.sqrt(np.mean(jerk**2))),
        "theta_rms": float(np.sqrt(np.mean(theta**2))),
        "omega_rms": float(np.sqrt(np.mean(omega**2))),
        "drag_power_mean": P_ref,
    }
```

E.2 IMU Processing Pipeline (`imu_analysis.py`)

```
"""imu_analysis.py

IMU processing pipeline for rowing signals.

This module is written to be used in two ways:
1) Demonstration on synthetic IMU signals generated by code/
    rowing_simulator.py.
2) Drop-in for real IMU CSV logs (boat-mounted or athlete-mounted).

Schema expected for CSV:
    t  (seconds, monotonic)
    ax, ay, az  (m/s^2, sensor frame)
    gx, gy, gz  (rad/s, sensor frame)

Key steps
---------
- Preprocessing: resampling, filtering, outlier clipping
- Gravity removal & orientation: complementary filter (gyro + accel
    correction)
- Stroke segmentation: peak detection on bandpassed "drive" energy
    proxy
- Feature extraction: time-domain + spectral metrics capturing
    disturbance
- Disturbance Risk Index: composite score (dimensionless) designed for
    ranking

NOTE: This file intentionally avoids dependencies beyond numpy/scipy/
    matplotlib
for ease of replication.
"""

from __future__ import annotations

from dataclasses import dataclass
from typing import Dict, Iterable, Optional, Tuple

import numpy as np
```

```python
from scipy.signal import butter, filtfilt, find_peaks, resample
from scipy.fft import rfft, rfftfreq

G = 9.80665


@dataclass
class IMUData:
    t: np.ndarray
    a: np.ndarray  # shape (N,3)
    w: np.ndarray  # shape (N,3)


def load_csv(path: str, delimiter: str = ",") -> IMUData:
    """Load an IMU CSV with columns: t, ax, ay, az, gx, gy, gz."""
    raw = np.genfromtxt(path, delimiter=delimiter, names=True)
    t = np.asarray(raw["t"], dtype=float)
    a = np.vstack([raw["ax"], raw["ay"], raw["az"]]).T.astype(float)
    w = np.vstack([raw["gx"], raw["gy"], raw["gz"]]).T.astype(float)
    # Drop NaNs
    ok = np.isfinite(t) & np.all(np.isfinite(a), axis=1) & np.all(np.
        isfinite(w), axis=1)
    t, a, w = t[ok], a[ok], w[ok]
    # Enforce monotonic time
    order = np.argsort(t)
    return IMUData(t=t[order], a=a[order], w=w[order])


def _butter_bandpass(low_hz: float, high_hz: float, fs: float, order:
    int = 4):
    nyq = 0.5 * fs
    low = max(low_hz / nyq, 1e-6)
    high = min(high_hz / nyq, 0.999999)
    return butter(order, [low, high], btype="band")


def _butter_lowpass(cut_hz: float, fs: float, order: int = 4):
    nyq = 0.5 * fs
    cut = min(cut_hz / nyq, 0.999999)
    return butter(order, cut, btype="low")


def resample_to_uniform(imu: IMUData, fs: float) -> IMUData:
    """Resample to uniform sample rate fs (Hz)."""
    t0, tf = float(imu.t[0]), float(imu.t[-1])
    n = int(np.floor((tf - t0) * fs)) + 1
    t_u = t0 + np.arange(n) / fs
    # Interpolate each channel
    a_u = np.column_stack([np.interp(t_u, imu.t, imu.a[:, i]) for i in range(3)])
    w_u = np.column_stack([np.interp(t_u, imu.t, imu.w[:, i]) for i in range(3)])
    return IMUData(t=t_u, a=a_u, w=w_u)


def clip_outliers(x: np.ndarray, k: float = 6.0) -> np.ndarray:
    """Soft clip using median absolute deviation."""
    med = np.median(x, axis=0)
    mad = np.median(np.abs(x - med), axis=0) + 1e-12
    z = (x - med) / (1.4826 * mad)
    return med + np.clip(z, -k, k) * (1.4826 * mad)


def complementary_orientation(
    a: np.ndarray,
    w: np.ndarray,
    fs: float,
    alpha: float = 0.02,
) -> Tuple[np.ndarray, np.ndarray]:
    """Estimate orientation (roll, pitch) and gravity-compensated acceleration.

    We use a minimal complementary filter:
      - Integrate gyro to get roll/pitch
      - Correct slowly toward accel-derived roll/pitch

    Returns:
      euler: (N,2) roll,pitch in radians
      a_lin: (N,3) linear acceleration in the sensor frame (gravity removed)

    Assumes the dominant acceleration direction over long windows is gravity.
    """
    dt = 1.0 / fs
    N = a.shape[0]
    roll = np.zeros(N)
    pitch = np.zeros(N)

    # Initialize from accelerometer
    ax0, ay0, az0 = a[0]
    roll[0] = np.arctan2(ay0, az0)
    pitch[0] = np.arctan2(-ax0, np.sqrt(ay0**2 + az0**2))

    for k in range(1, N):
        gx, gy, gz = w[k]
        # Small-angle integration (good for modest angles)
        roll_g = roll[k - 1] + gx * dt
        pitch_g = pitch[k - 1] + gy * dt

        ax, ay, az = a[k]
        roll_a = np.arctan2(ay, az)
        pitch_a = np.arctan2(-ax, np.sqrt(ay**2 + az**2))

        roll[k] = (1 - alpha) * roll_g + alpha * roll_a
        pitch[k] = (1 - alpha) * pitch_g + alpha * pitch_a

    # Estimate gravity in sensor frame from roll/pitch
    # g_s = R^T * [0,0,g] with yaw neglected
    g_s = np.zeros_like(a)
    cr = np.cos(roll)
    sr = np.sin(roll)
    cp = np.cos(pitch)
    sp = np.sin(pitch)

    # With yaw=0, gravity in sensor frame:
    # gx = -g*sp
    # gy = g*sr*cp
    # gz = g*cr*cp
    g_s[:, 0] = -G * sp
    g_s[:, 1] = G * sr * cp
    g_s[:, 2] = G * cr * cp

    a_lin = a - g_s
    euler = np.column_stack([roll, pitch])
    return euler, a_lin


def segment_strokes(
    t: np.ndarray,
    a_lin: np.ndarray,
    fs: float,
    expected_rate_spm: Tuple[float, float] = (16.0, 44.0),
) -> np.ndarray:
    """Return indices of stroke boundaries (catch-like events).

    A robust proxy for drive onset is a peak in the bandpassed linear
        acceleration
    magnitude.

    Returns
    -------
    idx : array of indices into t marking catch events.
    """
    amag = np.sqrt(np.sum(a_lin**2, axis=1))

    # Bandpass around stroke frequency range
    f_lo = expected_rate_spm[0] / 60.0
    f_hi = expected_rate_spm[1] / 60.0
    b, a = _butter_bandpass(0.6 * f_lo, 3.0 * f_hi, fs, order=3)
    z = filtfilt(b, a, amag)

    # Peak distance in samples
    min_period = 60.0 / expected_rate_spm[1]
    max_period = 60.0 / expected_rate_spm[0]
    min_dist = int(0.6 * min_period * fs)

    # Adaptive threshold: robust scale
    thr = np.median(z) + 0.8 * np.std(z)
    peaks, _ = find_peaks(z, height=thr, distance=min_dist)

    # Remove peaks that imply unrealistic periods
    if peaks.size < 3:
        return peaks

    # Keep peaks with plausible intervals
    dt_peaks = np.diff(t[peaks])
    ok = (dt_peaks > 0.6 * min_period) & (dt_peaks < 1.4 * max_period)
    # Always keep the first peak
    keep = np.concatenate([[True], ok])
    return peaks[keep]


def _band_energy(x: np.ndarray, fs: float, f1: float, f2: float) ->
    float:
    X = np.abs(rfft(x)) ** 2
    f = rfftfreq(x.size, d=1.0 / fs)
    m = (f >= f1) & (f <= f2)
    return float(X[m].sum() / (X.sum() + 1e-12))


def stroke_features(
    t: np.ndarray,
    a_lin: np.ndarray,
    w: np.ndarray,
    euler_rp: np.ndarray,
    idx: np.ndarray,
    fs: float,
) -> Dict[str, np.ndarray]:
    """Compute per-stroke feature vectors."""
    if idx.size < 2:
        return {}

    feats = {
        "dt": [],
        "a_rms": [],
        "a_jerk_rms": [],
        "pitch_rms": [],
        "pitch_rate_rms": [],
        "hi_freq_energy": [],
        "gyro_hi_energy": [],
    }

    for k in range(idx.size - 1):
        i0, i1 = int(idx[k]), int(idx[k + 1])
        if i1 <= i0 + 5:
            continue
        tt = t[i0:i1]
        aa = a_lin[i0:i1]
        ww = w[i0:i1]
        pitch = euler_rp[i0:i1, 1]

        dt = float(tt[-1] - tt[0])
        feats["dt"].append(dt)

        amag = np.sqrt(np.sum(aa**2, axis=1))
        feats["a_rms"].append(float(np.sqrt(np.mean(amag**2))))

        jerk = np.gradient(amag, 1.0 / fs)
        feats["a_jerk_rms"].append(float(np.sqrt(np.mean(jerk**2))))

        feats["pitch_rms"].append(float(np.sqrt(np.mean(pitch**2))))
        feats["pitch_rate_rms"].append(float(np.sqrt(np.mean(ww[:, 1]
            ** 2))))

        # Energy in high-frequency band (proxy for abrupt motion)
        feats["hi_freq_energy"].append(_band_energy(amag - amag.mean(),
             fs, 2.5, 10.0))
        feats["gyro_hi_energy"].append(_band_energy(ww[:, 1] - ww[:,
            1].mean(), fs, 2.5, 10.0))

    return {k: np.asarray(v) for k, v in feats.items()}


def disturbance_risk_index(features: Dict[str, np.ndarray]) -> np.
    ndarray:
    """Composite disturbance risk index per stroke.

    This is a dimensionless, z-scored combination of:
      - acceleration jerk (abrupt internal motion)
      - pitch RMS (attitude disturbance)
      - high-frequency energy (vibration/instability)

    Lower is ``smoother''.
    """
    if not features:
        return np.array([])

    comps = [
        features["a_jerk_rms"],
        features["pitch_rms"],
        features["hi_freq_energy"],
        features["gyro_hi_energy"],
    ]

    Z = []
    for x in comps:
        mu = np.mean(x)
        sd = np.std(x) + 1e-12
        Z.append((x - mu) / sd)
    Z = np.vstack(Z).T

    # Weights can be tuned or estimated; default equal.
    w = np.array([0.35, 0.35, 0.15, 0.15])
    return Z @ w


def process_imu_csv(
    path: str,
    fs: float = 100.0,
    expected_rate_spm: Tuple[float, float] = (16.0, 44.0),
) -> Dict[str, object]:
    """Full pipeline: load, resample, filter, segment, features."""
    imu = load_csv(path)
    imu = resample_to_uniform(imu, fs=fs)
    imu = IMUData(
        t=imu.t,
        a=clip_outliers(imu.a, k=8.0),
        w=clip_outliers(imu.w, k=8.0),
    )

    euler, a_lin = complementary_orientation(imu.a, imu.w, fs=fs, alpha
        =0.03)
    idx = segment_strokes(imu.t, a_lin, fs=fs, expected_rate_spm=
        expected_rate_spm)
    feats = stroke_features(imu.t, a_lin, imu.w, euler, idx, fs=fs)
    R = disturbance_risk_index(feats)

    out = {
        "t": imu.t,
        "a_raw": imu.a,
        "w": imu.w,
        "euler_rp": euler,
        "a_lin": a_lin,
        "stroke_idx": idx,
        "features": feats,
        "risk": R,
    }
    return out


def summarize_risk(risk: np.ndarray) -> Dict[str, float]:
    if risk.size == 0:
        return {"risk_mean": float("nan"), "risk_std": float("nan"), "
            risk_p90": float("nan")}
    return {
        "risk_mean": float(np.mean(risk)),
        "risk_std": float(np.std(risk)),
        "risk_p90": float(np.quantile(risk, 0.90)),
    }


if __name__ == "__main__":
    import argparse

    ap = argparse.ArgumentParser()
    ap.add_argument("csv", help="Path to IMU CSV with columns t,ax,ay,
        az,gx,gy,gz")
    ap.add_argument("--fs", type=float, default=100.0)
    args = ap.parse_args()

    out = process_imu_csv(args.csv, fs=args.fs)
    summ = summarize_risk(out["risk"])
    print("Stroke count:", int(out["risk"].size))
    print("Risk summary:", summ)
```

E.3 Reproduction Script (run_all.py)

```
"""run_all.py

Reproduce all figures for the unified manuscript.

Usage:
  python code/run_all.py

Outputs:
  PNG figures saved into ./figures

Notes
-----
- Overleaf will not execute this code. Run locally to regenerate
    figures.
- The script generates both (i) purely synthetic simulation signals and
  (ii) a mocked "real" IMU dataset obtained by corrupting synthetic
      signals
  with bias, drift, and noise. This mirrors common on-water/erg IMU
      issues.
"""

from __future__ import annotations

from dataclasses import replace
from pathlib import Path

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import least_squares

from rowing_simulator import (
    BaselineParams,
    MultibodyParams,
    simulate_baseline,
    simulate_multibody,
    disturbance_metrics,
)


def _save(fig: plt.Figure, out: Path) -> None:
    out.parent.mkdir(parents=True, exist_ok=True)
    fig.tight_layout()
    fig.savefig(out, dpi=300)
    plt.close(fig)


def _steady_slice(t: np.ndarray, frac_discard: float = 0.35) -> slice:
    i0 = int(frac_discard * len(t))
    return slice(i0, len(t))


def _mock_real_imu(imu: dict, seed: int = 7) -> dict:
    """Return a corrupted copy of IMU streams (bias + random-walk drift
        + noise)."""
    rng = np.random.default_rng(seed)
    t = imu["t"]
    dt = float(np.median(np.diff(t)))

    # Noise levels: modest, plausible for consumer-grade IMUs
    sigma_a = 0.20  # m/s^2
    sigma_g = 0.015  # rad/s

    # Bias + drift (random walk)
    bias_ax = rng.normal(0.0, 0.15)
    bias_az = rng.normal(0.0, 0.15)
    bias_gy = rng.normal(0.0, 0.01)

    rw_ax = np.cumsum(rng.normal(0.0, 0.02, size=len(t))) * np.sqrt(dt)
    rw_az = np.cumsum(rng.normal(0.0, 0.02, size=len(t))) * np.sqrt(dt)
    rw_gy = np.cumsum(rng.normal(0.0, 0.002, size=len(t))) * np.sqrt(dt
        )

    ax = imu["ax"] + bias_ax + rw_ax + rng.normal(0.0, sigma_a, size=
        len(t))
    az = imu["az"] + bias_az + rw_az + rng.normal(0.0, sigma_a, size=
        len(t))
    gy = imu["gy"] + bias_gy + rw_gy + rng.normal(0.0, sigma_g, size=
        len(t))

    out = dict(imu)
    out["ax"] = ax
    out["az"] = az
    out["gy"] = gy
    out["label"] = "mock_real"
    return out


def main() -> None:
    root = Path(__file__).resolve().parents[1]
    figdir = root / "figures"
    figdir.mkdir(exist_ok=True)

    # -------------------- Part I: baseline --------------------
    p = BaselineParams(
        m_total=750.0,
        k_drag=40.0,
        stroke_rate_spm=30.0,
        drive_fraction=0.45,
        thrust_kind="beta",
        F_peak=3200.0,
        beta_a=2.6,
        beta_b=3.2,
    )

    # Keep runs short enough for quick figure regeneration.
    out = simulate_baseline(p, t_end=20.0, dt=2e-3)
    t, v = out["t"], out["v"]
    sl = _steady_slice(t, 0.35)

    # Speed profile
    for fname in ["baseline_speed_profile.png", "partI_speed_profile.
        png"]:
```

```
        fig = plt.figure()
        plt.plot(t[sl], v[sl])
        plt.xlabel("time (s)")
        plt.ylabel("surge speed v (m/s)")
        plt.title("Baseline surge speed (steady segment)")
        _save(fig, figdir / fname)

    # Energy budget
    for fname in ["baseline_energy_budget.png", "partI_energy_budget.
        png"]:
        fig = plt.figure()
        plt.plot(t[sl], out["E_in"][sl], label=r"$E_{in}$")
        plt.plot(t[sl], out["E_drag"][sl], label=r"$E_{drag}$")
        plt.plot(t[sl], out["E_k"][sl], label=r"$E_k$")
        plt.xlabel("time (s)")
        plt.ylabel("energy (J)")
        plt.legend()
        plt.title("Energy accounting (baseline)")
        _save(fig, figdir / fname)

    # Mean-speed sweep over drag coefficient and stroke rate
    kd_vals = np.linspace(25.0, 70.0, 6)
    spm_vals = np.linspace(22.0, 40.0, 6)
    v_mean = np.zeros((len(kd_vals), len(spm_vals)))

    for i, kd in enumerate(kd_vals):
        for j, spm in enumerate(spm_vals):
            pp = replace(p, k_drag=float(kd), stroke_rate_spm=float(spm
                ))
            # Short horizon + coarser step: sufficient for a steady-
                speed estimate
            # while keeping total runtime modest.
            oo = simulate_baseline(pp, t_end=10.0, dt=6e-3)
            ss = _steady_slice(oo["t"], 0.5)
            v_mean[i, j] = float(np.mean(oo["v"][ss]))

    fig = plt.figure()
    plt.imshow(
        v_mean,
        aspect="auto",
        origin="lower",
        extent=[spm_vals[0], spm_vals[-1], kd_vals[0], kd_vals[-1]],
    )
    plt.xlabel("stroke rate (spm)")
    plt.ylabel(r"$k_d$ (N s$^2$ m$^{-2}$)")
    plt.title("Mean speed sweep (baseline)")
    plt.colorbar(label="m/s")
    _save(fig, figdir / "partI_speed_sweep.png")

    # Slice sensitivity
    mid_j = len(spm_vals) // 2
    fig = plt.figure()
    plt.plot(kd_vals, v_mean[:, mid_j])
    plt.xlabel(r"$k_d$ (N s$^2$ m$^{-2}$)")
    plt.ylabel("mean speed (m/s)")
    plt.title(f"Sensitivity to drag at {spm_vals[mid_j]:.0f} spm")
    _save(fig, figdir / "partI_sweep_drag_mass_rate.png")

    # -------------------- Part II: multibody + pitch surrogate
        --------------------
    mp = MultibodyParams(
        **p.__dict__,
        m_hull=250.0,
        m_crew=500.0,
        s_max=0.85,
        recovery_shape="blend",
        recovery_blend=0.25,
        Iyy=1200.0,
        k_theta=5.0e4,
        c_theta=2.5e3,
        k_trim=1.8e4,
        alpha_pitch=2.5,
        beta_pitchrate=0.15,
    )

    mout = simulate_multibody(mp, t_end=18.0, dt=4e-3)
    tt, vv, th = mout["t"], mout["v"], mout["theta"]
    sl2 = _steady_slice(tt, 0.35)

    # Pitch + speed (two axes)
    fig, ax1 = plt.subplots()
    ax1.plot(tt[sl2], vv[sl2])
    ax1.set_xlabel("time (s)")
    ax1.set_ylabel("v (m/s)")
    ax2 = ax1.twinx()
    ax2.plot(tt[sl2], th[sl2])
    ax2.set_ylabel(r"pitch $\theta$ (rad)")
    ax1.set_title("Pitch-coupled multibody simulation")
    _save(fig, figdir / "partII_pitch_and_speed.png")

    # Smoothness sweep: blend cosine->poly recovery
    blends = np.linspace(0.0, 1.0, 7)
    jerk_rms = np.zeros_like(blends)
    theta_rms = np.zeros_like(blends)
    cv_v = np.zeros_like(blends)

    for k, w in enumerate(blends):
        mpi = replace(mp, recovery_blend=float(w))
        so = simulate_multibody(mpi, t_end=14.0, dt=4e-3)
        m = disturbance_metrics(so, dt=4e-3)
        jerk_rms[k] = m["jerk_rms"]
        theta_rms[k] = m["theta_rms"]
        cv_v[k] = m["cv_v"]

    fig = plt.figure()
    plt.plot(blends, jerk_rms, label="jerk RMS")
    plt.plot(blends, theta_rms, label=r"$\theta$ RMS")
    plt.plot(blends, cv_v, label="CV(v)")
    plt.xlabel("recovery smoothness blend (0=cosine, 1=poly)")
    plt.ylabel("metric value")
    plt.legend()
    plt.title("Technique smoothness sweep (proxy disturbance metrics)")
    _save(fig, figdir / "partII_smoothness_sweep.png")

    # -------------------- Part III: synthetic + mocked-real IMU
        --------------------
    imu_syn = mout["imu"]
    imu_real = _mock_real_imu(imu_syn, seed=7)

    # Synthetic + mocked real overlay
    fig = plt.figure(figsize=(7.2, 7.5))
    axa = plt.subplot(3, 1, 1)
    axa.plot(tt[sl2], imu_syn["ax"][sl2], label="synthetic")
    axa.plot(tt[sl2], imu_real["ax"][sl2], label="mock real")
    axa.set_ylabel("ax (m/s$^2$)")
    axa.legend()

    axb = plt.subplot(3, 1, 2)
    axb.plot(tt[sl2], imu_syn["az"][sl2], label="synthetic")
    axb.plot(tt[sl2], imu_real["az"][sl2], label="mock real")
    axb.set_ylabel("az (m/s$^2$)")

```

```python
    axc = plt.subplot(3, 1, 3)
    axc.plot(tt[sl2], imu_syn["gy"][sl2], label="synthetic")
    axc.plot(tt[sl2], imu_real["gy"][sl2], label="mock real")
    axc.set_ylabel("gy (rad/s)")
    axc.set_xlabel("time (s)")

    plt.suptitle("Synthetic IMU vs. mocked real IMU (bias/drift/noise)"
        )
    _save(fig, figdir / "partIII_synthetic_imu.png")

    # -------------------- Part III: grey-box system identification
        demo --------------------
    # Fit a small subset of multibody parameters using noisy IMU
        signals.
    # We fit alpha_pitch (drag augmentation) and k_trim (pitch
        excitation) to minimize
    # mismatch in ax and gy on a short window.

    t_fit = tt[sl2]
    ax_meas = imu_real["ax"][sl2]
    gy_meas = imu_real["gy"][sl2]

    # Downsample for faster fitting
    step = 5
    t_fit = t_fit[::step]
    ax_meas = ax_meas[::step]
    gy_meas = gy_meas[::step]

    def simulate_for_params(alpha_pitch: float, k_trim: float) -> tuple
        [np.ndarray, np.ndarray]:
        mpi = replace(mp, alpha_pitch=float(alpha_pitch), k_trim=float(
            k_trim))
        so = simulate_multibody(mpi, t_end=float(t_fit[-1]), dt=4e-3)
        # Align by truncation
        t0 = so["t"]
        imu0 = so["imu"]
        # sample at t_fit indices using nearest
        idx = np.searchsorted(t0, t_fit)
        idx = np.clip(idx, 0, len(t0) - 1)
        return imu0["ax"][idx], imu0["gy"][idx]

    def residual(q: np.ndarray) -> np.ndarray:
        alpha, ktrim = q
        ax_hat, gy_hat = simulate_for_params(alpha, ktrim)
        # Normalize residual components to balance units
        r1 = (ax_hat - ax_meas) / 0.5
        r2 = (gy_hat - gy_meas) / 0.05
        return np.concatenate([r1, r2])

    q0 = np.array([1.5, 1.0e4], dtype=float)
    bounds = (
        np.array([0.0, 1.0e3], dtype=float),
        np.array([10.0, 6.0e4], dtype=float),
    )

    res = least_squares(residual, q0, bounds=bounds, max_nfev=6)
    alpha_hat, ktrim_hat = res.x

    ax_hat, gy_hat = simulate_for_params(alpha_hat, ktrim_hat)

    fig = plt.figure(figsize=(7.2, 6.8))
    ax1 = plt.subplot(2, 1, 1)
    ax1.plot(t_fit, ax_meas, label="measured (mock real)")
    ax1.plot(t_fit, ax_hat, label="model fit")
    ax1.set_ylabel("ax (m/s$^2$)")
    ax1.legend()
    ax1.set_title(f"Grey-box SysID fit: alpha={alpha_hat:.2f}, k_trim={
        ktrim_hat:.0f}")

    ax2 = plt.subplot(2, 1, 2)
    ax2.plot(t_fit, gy_meas, label="measured (mock real)")
    ax2.plot(t_fit, gy_hat, label="model fit")
    ax2.set_ylabel("gy (rad/s)")
    ax2.set_xlabel("time (s)")
    ax2.legend()

    _save(fig, figdir / "partIII_sysid_fit.png")

    # -------------------- Console summary --------------------
    m_base = disturbance_metrics(out, dt=2e-3)
    m_multi = disturbance_metrics(mout, dt=4e-3)
    print("Generated figures in", figdir)
    print("Baseline disturbance metrics:", m_base)
    print("Multibody disturbance metrics:", m_multi)
    print("SysID estimated [alpha_pitch, k_trim] =", [float(alpha_hat),
         float(ktrim_hat)])


```

```
if __name__ == "__main__":
    main()
```

E.4 Convenience Wrapper (rowing_model.py)

```
"""rowing_model.py

Compatibility wrapper (baseline project naming).

The original project used ‘rowing_model.py‘ as the entry point for generating
figures from a simplified thrust--drag simulator.

In the unified manuscript project, the primary figure generation lives in:
  - code/run_all.py (fast, main paper figures)
  - code/make_missing_figures.py (heavier extended figures)
  - rowing_advanced_models.py (cohort ranking demo)

This wrapper simply runs the main figure pipeline from the project root.

Run:
  python rowing_model.py
"""

from __future__ import annotations

from pathlib import Path
import sys

ROOT = Path(__file__).resolve().parent
CODE_DIR = ROOT / "code"
if str(CODE_DIR) not in sys.path:
    sys.path.insert(0, str(CODE_DIR))

import run_all


if __name__ == "__main__":
    run_all.main()
```

E.5 Advanced Extensions Demo (rowing_advanced_models.py)

```
"""rowing_advanced_models.py

Advanced, publication-oriented model extensions and analysis utilities.

This script demonstrates how the unified simulator and IMU pipeline can be used
for *program-scale* analysis: generating technique-sensitive features and a
"disturbance risk" ranking that is not based solely on erg power.

It intentionally avoids any product/app discussion. The objective is scientific:
to quantify technique-driven loss channels and estimate how robustly they can be
measured using land-based IMU data.

What it does
------------
1) Simulates a synthetic athlete cohort with controlled technique parameters:
   - recovery smoothness (slide acceleration profile)
   - pitch sensitivity (drag augmentation strength)
   - slight random variation in stroke rate and peak force
2) Produces two datasets:
   - purely synthetic IMU (model-consistent)
   - "mock real" IMU created by adding bias, drift, and noise
3) Extracts stroke-level features and computes a disturbance-risk index (DRI).
4) Saves ranked CSVs and generates feature-space visualizations.

Usage
-----
  python rowing_advanced_models.py

Outputs
-------
- data/athlete_risk_ranking_synthetic.csv
- data/athlete_risk_ranking_mockreal.csv
- data/athlete_risk_ranking_robust.csv
- figures/partIII_feature_space.png
- figures/partIII_feature_robustness.png
"""

```

```python
from __future__ import annotations

import os
import sys
from dataclasses import replace
from pathlib import Path

import numpy as np
import matplotlib

matplotlib.use("Agg")
import matplotlib.pyplot as plt

# Make code/ importable when run from repository root.
ROOT = Path(__file__).resolve().parent
CODE_DIR = ROOT / "code"
if str(CODE_DIR) not in sys.path:
    sys.path.insert(0, str(CODE_DIR))

from rowing_simulator import BaselineParams, MultibodyParams,
    simulate_multibody
from imu_analysis import (
    IMUData,
    resample_to_uniform,
    complementary_orientation,
    segment_strokes,
    stroke_features,
    disturbance_risk_index,
)


def _ensure_dirs() -> None:
    (ROOT / "figures").mkdir(parents=True, exist_ok=True)
    (ROOT / "data").mkdir(parents=True, exist_ok=True)


def _mock_real_imu(imu: dict[str, np.ndarray], seed: int = 7) -> dict[
    str, np.ndarray]:
    """Corrupt synthetic IMU with bias/drift/noise to mimic field
        recordings."""
    rng = np.random.default_rng(seed)
    t = imu["t"]
    dt = float(np.mean(np.diff(t)))

    sigma_a = 0.25
    sigma_g = 0.018
    bias_ax = 0.12
    bias_az = -0.08
    bias_g = 0.004

    rw_ax = np.cumsum(rng.normal(0.0, 0.008, size=len(t))) * np.sqrt(dt
        )
    rw_az = np.cumsum(rng.normal(0.0, 0.008, size=len(t))) * np.sqrt(dt
        )
    rw_g = np.cumsum(rng.normal(0.0, 0.00035, size=len(t))) * np.sqrt(
        dt)

    ax = imu["ax"] + bias_ax + rw_ax + rng.normal(0.0, sigma_a, size=
        len(t))
    az = imu["az"] + bias_az + rw_az + rng.normal(0.0, sigma_a, size=
        len(t))
    gy = imu["gy"] + bias_g + rw_g + rng.normal(0.0, sigma_g, size=len(
        t))

    out = dict(imu)
    out["ax"] = ax
    out["az"] = az
    out["gy"] = gy
    return out


def _athlete_params(seed: int, blend: float, alpha_pitch: float) ->
    MultibodyParams:
    rng = np.random.default_rng(seed)

    spm = 30.0 + rng.normal(0.0, 0.9)
    F_peak = 3200.0 * (1.0 + rng.normal(0.0, 0.05))

    p = BaselineParams(
        m_total=750.0,
        k_drag=40.0,
        stroke_rate_spm=float(spm),
        drive_fraction=0.45,
        thrust_kind="beta",
        F_peak=float(F_peak),
        beta_a=2.6,
        beta_b=3.2,
    )
```

```python

    mp = MultibodyParams(
        **p.__dict__,
        m_hull=250.0,
        m_crew=500.0,
        s_max=0.85,
        recovery_shape="blend",
        recovery_blend=float(blend),
        Iyy=1200.0,
        k_theta=5.0e4,
        c_theta=2.5e3,
        k_trim=1.8e4,
        alpha_pitch=float(alpha_pitch),
        beta_pitchrate=0.15,
    )
    return mp


def _features_from_imu(imu: dict[str, np.ndarray], fs_hz: float =
    100.0) -> tuple[dict[str, np.ndarray], np.ndarray]:
    """Run the IMU pipeline and return (stroke-level features, DRI)."""
    t = imu["t"].astype(float)
    z = np.zeros_like(t)
    a = np.column_stack([imu["ax"], imu.get("ay", z), imu["az"]]).
        astype(float)
    w = np.column_stack([imu.get("gx", z), imu["gy"], imu.get("gz", z)
        ]).astype(float)

    data = IMUData(t=t, a=a, w=w)
    data = resample_to_uniform(data, fs=fs_hz)

    euler, a_lin = complementary_orientation(data.a, data.w, fs=fs_hz,
        alpha=0.03)
    idx = segment_strokes(data.t, a_lin, fs=fs_hz)
    feats = stroke_features(data.t, a_lin, data.w, euler, idx, fs=fs_hz
        )
    dri = disturbance_risk_index(feats)
    return feats, dri


def main() -> None:
    _ensure_dirs()

    # Cohort grid: 6x6 = 36 athletes
    blends = np.linspace(0.0, 1.0, 6)
    alphas = np.linspace(1.2, 3.4, 6)

    summaries_syn = []
    summaries_real = []

    pts_syn = []
    pts_real = []

    athlete_id = 1
    for b in blends:
        for a in alphas:
            mp = _athlete_params(seed=1000 + athlete_id, blend=float(b)
                , alpha_pitch=float(a))

            out = simulate_multibody(mp, t_end=18.0, dt=0.012)
            imu_syn = out["imu"]
            imu_real = _mock_real_imu(imu_syn, seed=2000 + athlete_id)

            feats_syn, dri_syn = _features_from_imu(imu_syn)
            feats_real, dri_real = _features_from_imu(imu_real)

            # Aggregate
            syn = {
                "athlete_id": f"A{athlete_id:02d}",
                "recovery_blend": float(b),
                "alpha_pitch": float(a),
                "dri_mean": float(np.mean(dri_syn)),
                "dri_p90": float(np.quantile(dri_syn, 0.90)),
                "jerk_rms_mean": float(np.mean(feats_syn["a_jerk_rms"])
                    ),
                "pitch_rms_mean": float(np.mean(feats_syn["pitch_rms"])
                    ),
                "pitch_rate_rms_mean": float(np.mean(feats_syn["
                    pitch_rate_rms"])),
                "hi_freq_energy_mean": float(np.mean(feats_syn["
                    hi_freq_energy"])),
            }
            real = {
                "athlete_id": f"A{athlete_id:02d}",
                "recovery_blend": float(b),
                "alpha_pitch": float(a),
                "dri_mean": float(np.mean(dri_real)),
                "dri_p90": float(np.quantile(dri_real, 0.90)),
                "jerk_rms_mean": float(np.mean(feats_real["a_jerk_rms"])),
                "pitch_rms_mean": float(np.mean(feats_real["pitch_rms"])),
                "pitch_rate_rms_mean": float(np.mean(feats_real["pitch_rate_rms"])),
                "hi_freq_energy_mean": float(np.mean(feats_real["hi_freq_energy"])),
            }

            summaries_syn.append(syn)
            summaries_real.append(real)

            pts_syn.append((syn["jerk_rms_mean"], syn["pitch_rms_mean"], syn["dri_mean"]))
            pts_real.append((real["jerk_rms_mean"], real["pitch_rms_mean"], real["dri_mean"]))

            athlete_id += 1

    # Write CSVs
    def write_csv(rows, path: Path):
        header = list(rows[0].keys())
        with open(path, "w", encoding="utf-8") as f:
            f.write(",".join(header) + "\n")
            for r in sorted(rows, key=lambda rr: rr["dri_mean"]):
                f.write(",".join(str(r[h]) for h in header) + "\n")

    write_csv(summaries_syn, ROOT / "data" / "athlete_risk_ranking_synthetic.csv")
    write_csv(summaries_real, ROOT / "data" / "athlete_risk_ranking_mockreal.csv")

    # A robustness score: average normalized rank across synthetic + mockreal
    syn_rank = {r["athlete_id"]: i for i, r in enumerate(sorted(summaries_syn, key=lambda rr: rr["dri_mean"]))}
    real_rank = {r["athlete_id"]: i for i, r in enumerate(sorted(summaries_real, key=lambda rr: rr["dri_mean"]))}
    n = len(summaries_syn)
    robust = []
    for r in summaries_syn:
        aid = r["athlete_id"]
```

```
            score = 0.5 * (syn_rank[aid] / (n - 1) + real_rank[aid] / (n -
                1))
            robust.append({
                "athlete_id": aid,
                "robust_rank_score": float(score),
                "rank_synthetic": int(syn_rank[aid] + 1),
                "rank_mockreal": int(real_rank[aid] + 1),
            })
        robust_sorted = sorted(robust, key=lambda rr: rr["robust_rank_score
            "])

        # Write robustness CSV
        header = list(robust_sorted[0].keys())
        with open(ROOT / "data" / "athlete_risk_ranking_robust.csv", "w",
            encoding="utf-8") as f:
            f.write(",".join(header) + "\n")
            for r in robust_sorted:
                f.write(",".join(str(r[h]) for h in header) + "\n")

        # Feature-space plots
        Xs = np.array(pts_syn)
        Xr = np.array(pts_real)

        # Synthetic feature space
        fig = plt.figure(figsize=(6.6, 4.3))
        sc = plt.scatter(Xs[:, 0], Xs[:, 1], c=Xs[:, 2])
        plt.xlabel("Mean jerk RMS (arb. units)")
        plt.ylabel("Mean pitch RMS (rad)")
        plt.title("Synthetic athlete cohort: disturbance-risk feature space
            ")
        cb = plt.colorbar(sc)
        cb.set_label("Mean disturbance-risk index")
        fig.tight_layout()
        fig.savefig(ROOT / "figures" / "partIII_feature_space.png", dpi
            =300)
        plt.close(fig)

        # Robustness plot: synthetic vs mock-real DRI per athlete
        syn_dri = np.array([r["dri_mean"] for r in summaries_syn])
        real_dri = np.array([r["dri_mean"] for r in summaries_real])

        fig = plt.figure(figsize=(6.2, 4.3))
        plt.scatter(syn_dri, real_dri)
        m = float(max(syn_dri.max(), real_dri.max()))
```

Appendix: Expanded Explanations and Derivations

This appendix collects expanded explanations (modeling choices, derivations, and interpretation) that were developed alongside the main text. It is written to be self-contained and does not assume access to any intermediate draft documents.

A. Thrust forcing profiles and normalization

Let T denote the stroke period and $\delta \in (0,1)$ the drive fraction. Define stroke phase $\phi = t \bmod T$, and the drive interval $\phi \in [0, \delta T)$.

Square-wave drive–recovery forcing. A transparent limiting case is

$$F(\phi) = \begin{cases} F_0, & 0 \le \phi < \delta T \\ 0, & \delta T \le \phi < T. \end{cases} \tag{1}$$

Here F_0 represents an effective propulsive force transmitted to the shell during the drive.

Smooth parametric forcing. To better approximate measured handle/oar force curves while keeping a low-dimensional model, we use a smooth family supported on the drive interval. A convenient choice is a raised-cosine "bump"

$$F(\phi) = \begin{cases} F_{\max} \sin^p(\pi\, \phi/(\delta T)), & 0 \le \phi < \delta T \\ 0, & \delta T \le \phi < T, \end{cases} \tag{2}$$

where $p \ge 1$ controls the sharpness. $p = 1$ yields a half-sine; larger p produces a flatter mid-drive plateau with sharper onset/offset.

Impulse (work) consistency across shapes. If different forcing shapes are compared, it is useful to hold fixed either (i) average force per cycle, $\bar{F} = (1/T)\int_0^T F(\phi)\, d\phi$, or (ii) total impulse per cycle, $J = \int_0^T F(\phi)\, d\phi = \bar{F}T$. For the square wave, $J = F_0\, \delta T$. For the raised-cosine family,

$$J = F_{\max} \int_0^{\delta T} \sin^p(\pi\, \phi/(\delta T))\; d\phi = F_{\max}\, \delta T\, \frac{\Gamma\left(\frac{p+1}{2}\right)}{\sqrt{\pi}\,\Gamma\left(\frac{p}{2}+1\right)}. \tag{3}$$

Thus, fixing J (or $\bar{F}$) determines $F_{\max}$ for a chosen p.

B. Energy accounting and "efficiency" definitions

In the surge-only baseline, the shell speed $v(t)$ obeys

$$m\,\dot{v} = F(t) - D(v), \qquad D(v) = k_d\, v|v|, \tag{4}$$

with total effective mass m and quadratic drag coefficient k_d.

Multiplying by v gives a power balance:

$$\frac{d}{dt}\left(\tfrac{1}{2}mv^2\right) = F(t)v - k_d|v|^3. \tag{5}$$

Integrating over an interval $[t_0, t_1]$ yields

$$W_{\text{in}} = \Delta K + W_{\text{drag}}, \quad W_{\text{in}} = \int_{t_0}^{t_1} Fv\,dt, \quad W_{\text{drag}} = \int_{t_0}^{t_1} k_d |v|^3\,dt, \quad \Delta K = \tfrac{1}{2}m\big(v^2(t_1) - v^2(t_0)\big). \tag{6}$$

Cycle-averaged steady state. Over one full stroke in periodic steady state, $\Delta K \approx 0$, so input work per cycle approximately equals drag dissipation per cycle: $W_{\text{in}} \approx W_{\text{drag}}$. The model therefore emphasizes that, at fixed average speed, essentially all mechanical work ultimately leaves the system through hydrodynamic drag (with intermediate storage and return through kinetic energy).

Within-cycle "storage" fraction. A commonly reported diagnostic is the fraction of drive-phase input work that appears as a positive kinetic-energy increment during the drive. Defining the drive interval $[t_d^{\text{start}}, t_d^{\text{end}}]$, one may compute

$$\eta_{\text{drive}} \equiv \frac{\max\{0,\, K(t_d^{\text{end}}) - K(t_d^{\text{start}})\}}{\int_{t_d^{\text{start}}}^{t_d^{\text{end}}} Fv\,dt}. \tag{7}$$

This "storage" metric is useful for comparing stroke shapes and timing, but it is not a thermodynamic efficiency: kinetic energy accumulated in the drive is largely dissipated in later portions of the cycle.

C. Why velocity oscillations increase effective drag

Quadratic drag induces a *cubic* dissipation rate $k_d|v|^3$. Let $v(t) = \bar{v} + \tilde{v}(t)$, where $\bar{v}$ is the cycle mean and $\tilde{v}$ is a zero-mean fluctuation. Because $x \mapsto |x|^3$ is convex,

$$\langle |v|^3 \rangle \geq |\langle v \rangle|^3 = |\bar{v}|^3, \tag{8}$$

with strict inequality when fluctuations are present. A compact measure is the *drag augmentation factor*

$$\phi \equiv \frac{\langle |v|^3 \rangle}{|\bar{v}|^3} \geq 1. \tag{9}$$

Thus, for a fixed mean speed $\bar{v}$, larger within-stroke oscillations necessarily increase mean drag power and therefore increase the work required to maintain that speed. This mechanism provides a mathematical pathway linking technique-induced disturbances (abrupt accelerations, recovery checks, and attitude excursions that feed back into v) to measurable performance penalties.

D. Reduced-order multibody extension: slide mass coupling

To isolate the mechanical effect of crew mass motion without introducing a full biomechanical digital twin, we treat the system as (i) a shell/hull mass m_b translating with surge speed $v_b(t)$ and (ii) an effective crew mass m_c moving relative to the shell along the slide coordinate $x_s(t)$ (positive toward the stern). The crew absolute surge speed is $v_c = v_b + \dot{x}_s$.

The total linear momentum is

$$P = m_b v_b + m_c v_c = (m_b + m_c)v_b + m_c \dot{x}_s. \tag{10}$$

External forces in surge are propulsive forcing $F(t)$ and hydrodynamic drag $D(v_b, \theta)$ (with θ a pitch surrogate described below). Newton's law yields

$$\dot{P} = F(t) - D(v_b, \theta) \quad \Rightarrow \quad (m_b + m_c)\dot{v}_b = F(t) - D(v_b, \theta) - m_c \ddot{x}_s. \tag{11}$$

The term $-m_c \ddot{x}_s$ captures the familiar "check" during recovery: rapid seat acceleration produces a reaction on the shell that reduces v_b and (via Section C) increases effective drag losses.

Pitch surrogate and drag coupling. A minimal surrogate for attitude dynamics is to introduce a scalar pitch state $\theta(t)$ driven by slide motion and/or surge acceleration, then couple θ to drag through a multiplicative factor:

$$\dot{\theta} = g\big(\dot{x}_s, \ddot{x}_s, v_b\big), \qquad D(v_b, \theta) = k_d \left(1 + \alpha|\theta|\right) v_b |v_b|, \tag{12}$$

with coupling strength $\alpha \geq 0$. This construction is intentionally "grey-box": it captures the experimentally observed fact that departures from optimal trim/pitch can measurably change resistance without requiring CFD or detailed hull geometry.

E. IMU-informed parameter estimation blueprint

Wearable inertial measurement units (IMUs) provide tri-axial angular velocity and linear acceleration. A reduced-order pipeline can connect these signals to the disturbance parameters that matter most in the models above.

E.1 Orientation and gravity removal. First estimate orientation (quaternion or rotation matrix) using a complementary or gradient-based filter that fuses gyro integration with accelerometer-based gravity cues. Transform measured acceleration to a body-fixed frame and subtract gravity to obtain "specific force".

E.2 Stroke segmentation and features. Segment strokes using repeatable events (e.g., peaks in fore–aft acceleration, angular-velocity signatures, or filtered energy in a drive-band). Extract features such as:

- drive fraction δ and stroke rate $1/T$;
- peak and RMS accelerations; recovery "check" magnitude (negative acceleration impulse);
- time–frequency content (e.g., short-time Fourier features) that correlate with abruptness of mass motion or oar-water interaction.

E.3 Grey-box system identification (SysID). Let $y(t)$ denote an IMU-derived observable (e.g., surge acceleration proxy or a pitch proxy), and let $\hat{y}(t; \vartheta)$ be the model prediction given parameters ϑ (drag k_d, coupling α, slide-motion parameters, etc.). Estimate parameters by solving

$$\vartheta^* = \arg\min_{\vartheta} \sum_i \big\| y(t_i) - \hat{y}(t_i; \vartheta) \big\|^2 + \lambda\, R(\vartheta), \tag{13}$$

with regularization $R(\vartheta)$ enforcing plausible ranges and smoothness across strokes. The key design goal is not "perfect reconstruction" but stable estimation of the disturbance-sensitive parameters that drive loss channels: those controlling ϕ (drag augmentation from oscillations), recovery checks ($\ddot{x}_s$), and trim-related drag modulation ($\alpha|\theta|$).

E.4 Interpretation. Because the models are low-dimensional and interpretable, estimated parameters can be compared across athletes, rigging, and technical interventions. This supports a practical loss-minimization view of performance: reduce the disturbance channels that inflate ϕ and amplify $D(v_b, \theta)$, rather than pursuing speed solely through increased peak power.