Further Insight: Physics of an Electrodes-Driven Rotor System

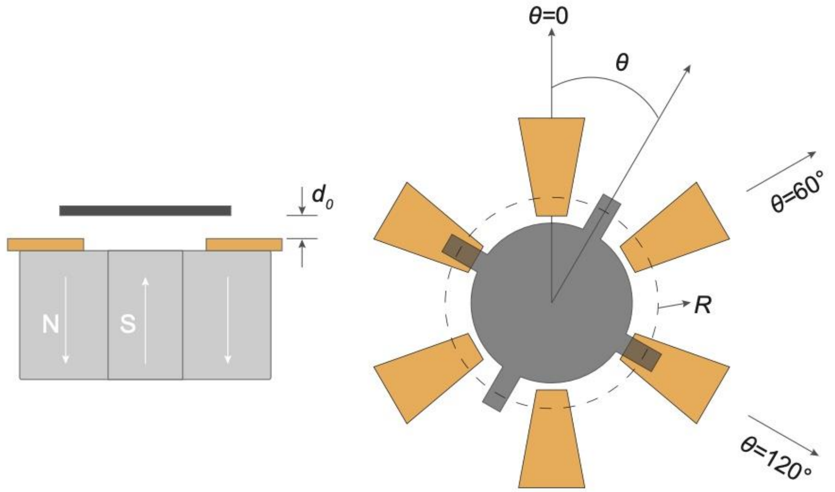


Figure 1: Schematic of diamagnetically levitating rotor.

- 6 rotating arms with 6 fixed electrodes
- Original python script only accounts for torque exerted by 6 electrodes on 2 arms (for simplification first?)
- Torques generated by electrostatic attraction between charged electrodes and rotor arms

Physics of an Electrodes-Driven Rotor System

- Electrodes have oscillating voltages in triangular waveforms at 120 degrees phase difference from each other.
- Coulomb forces generated due to distances between electrodes and charged rotor arms, d_{ij}
- Oscillatory polarities of electrodes generate a rotating electric field that produce a net unidirectional torque for a specific oscillatory frequency f of voltage in each electrode.

Physics of an Electrodes-Driven Rotor System

- Problem with infinite acceleration for any torque at any instant in time.
- Resolved via equation of motion $I\ddot{\theta} + c\dot{\theta} = \tau_{ij}\,,$
- Set theta double dot = 0 to get c(theta dot) = tau.
- 3-phase symmetry of electrodes: Angular velocity corresponds to f_{rotor} = $f_{voltage}$/2. Damping occurs along this set frequency.
- Self-regulating synchronization of rotor adjusts torque to achieve set theta dot at any point in time.
- If $f_{voltage}$ >> f_{rotor}, rotor cannot keep up to achieve the torque created by $f_{voltage}$. Approximate zero acceleration and torque --> system decays.

Implementation with Numpy

$$\tau_{ij} = \epsilon \frac{AV_j^2}{d_{ij}^2} R \frac{\sin\left(\theta_{ai} - \theta_{ej}\right)}{\left|\sin\left(\theta_{ai} - \theta_{ej}\right)\right|},$$

$$V_1 = V_4 = V_0 \left(2 \left|2 \left(ft \bmod 1\right) - 1\right| - 1\right),$$

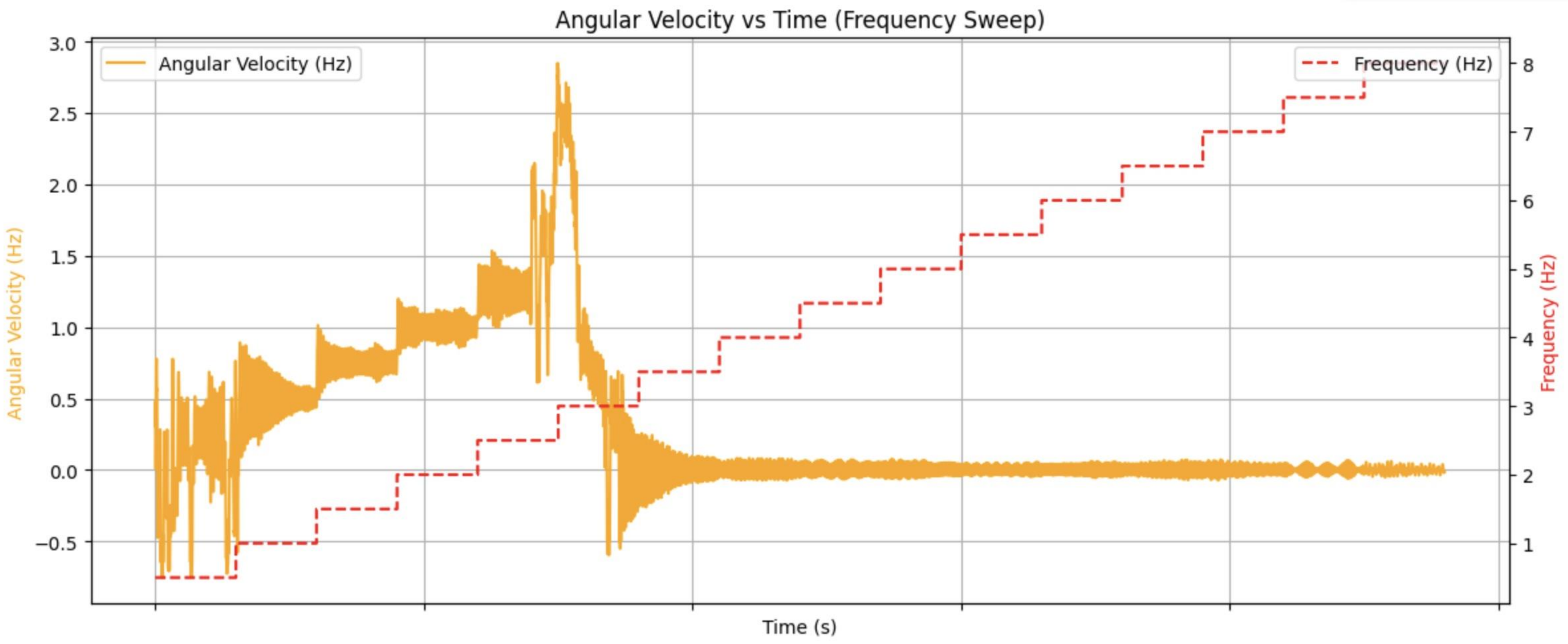
$$V_2 = V_5 = V_0 \left(2 \left|2 \left(\left(ft + \frac{120}{360}\right) \bmod 1\right) - 1\right| - 1\right),$$

$$V_3 = V_6 = V_0 \left(2 \left|2 \left(\left(ft + \frac{240}{360}\right) \bmod 1\right) - 1\right| - 1\right).$$

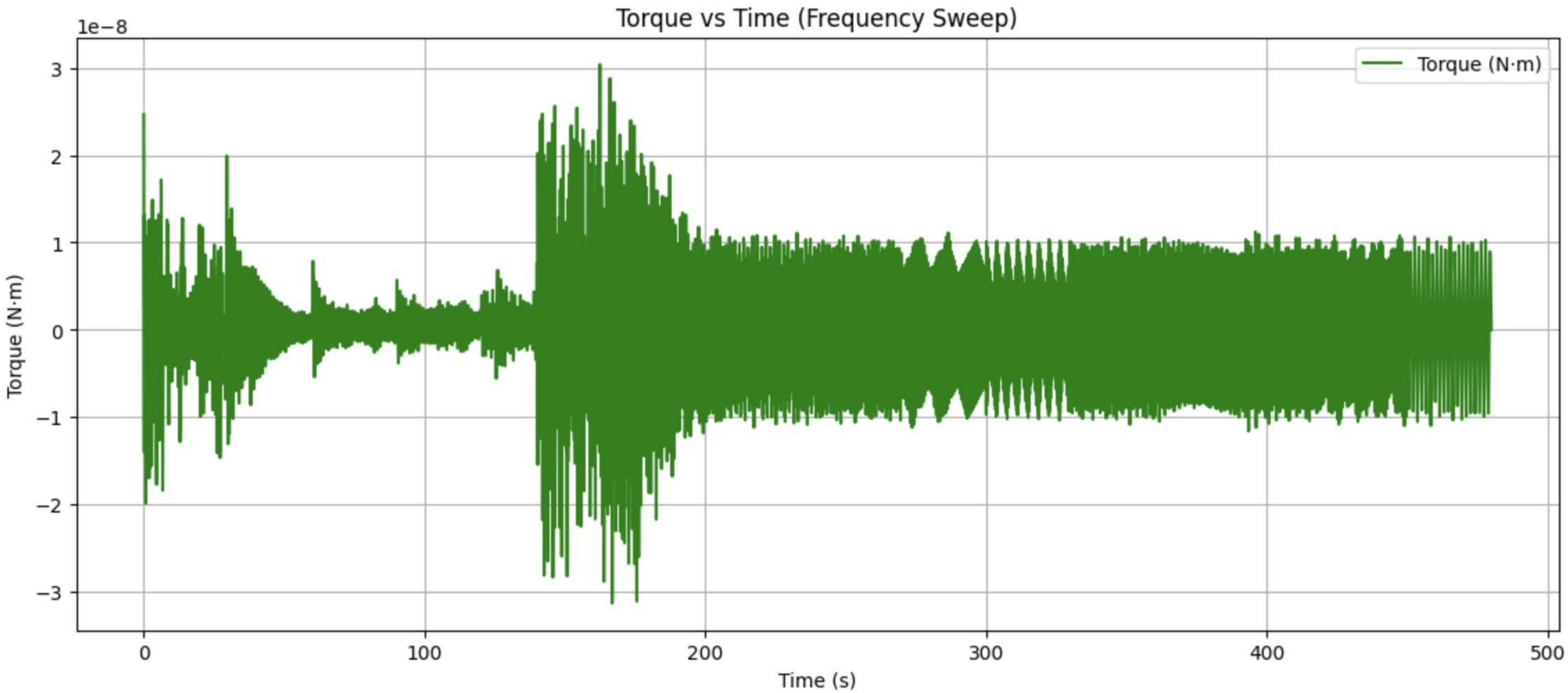
```
# Simulation parameters
t_span = (0, 30)  # Time range (s)
t_eval = np.linspace(t_span[0], t_span[1], 500)
frequencies = np.arange(0.5, 8.5, 0.5)  # Frequencies from 0.5 Hz to 8 Hz

# Initial conditions
theta_0 = 0.1  # Initial angular position (rad)
theta_dot_0 = 0.1  # Initial angular velocity (rad/s)
y0 = [theta_0, theta_dot_0]
# Arrays to store results
all_theta = []
all_theta_dot = []
all_time = []
all_frequency = []
```

Why decay happens during simulation??



Why decay happens during simulation??



Implementation on JAX

```
# Precompute triangular waveforms on a time grid
time_grid = jnp.linspace(0, 1, 1000)
V1_table = V0 * (2 * jnp.abs(2 * (time_grid % 1) - 1) - 1)
V2_table = jnp.roll(V1_table, jnp.floor(len(time_grid) / 3).astype(int)) #JAX has explicit emphasis on data types.
V3_table = jnp.roll(V1_table, jnp.floor(2 * len(time_grid) / 3).astype(int))

@jit
def electrode_voltages_precomputed(t, f):
    # Use jnp.floor to calculate the index as a JAX array and convert to int
    idx = jnp.floor((f * t) % 1 * len(time_grid)).astype(int) #.astype designed for arrays use
    v1 = V1_table[idx]
    v2 = V2_table[idx]
    v3 = V3_table[idx]
    return jnp.array([v1, v2, v3, v1, v2, v3])

# Torque computation
def compute_torques_scalar(theta, t, f):
    angle_diff = theta + arm_angles[:, jnp.newaxis] - electrode_angles[jnp.newaxis, :]
    sin_half_angle_diff = jnp.sin(angle_diff / 2)
    distances_squared = d0_squared + (R2 * sin_half_angle_diff)**2
    voltages_squared = electrode_voltages_precomputed(t, f)**2
    torques = R_epsilon_A * voltages_squared / distances_squared * jnp.sign(jnp.sin(angle_diff))
    return jnp.sum(torques)
```

Literature Review: The APHYNITY Model

- Augmenting incomplete over-simplistic physical dynamics described by differential equations.
- Decomposes dynamics into 2 components: Physical component from prior knowledge of first principles. Data-driven component accounting for errors of physical model.
- APHYNITY uses a Model Based(MB)/Machine Leaning(ML) approach, going beyond pure ML methods.
- Emphasizes parametrized dynamics.
- Supersedes pre-existing models of augmentation by addressing uniqueness of decomposition and increased accuracy via parametrization.

Literature Review: The APHYNITY Model. Principles of Implementation

3. The APHYNITY Model

In the following, we study dynamics driven by an equation of the form:

$$\frac{\mathrm{d}X_t}{\mathrm{d}t} = F(X_t) \tag{1}$$

Xt – state of system at time t.

defined over a finite time interval $[0, T]$, where the state X is either vector-valued, i.e. we have $X_t \in \mathbb{R}^d$ for every t (pendulum equations in Section 4), or X_t is a d-dimensional vector field over a spatial domain $\Omega \subset \mathbb{R}^k$, with $k \in \{2, 3\}$, i.e. $X_t(x) \in \mathbb{R}^d$ for every $(t, x) \in [0, T] \times \Omega$ (reaction-diffusion and wave equations in Section 4). We suppose that we have access to a set of observed trajectories $\mathcal{D} = \{X_\cdot : [0, T] \to \mathcal{A} \mid \forall t \in [0, T], \mathrm{d}X_t/\mathrm{d}t = F(X_t)\}$, where $\mathcal{A}$ is the set of X values (either $\mathbb{R}^d$ or vector field). In our case, the unknown F has $\mathcal{A}$ as domain and we only assume that $F \in \mathcal{F}$, with $(\mathcal{F}, \|\cdot\|)$ a normed vector space.

$$F_p \in \mathcal{F}_p \subset \mathcal{F}.$$

$$F_a \in \mathcal{F}$$

F_a only plays a complementary role. Must be minimized

Literature Review: The APHYNITY Model. Principles of Implementation

$$\min_{F_p \in \mathcal{F}_p, F_a \in \mathcal{F}} \|F_a\| \quad \text{subject to} \quad \forall X \in \mathcal{D}, \forall t, \frac{\mathrm{d}X_t}{\mathrm{d}t} = (F_p + F_a)(X_t)$$

constraint

We need to make sure ||Fa|| is well-defined. Depends on geometry of *prior knowledge* function space. 2 propositions:

1. Function space must be a proximinal set, meaning every point of space has at least one nearest point. Ensures existence of minimum.
2. It is a Chebyshev set (unique nearest points). Ensures uniqueness of minimization. Applies to most finite dimensional subspaces.

Literature Review: The APHYNITY Model. Parameterization

$F_p^{\theta_p}$ and $F_a^{\theta_a}$.

dataset of trajectories discretized with a given temporal resolution Δt: $\mathcal{D}_{\text{train}} = \{(X^{(i)}_{k\Delta t})_{0 \le k \le \lfloor T/\Delta t \rfloor}\}_{1 \le i \le N}$. Solving (2) requires estimating the state derivative $\mathrm{d}X_t/\mathrm{d}t$ appearing in the constraint term. One solution is to approximate this derivative using

Integral-Trajectory Based Approach

based approach: we compute $\widetilde{X}^i_{k\Delta t, X_0}$ from an initial state $X^{(i)}_0$ using the current $F_p^{\theta_p} + F_a^{\theta_a}$ dynamics, then enforce the constraint $\widetilde{X}^i_{k\Delta t, X_0} = X^i_{k\Delta t}$. This leads to our final objective function on (θ_p, θ_a):

simulated state

ground truth

$$\min_{\theta_p, \theta_a} \left\| F_a^{\theta_a} \right\| \quad \text{subject to} \quad \forall i, \forall k, \widetilde{X}^{(i)}_{k\Delta t} = X^{(i)}_{k\Delta t} \tag{3}$$

where $\widetilde{X}^{(i)}_{k\Delta t}$ is the approximate solution of the integral $X^{(i)}_0 + \int_0^{k\Delta t} (F_p^{\theta_p} + F_a^{\theta_a})(X_s)\,\mathrm{d}s$ obtained by a differentiable ODE solver.

t dt

Literature Review: The APHYNITY Model. Adaptive Constrained Optimization

$$\mathcal{L}_{\lambda_j}(\theta_p, \theta_a) = \|F_a^{\theta_a}\| + \lambda_j \cdot \mathcal{L}_{traj}(\theta_p, \theta_a) \qquad (4)$$

where $\mathcal{L}_{traj}(\theta_p, \theta_a) = \sum_{i=1}^{N} \sum_{h=1}^{T/\Delta t} \|X_{h\Delta t}^{(i)} - \widetilde{X}_{h\Delta t}^{(i)}\|$.

Use of Lagrange Multipliers as weights. Higher the weight, forces next iteration to enforce stricter constraint. Smaller the weight, emphasis on more simplified model.

Literature Review: The APHYNITY Model. Adaptive Constrained Optimization

converge to a solution (at least a local one) of the constrained problem (3). We select $(\lambda_j)_j$ by using an iterative strategy: starting from a value λ_0, we iterate, minimizing $\mathcal{L}_{\lambda_j}$ by gradient descent§, then update λ_j with: $\lambda_{j+1} = \lambda_j + \tau_2 \mathcal{L}_{traj}(\theta_{j+1})$, where τ_2 is a chosen hyper-parameter and $\theta = (\theta_p, \theta_a)$. This procedure is summarized in Algorithm 1. This adaptive iterative procedure allows us to obtain stable and robust results, in a reproducible fashion, as shown in the experiments.

idea for this to approach 0.

<u>**Iterative algorithm for parameters theta and Lagrange weights. Theta continuously adjusted until optimal accuracy and minimal Fa obtained.**</u>

Algorithm 1: APHYNITY

Initialization:

$\lambda_0 \geq 0, \tau_1 > 0, \tau_2 > 0$;

for *epoch* $= 1 : N_{epochs}$ **do**

  for *iter in* $1 : N_{iter}$ **do**

    for *batch in* $1 : B$ **do**

      $\theta_{j+1} = \theta_j - \tau_1 \nabla \left[\lambda_j \mathcal{L}_{traj}(\theta_j) + \|F_a\|\right]$

  $\lambda_{j+1} = \lambda_j + \tau_2 \mathcal{L}_{traj}(\theta_{j+1})$

$\nabla \mathcal{L}_{traj} \Rightarrow$ steepest gradient ascent

Possible Implementation to Rotor System

Steps:

1. Simulate idealized physics using only F_p predictions.
2. Train F_a using a neural network model for its parameterization. Optimize the APHYNITY loss (next slide).
3. Adaptation of Lagrange weights. Start small to prioritize simplicity, then increase to enforce trajectory matching.

Possible Implementation to Rotor System

$$F_p(\theta,\dot{\theta}) = \frac{1}{I}\left(\tau_{ij}^{\text{ideal}} - c\dot{\theta}\right),$$

$$\min_{\theta_p,\theta_a} \|F_a^{\theta_a}\| \text{subject to} \forall t, \ddot{\theta}_{\text{obs}}(t) = F_p^{\theta_p}(\theta,\dot{\theta}) + F_a^{\theta_a}(\theta,\dot{\theta}).$$

Trajectory loss ($\mathcal{L}_{traj}$):

$$\mathcal{L}_{traj} = \sum_t \left\| \ddot{\theta}_{\text{obs}}(t) - \left(F_p^{\theta_p} + F_a^{\theta_a}\right) \right\|^2.$$