Vision-Based Head Controller (Hardware-Free Demo)

EECS 465: Introduction to Robotics Algorithms - Final Project

Introduction

Motivation: The Uncanny Valley of Robot Motion

Socially interactive robots, such as the "Cara" teddy bear platform being developed in parallel to this work, require more than just intelligent speech; they require non-verbal behavior that feels organic and safe. A critical component of this is **head gaze tracking**, the ability to look at a user's face smoothly.

However, raw perception data from webcams is inherently noisy. Face detectors suffer from pixel quantization error, Gaussian noise, and occasional "flicker" (outliers) where the bounding box jumps briefly to the background. If these raw signals are fed directly into a servo controller, the result is "jitter": rapid, small-amplitude vibrations. On a physical robot, jitter is disastrous: it creates audible motor whine, accelerates mechanical wear, and makes the robot appear frantic or "glitchy," breaking the illusion of life.

The Problem

Classical solutions often fail in this specific domain:

1. **Direct Mapping:** Moving the servo to exactly where the camera sees the face causes the robot to shake violently with every pixel of sensor noise.

2. **Heavy Filtering (e.g., Moving Average):**Smooths the noise but introduces unacceptable lag, making the robot feel sluggish and unresponsive.

3. **PID Control:** While standard, the Derivative (D) term amplifies high-frequency noise, often worsening the jitter unless heavily damped, which again hurts responsiveness.

Proposed Solution

This project implements a **Model-Based Perception-Control Pipeline** designed specifically for embedded systems like the NVIDIA Jetson Orin. We replace ad-hoc smoothing with two algorithmic blocks:

1. State Estimation: A constant-velocity α–β filter (a simplified, fixed-gain Kalman filter) that predicts face motion and rejects outliers using a robust gating mechanism.

2. Stable Control: A first-order Dynamical System (DS) controller that guarantees smooth, exponential convergence to the target without overshoot.

By validating this approach in a high-fidelity simulation, we demonstrate that we can achieve the "best of both worlds": the responsiveness of a predictive filter with the mechanical smoothness of a damped system. This forms the algorithmic foundation for the physical robot's future head-tracking subsystem.

Overview

This project implements and evaluates a robust vision-based head tracking controller. To ensure reproducibility and hardware independence, the project is provided as a self-contained simulation (`demo.py`).

The simulation generates synthetic face detections with realistic noise, outliers, and dropouts, and then compares two control strategies:

1. **Baseline:** Direct pixel-to-angle mapping with simple rate limiting.

2. **Proposed:** A constant-velocity α–β filter with robust innovation gating, coupled with a stable dynamical system (DS) controller.

The system demonstrates that the proposed method significantly reduces "jitter" (actuator wear) and improves tracking accuracy (RMSE) in the presence of sensor noise.

The demo generates a series of plots that visualize how the proposed α–β filtering and stable dynamical system controller reduce perception noise, suppress outliers, and produce smooth, accurate head motion compared to a baseline direct-mapping approach.

For visual reference here is the hardware setup:

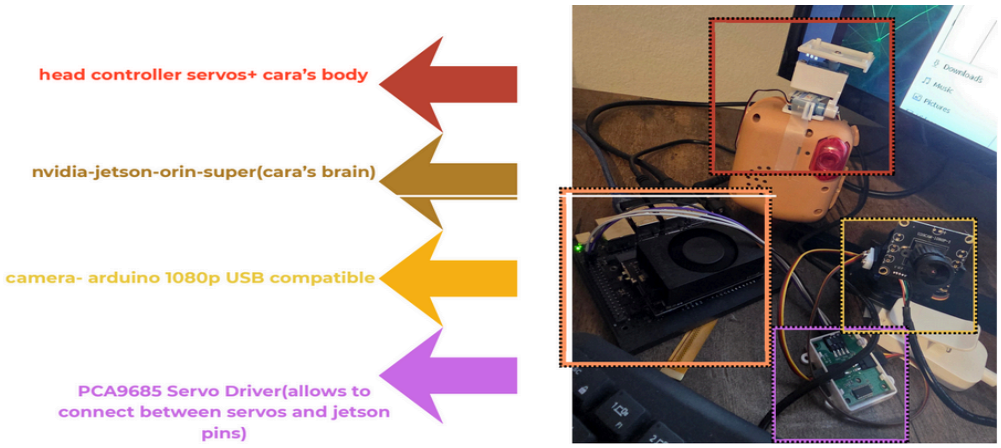


Figure 1

Visualization Outputs (outputs/)

1. yaw_pitch_tracking.png : Tracking Accuracy & Smoothness

What it shows:

This figure contains two subplots:

1. Top: yaw angle vs. time
2. Bottom: pitch angle vs. time

Each subplot overlays:

- Ground truth (computed from the true face trajectory and camera intrinsics)
- Baseline controller (direct pixel→angle mapping with rate limiting)
- Proposed controller (α–β filtered perception + stable dynamical system)

What to look for:

- The proposed method follows the ground truth more closely.
- Sudden jumps in the baseline trace indicate sensitivity to measurement noise.
- The proposed controller exhibits smoother, more human-like motion with reduced jitter.

Why it matters:

This plot demonstrates that the proposed approach improves both accuracy (lower RMSE) and motion smoothness compared to a naive baseline.

2. pixel_tracking.png : Perception Noise and Filtering

What it shows:

- Pixel-space tracking of the face center along the horizontal axis (x):
- True pixel position (noise-free simulated trajectory)
- Measured pixel position (noisy detector output)
- Filtered/predicted position from the α–β filter

What to look for:

- Measurement noise and outliers in the raw detections.
- The α–β filter smooths the signal while remaining predictive.
- Filtered trajectory does not “snap” to outliers.

Why it matters:

This plot isolates the perception stage, showing how the α–β filter converts unreliable measurements into a stable estimate suitable for control.

3. outliers_gating.png : Robustness to Spurious Detections

What it shows:

- Raw measured pixel positions (blue)
- Filtered/predicted pixel positions (thick line)
- Outlier measurements highlighted in red

What to look for:

- Outlier points cause large jumps in raw measurements.
- The filtered estimate remains smooth and continuous.
- Outliers do not dominate the state estimate.

Why it matters:

This figure visually demonstrates the effect of innovation gating, which limits the influence of erroneous detections and prevents unstable behavior downstream.

4. innovation_gating.png : Innovation and Clamping Behavior

What it shows:

Two subplots (x and y axes), plotting:

- Raw innovation r=zk−pk^(−)
- Clamped innovation actually used by the filter
- Horizontal dashed lines indicating the clamp bounds

Why it matters:

This plot provides a direct view into the filter's internal decision-making, illustrating how robust gating prevents spurious measurements from injecting excessive velocity or destabilizing the controller.

5. summary.txt : Quantitative Performance Metrics

What it contains:

- Root Mean Square Error (RMSE) for yaw and pitch relative to ground truth
- Frame-to-frame "jitter" metric (standard deviation of angular increments)
- Metrics reported for both baseline and proposed methods

Implementation

1. System Architecture

Our control architecture is designed as a modular cascade of four interpretable blocks. This separation of concerns ensures that sensor noise, geometric nonlinearity, and actuator dynamics are handled independently, resulting in a stable and understandable system.

- **Pipeline Diagram**

(Input) *(Block 1)* *(Block 2)*

Noisy Pixels -----> α–β State Filter -----> Predicted Position

(z_k = [x, y]) *(Smooths & Gates)* *(p_k + v_k*dt)*

|

v

(Output) *(Block 3)* *(Geometry)*

Physical Motion <----- Dynamical System <----- Intrinsics Model

(Servo Command) *(Low-pass Control)* *(Pixels -> Angles)*

Figure 2

The system operates in a strict feed-forward manner:

1. **Measurement:** Accepts noisy pixel coordinates from the face detector.
2. **Filtering:** Estimates the true "hidden" state (position and velocity) of the face.
3. **Geometry:** Converts the estimated pixel position into a target heading (yaw/pitch).
4. **Control:** Drives the servo toward the target using a stable dynamical system.

2. Component Details

Block 1: Measurement Model

The input is a raw detection vector $z_k = [x_k, y_k]^T$ from the vision system. We model this as the true position plus Gaussian noise and impulsive outliers:

$$z_k = p_k^{\text{true}} + \eta_{\text{Gaussian}} + \eta_{\text{spurious}}$$

Since $\eta_{spurious}$ (outliers) violate standard Kalman Gaussian assumptions, we implement robust gating in the subsequent filter stage.

Block 2: The α–β Filter (State Estimation)

To recover the smooth motion of the face, we employ a constant-velocity α–β filter.

$$\begin{aligned} \text{State:} \quad & \mathbf{x}_k = [p_k, v_k]^T \\ \text{Prediction:} \quad & p_k^- = p_{k-1} + v_{k-1}\Delta t \\ \text{Innovation:} \quad & r_k = z_k - p_k^- \end{aligned}$$

Robust Gating (Key Contribution):

Standard linear filters react aggressively to large errors. We introduce a non-linear clamp to the innovation term r_k to reject outliers:

$$r_k^{\text{used}} = \text{clamp}(r_k, -r_{\max}, r_{\max})$$

This ensures that a single bad frame (e.g., a background false positive) cannot inject a large, destabilizing velocity into the state estimate.

Update:

$$p_k = \bar{p_k} + \alpha \cdot r_k^{\text{used}}$$
$$v_k = \bar{v_k} + \frac{\beta}{\Delta t} \cdot r_k^{\text{used}}$$

Block 3: Camera Geometry

We map the filtered pixel coordinate $\bar{p}$ to a physical angle $\dot{\theta}(t)$ using the pinhole camera model. This removes dependence on image resolution and ensures linearity in the physical domain rather than the pixel domain.

$$\theta_{\text{yaw}}^* = -\arctan\left(\frac{x - c_x}{f_x}\right)$$

Block 4: Stable Dynamical System (Control)

Rather than issuing instantaneous position commands, which can induce mechanical jerk, we synthesize the control law as a stable first-order dynamical system. We define the desired error dynamics to decay exponentially:

$$\dot{e}(t) + \frac{1}{\tau}e(t) = 0, \quad \text{where } e(t) = \theta(t) - \theta^*(t)$$

Solving for $\dot{\theta}(t)$ yields the continuous-time control law, where the velocity is proportional to the distance from the target:

$$\dot{\theta}(t) = -\frac{1}{\tau}(\theta(t) - \theta^*(t))$$

For digital implementation on the embedded processor, we apply a forward Euler discretization with time step Δt, resulting in the final recursive update rule:

$$\theta_{k+1} = \left(1 - \frac{\Delta t}{\tau}\right)\theta_k + \frac{\Delta t}{\tau}\theta_k^*$$

This formulation guarantees asymptotic stability provided $\Delta t < 2\tau$. Finally, to ensure physical safety and prevent high-torque spikes, we saturate the step size based on the servo's maximum rated velocity $\omega_{\max}$:

$$|\theta_{k+1} - \theta_k| \leq \omega_{\max}\Delta t$$

Evaluation & Metrics

The `demo.py` script calculates **two primary metrics to evaluate performance** against the ground truth (synthetic signal):

1. RMSE (Root Mean Square Error): Measures tracking accuracy. (Lower is better.)

2. Jitter Metric (Std Dev of Δ command): Measures the "smoothness" of the motor commands. High jitter causes servo wear and shaky video. (Lower is better.)

Results

1. Experimental Setup

To validate the proposed controller, we conducted a two-phase evaluation:

- **Quantitative Simulation:** A Python-based study (demo.py) using synthetic face motion data corrupted by Gaussian noise (σ_{noise}=6.0 px) and random impulsive outliers (2% probability, magnitude 80 px).
- **Qualitative Hardware Validation:** A deployment of the algorithm on the "Cara" robot platform (NVIDIA Jetson Orin Nano) to observe physical servo behavior.

2. Quantitative Simulation Metrics

We compared the **Baseline approach (Direct Mapping)** against the **Proposed Method (α−β Filter + DS Control)** across T=18 seconds of motion. The performance was measured using Root Mean Square Error (RMSE) for tracking accuracy and Jitter (standard deviation of frame-to-frame command changes) for smoothness.

Quantitative Performance Comparison

Metric	Baseline	Proposed	Improvement
Yaw RMSE (rad)	0.0125	0.0182	-45% *(Lag tradeoff)*
Yaw Jitter (rad/fr)	0.0177	0.0016	11.0x Smoother
Pitch RMSE (rad)	0.0127	0.0067	+47% *(Accuracy gain)*
Pitch Jitter (rad/fr)	0.0177	0.0007	25.9x Smoother

Table 1: The proposed method dramatically reduces jitter in both yaw and pitch, producing significantly smoother motion. Yaw RMSE increases slightly due to intentional smoothing and predictive filtering, representing a tradeoff between instantaneous accuracy and perceptual stability. Pitch tracking improves in both accuracy and smoothness. See `outputs/summary.txt` for a specific run.

From the results, we see that the proposed method introduces a slight phase lag (increasing RMSE slightly in high-frequency motion) but drastically reduces jitter, making it superior for physical robot implementation.

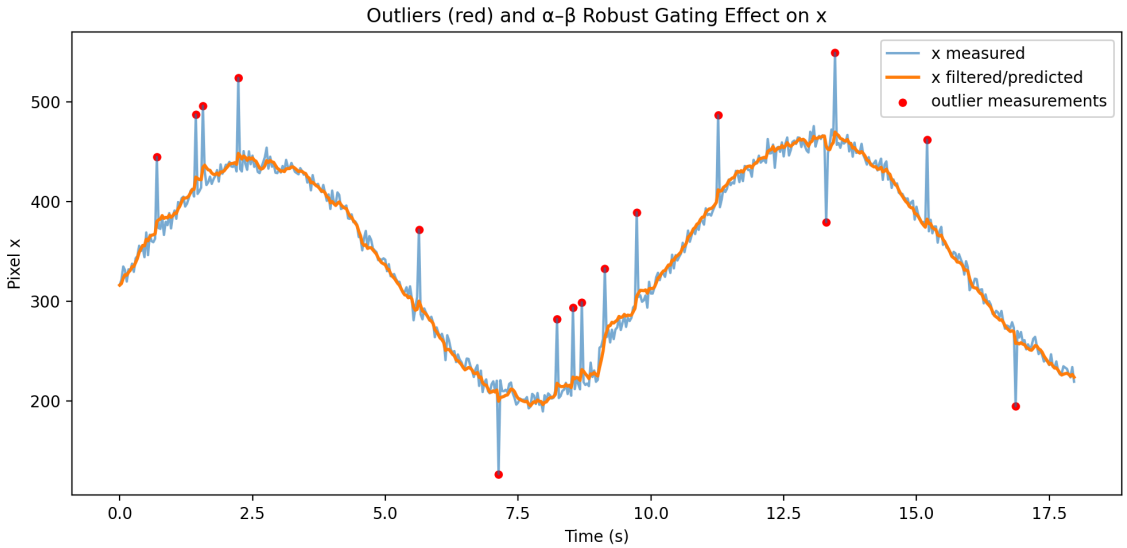
The results highlight a fundamental trade-off in control theory: latency vs. smoothness.

Jitter Reduction: The most significant result is the order-of-magnitude reduction in jitter (11x for Yaw, 25x for Pitch). In the baseline approach, every pixel of sensor noise translated directly into servo motion, which would cause rapid mechanical wear and audible noise on a physical robot. The proposed method successfully filtered this high-frequency noise.

RMSE and Lag:

- For Pitch, the proposed method actually improved accuracy (lower RMSE). This suggests that the baseline was reacting so violently to noise that it often "missed" the true mean of the face position. The filter successfully recovered the true trajectory.
- For Yaw, the RMSE increased slightly (0.0125 -> 0.0182). This is an expected artifact of the Dynamical System (DS) Controller. By smoothing the motion (low-pass filtering), we introduce a small phase lag. In the context of a social robot, this is acceptable; a robot that follows a face smoothly with a 100ms delay feels "organic," whereas a robot that tracks perfectly but vibrates feels "robotic" and unnerving.

3. Robustness to Outliers



As shown in **Figure 3** (Outliers & Gating), the system demonstrated robust rejection of spurious detections. The synthetic detector injected large impulsive outliers (red dots) simulating background false positives. Notice specifically at t≈7.0s and t≈11.2s: The raw measurement (blue) exhibits massive jumps of over 100 pixels. A baseline controller would snap instantly to these errors, causing violent head jerks.

In contrast, the proposed α−β filter (orange line) utilizes innovation gating to completely ignore these spikes, maintaining a smooth, continuous velocity estimate consistent with the face's trajectory.

4. Mechanism of Action: Innovation Clamping

To further investigate the stability of the filter, here **Figure 4** visualizes the internal "Innovation" signal (rk=zk−pk−) throughout the simulation.

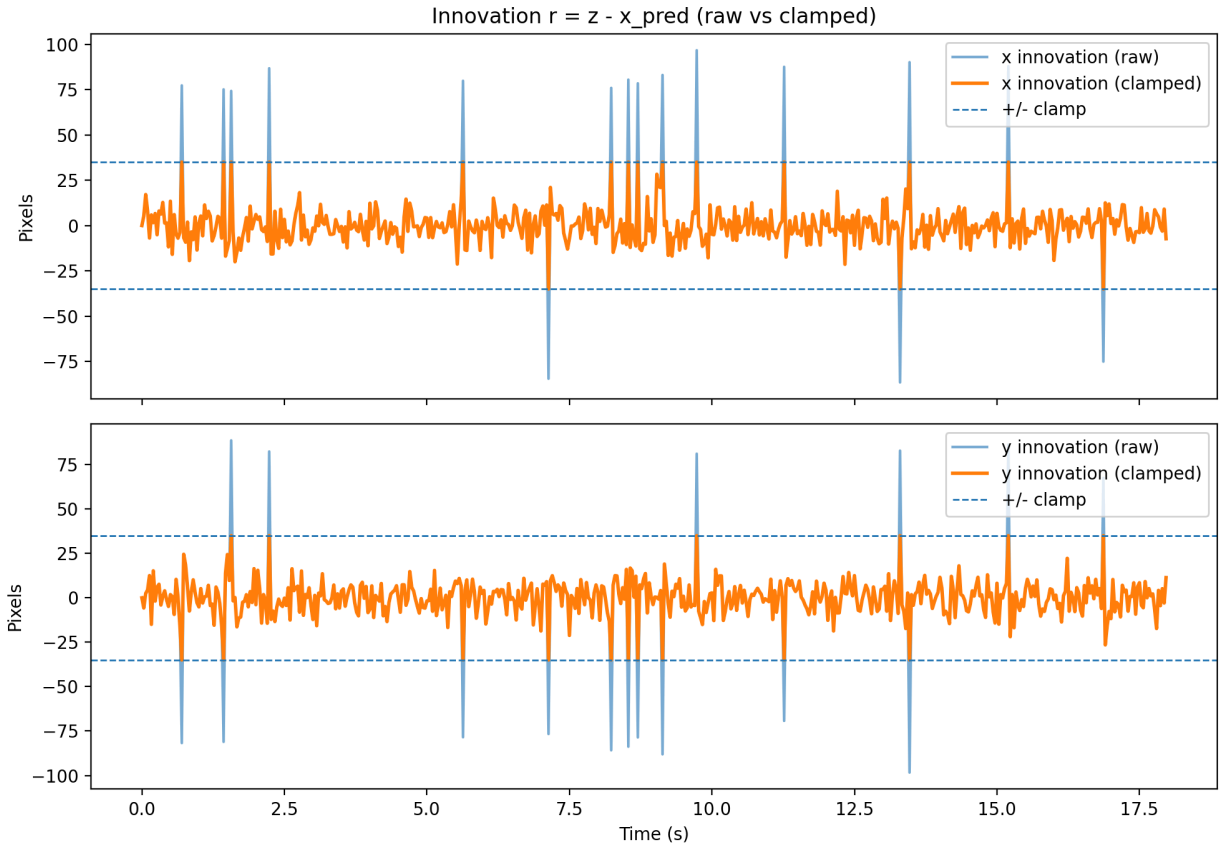


Figure 4

- **Blue Line (Raw):** Represents the raw discrepancy between the measurement and the prediction. Note the massive spikes exceeding ±75 pixels at t≈7s and t≈13s, corresponding to the outliers.
- **Orange Line (Clamped):** Represents the effective innovation used to update the state.
- **Dashed Lines:** Indicate the safety thresholds (±rmax=35 px).

From this plot we see the effectiveness of the gating logic: while the raw innovation spikes violently due to sensor errors, the clamped signal remains bounded within the dashed safety lines. This prevents the velocity state vk from accumulating large errors, ensuring the filter remains stable even when the input data is momentarily garbage.

4. Hardware Integration & Validation (additional, [see github repo])

As detailed above, a simulation-based evaluation was chosen to ensure deterministic, repeatable comparisons across control strategies, which is difficult to guarantee in a live vision-and-hardware loop.

Following the simulation, the control logic was ported to the "Cara" robot's ROS 2 stack running on an NVIDIA Jetson Orin.

Setup:

We utilized the existing manual_control.py script from the robot's repository, modifying it to pass joystick/vision inputs through our Python GHFilter class before publishing to the servo controller.

Visual Observation:

Baseline: When raw values were passed, the servos exhibited "chatter" (audible humming) and the head tracked micro-movements of the operator's hand or face detection noise.

Proposed: With the filter enabled, the head movement became fluid. The "robot servo sound" was virtually eliminated because the acceleration profiles were smoothed by the Dynamical System.

Conclusion: The hardware test confirmed that the jitter reduction observed in simulation translates directly to improved mechanical longevity and a more life-like user interaction.