The Human Control Protocol v0.4-131125 Technical paper

Thordur Arnason, Lena Thorsmæhlum
Gervi Labs, 2025

Abstract

The Human Control Protocol (HCP) is a real-time performance system that casts an AI agent as director and audience members as actuators. A WebSocket orchestrator links an autonomous agent to mobile clients. Over a n-minute session, the agent runs a perception, reflection, decision, and action loop, targeting instructions to audience percentages and adapting to completion and latency. The work yields an ephemeral performance and a generated visual artifact.

1. System overview

- **Goal:** produce a site-specific, unrepeatable performance by directing simple collective actions.
- **Principle:** constrained autonomy for the agent, explicit safety limits for participants, and clear consent.
- **Outputs:** live performance, JSON logs, Markdown narrative, and a generated abstract image.

2. Architecture

- **Orchestrator:** WebSocket server for bidirectional low-latency messaging. Tracks sessions, clients, heartbeats.
- **Agent:** LLM-driven controller with a compact policy prompt and rolling memory. Produces observations, strategy, tone, and actions per round.
- **Tools (clients):** mobile web app. Receives instructions, displays them plainly, and collects button-press confirmations and micro-timers.
- **Log system**: Three logs are created as Fossils. A human readable MD, a JSON reflecting the first and a JSON replay log capturing all client-server data during the performance.

Messaging

- `client_join`, `client_leave`, `heartbeat`
- `round_start`, `instruction_broadcast`, `instruction_ack`
- `client_action_complete`, `metrics_update`, `round_end`, `session_end`

3. Runtime loop and state machine

States: `AWAITING_START`, `CALIBRATING`, `PERFORMING`, `REFLECTING`, `COMPLETED`.

```
None
state := AWAITING_START
while state != COMPLETED:
  if state == AWAITING_START:
    wait until min_clients and operator_start
    init memory; state := CALIBRATING

  if state == CALIBRATING:
    send simple action to 100%
    collect completion, latency
    set baselines; state := PERFORMING

  if state == PERFORMING:
    for round in 1..7:
      s := sense(current_clients, prev_metrics, memory)
      plan := agent.think(s)   // observation, strategy, tone, actions[]
      actions := validate(plan.actions, constraints)
      broadcast(actions)
      m := collect_metrics()
      update(memory, plan, m)
    state := REFLECTING

  if state == REFLECTING:
    agent.write_self_eval(memory, metrics)
    persist logs and artifact
    state := COMPLETED
```

4. Agent policy

Per round the agent must output:

- `observation`
- `strategy`
- `narrative_thread`
- `reflection`
- `sentiment` one word
- `actions[]` two or three entries targeted by percentage

Guardrails:

- Safe, one-hand actions. No walking. No touching others. No invasive gestures.
- Clear, short sentences. Max 12 words per instruction.
- Percent targets only. Never select individuals.

5. Instruction format

```json
{
  "id": "r3-a1",
  "target_percent": 25,
  "text": "Stand and stretch your arms up, then sit.",
  "duration_s": 8,
  "count_in_s": 2,
  "visibility": "whole-room",
  "safety": ["one-hand-free", "no-walking"]
}
```

6. Client behavior

- Plain UI with large text, progress line, and a single confirm button.
- Accessibility: readable fonts, high contrast, no strobing, no audio cues required.
- Telemetry: `displayed_at`, `confirmed_at`, `skipped`, `device_latency_ms`.

7. Metrics

- **Completion rate:** share of clients pressing confirm within window.
- **Latency:** median and spread from display to confirm.
- **Engagement index:** rolling measure mixing join stability, completion, and latency.
- **Safety signals:** skip spikes or abnormal delays trigger simpler next actions and longer count-ins.

8. Narrative memory

Rolling structure carrying:

- last three `sentiment` values
- short `beats` summary per round
- notes on over- or under-reach
- constraints that were tightened or relaxed

9. Performance protocol

- Duration about 3-15 minutes.
- Calibration: one action to 100 percent.
- 5-50 rounds. Two or three actions per round.
- Count-in for synchronization when needed.

10. Data and persistence

- **JSON log:** full telemetry and agent outputs.
- **Markdown report:** readable timeline with observations, strategy, and outcomes.
- **Privacy:** no names, no device IDs beyond ephemeral session keys, no media capture.

Sample log skeleton:

```JSON
{
  "session_id": "hcp-2025-11-12-1900",
  "site": "venue-name",
  "clients_peak": 152,
  "calibration": {"completion": 0.81, "median_latency_ms": 980},
  "rounds": [
    {
      "round": 1,
      "observation": "eager, fast responses",
      "sentiment": "curious",
      "actions": [],
      "metrics": {"completion": 0.77, "median_latency_ms": 860}
    }
  ],
  "self_eval": "what worked, what failed, what to try next"
}
```

11. Generated artifact

- Input features: per-round completion, latency histogram, action mix, tone.
- Suggested rendering: parameterized field lines or radial timelines where thickness maps to completion and spacing maps to latency. Colors optional or venue-defined. One PNG per session.

12. Safety and consent

- Entry screen explains rules and consent.

- All actions reversible and low effort.
- Group targeting only.
- Immediate opt-out button.

13. Failure modes and mitigation

- **Network loss:** client caches next action and shows last stable state.
- **Agent overreach:** guardrail validator simplifies text and caps target_percent.
- **Low engagement:** switch to single simple action and longer rest.

14. Configuration

- Session parameters: rounds, max action length, min clients, count-in seconds.
- Venue profile: seating map optional, light level note, estimated aisle space.
- Model settings: temperature range, max tokens per reasoning step, memory window.

15. High level architecture