Robotic Keyboard Actuation: Hardware-in-the-Loop Input for Safety-Critical Systems

***Abstract*— This paper presents a lightweight robotic keyboard actuation system built with two servo motors and a Raspberry Pi to enable authentic physical interaction with safety-critical devices. Unlike software automation, which is infeasible in air-gapped or security-sensitive environments, our hardware-based approach provides reliable, high-speed input suitable for domains such as nuclear or biohazard facilities, medical and military systems, accessibility interfaces for motor-impaired users, and keyboard durability testing. We demonstrate that the system satisfies time-sensitive control requirements and effectively distinguishes human from robotic input traces in MountainCar control tasks. This work provides a foundation for safe and practical keyboard interaction in scenarios where simulations are difficult to construct and software-based methods are either prohibited or inadequate.**

***Index Terms*— Safety-Critical Systems, Assistive Technology, Physical Automation, Human-Robot Interaction, Sim-to-Real Gap**

I. Introduction

Physical keyboard interaction remains indispensable in safety-critical domains where software automation is infeasible or prohibited. In air-gapped hazardous environments (e.g., nuclear facilities or biohazard containment), human operators risk exposure when manually inputting codes on isolated systems [1], [2]. Similarly, security protocols on military equipment and medical devices often prohibit digital interfaces to prevent remote exploits [3], [4]. A critical motivation also emerges in recovery scenarios: hardware input becomes essential when systems become unresponsive and require physical reboot sequences such as Ctrl+Alt+Del, which cannot be triggered via software [5].

Beyond enabling input itself, an equally important challenge is verifying the *source* of the input. In safety-critical contexts, it is crucial to determine whether a key sequence originated from a human operator or from an automated actuator. Unauthorized robotic penetration attempts, spoofed commands, or misattributed assistive inputs could compromise both safety and accountability. Distinguishing human input is vital not only to prevent unauthorized external access but also to ensure that robotic systems themselves cannot bypass safeguards by impersonating legitimate human operators. However, distinguishing between human and robotic input is non-trivial: both produce the same discrete keypress events at the software level. The difference lies only in subtle temporal and force-based patterns.

Prior research on robotic keyboard interaction has largely focused on assistive technologies for motor-impaired users [6]–[8], industrial durability testing [9], or learning-based typing and piano performance [10]–[12]. These works established the feasibility of robotic keypress actuation but did not address the problem of *input provenance*—that is, attributing whether a sequence of keystrokes came from a person or a robot.

To bridge this gap, we present a minimalist robotic actuation platform augmented with formal methods for input source discrimination. The hardware system leverages servo motors and a Raspberry Pi to provide a cost-effective and modular solution, while the classification framework employs random forests and temporal features to distinguish between human and robotic inputs. By extracting temporal patterns, the approach achieves interpretable, high-accuracy classification in classic reinforcement learning benchmarks such as MountainCar, where precise action timing is essential.

Contributions. This paper makes the following contributions:

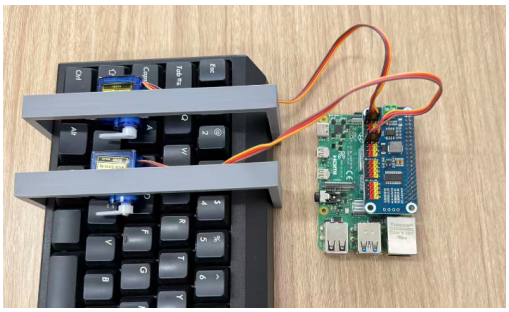
- A lightweight, reconfigurable robotic keyboard actuation system designed for keyboard interaction and could extend to single direction pressing buttons in safety-critical environments.
- A temporal feacture based classification pipeline for distinguishing human vs. robotic input traces, enabling interpretable classification.
- Experimental validation that exposes the challenges of sim-to-real transfer under physical actuation.

Together, these contributions establish both the practicality of portable robotic actuation and the necessity of formal input verification for resilient human–robot interaction in safety-critical systems.

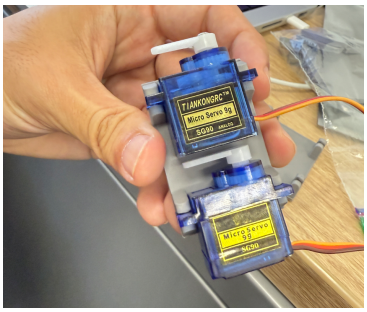
II. Related Work

Research on robotic interaction with keyboards spans multiple domains, reflecting diverse motivations across assistive technology, security testing, industrial validation, and creative or learning applications. Early efforts focused primarily on software-level input emulation or anthropomorphic robotic manipulators, which offered proof of feasibility but often lacked portability, cost-effectiveness, or attention to the physical dynamics of actuation. More recent studies have introduced learning-based methods, modular open-source hardware, and sim-to-real transfer benchmarks, underscoring the need for systems that are reliable in safety-critical environments while remaining adaptable to broader use cases.

In the domain of accessibility, robotic keyboard interaction has often been pursued as a means to assist users with motor impairments. Previous work emphasized either virtual input emulation [13] or the use of robotic manipulators

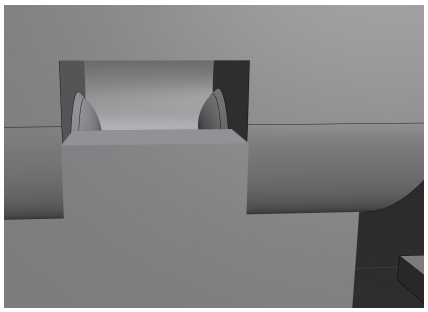


(a) Final design. The motor is mechanically gripped by 3D-printed frames, which are attached to the keyboard and can be repositioned depending on the desired key to trigger. The motor is controlled through a Raspberry Pi connection.

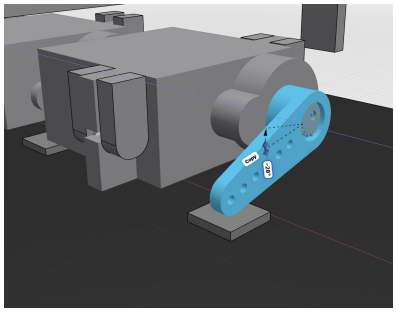


(b) Intermediate design. The motors are held by 3D-printed frames with a fixed pre-calculated spacing. This design works for a specific keyboard but offers limited transferability when the key spacing differs significantly.

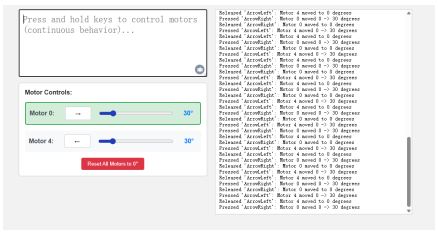
Fig. 1: System overview.



(a) 3D model of the gripping component. Since the design must firmly hold the motor, the shape of the grip is critical; if it is too thin, it may break. We iterated through two design versions to improve durability and reduce fragility.



(b) The servo motor is open-sourced, and its 3D model already exists. This allowed us to design and print the gripping part around the motor's known structure precisely.



(c) We created a web interface that allows motor control not only through direct web requests but also via the interface itself. We observed that when typing 'AD' repeatedly through this interface, the keyboard responded almost instantaneously, with negligible delay, effectively mimicking real-time key triggering.

Fig. 2: Overview of the 3D modeling and web interface.

that allow users to press keys via teleoperation [8], [14]. These assistive systems frequently neglected the mechanical force dynamics inherent in physical keypresses, focusing instead on abstracted or high-level control. Although some commercial solutions, such as wheelchair-mounted robotic arms, demonstrated practical benefits, they remained costly and slow. More recent studies have explored shared autonomy and human-in-the-loop control strategies [15], [16], illustrating the potential of robotic typing to promote independence for motor-impaired individuals. The study in [17] investigated robotic hand typing; while its typing speed lagged behind human performance, there is potential for further improvement. Our work extends these ideas by focusing on tasks that demand fast and precise actuation while remaining low cost. We prioritize simple control logic with minimal manipulation and computational complexity, ensuring a portable and efficient design.

Beyond accessibility, robotic keyboard interaction has been investigated for security and industrial purposes. Physical automation has been used in penetration testing and resilience evaluation of secure, air-gapped systems [18], while large-scale testing rigs have supported durability studies such as switch hysteresis measurement and fatigue analysis [9]. These approaches, however, tend to be bulky, expensive, and limited to laboratory settings. By comparison, the system we propose provides a compact and ruggedized solution, enabling deployment in hazardous or resource-constrained environments where portability and cost-effectiveness are critical.

Another important branch of related work comes from music and learning-based robotic interaction with keyboards. Early humanoid musician projects, such as WABOT-2 in the 1980s, showcased robotic piano playing with pre-programmed trajectories [19], and subsequent work introduced anthropomorphic controllers for piano robots [20]. While these systems achieved precise but scripted performances, advances in machine learning have enabled new forms of robotic typing. Qian *et al.* trained robotic hands via imitation learning from Internet piano videos [10], and the RoboPianist benchmark demonstrated that reinforcement learning can train virtual bimanual agents to play diverse repertoires [11]. Similarly, Baltes *et al.* achieved over 90% typing accuracy on a physical QWERTY keyboard using hierarchical reinforcement learning [12]. While these works highlight the potential of machine learning for keypress tasks, many remain confined to simulation or assume fixed

layouts, which limits their generalizability. By contrast, our approach emphasizes real-world deployment, focusing on safety-critical contexts while preserving adaptability across domains.

Notably absent in prior work is the problem of *input provenance*—determining whether a key sequence originates from a human or from a robotic actuator. In safety-critical contexts, this distinction is essential for accountability and security. Formal methods such as Signal Temporal Logic (STL) have been widely applied for specifying and monitoring temporal properties in cyber-physical systems [21], [22], but their use for distinguishing human and robotic input remains unexplored. Our work addresses this gap by leveraging a random forest classifier with temporal features to uncover distinctive temporal signatures that differentiate human from robotic input, enabling interpretable and highly accurate classification. Although classification tasks pose inherent challenges for STL specification mining [23], [24], we leverage STL specifically for its explainability benefits. Our approach uncovers human-interpretable temporal signatures that distinguish input sources, achieving high accuracy while providing transparent decision criteria.

Taken together, the literature reveals strong interest in robotic keyboard interaction but also exposes persistent gaps. Assistive systems often prioritize abstraction over real-world usability, industrial rigs sacrifice portability for precision, learning-based methods frequently remain confined to simulation, and distinguishing between human and robotic input has not been explored. Our contribution lies in bridging these domains through a lightweight, reconfigurable hardware platform that emphasizes low latency and portability, particularly in scenarios where constructing a digital twin is challenging. When combined with STL-based classification, this approach highlights a pathway toward applying robotic keyboard actuation in safety-critical contexts.

III. Preliminary

A. Signal Temporal Logic

Signal Temporal Logic (STL) [22] is a widely used formalism for expressing safety specifications in many CPS applications. STL formulas are defined over signal predicates of the form $f(\xi) \geq c$ or $f(\xi) \leq c$, where ξ is a signal and $f : \mathbb{R}^n \to \mathbb{R}$ is a real-valued function with $c \in \mathbb{R}$. STL formulas are constructed using the grammar shown in (1), where the time interval $I = [a, b]$ satisfies $a, b \in \mathbb{R}^{\geq 0}$ with $a \leq b$, and $\sim \in \{\leq, \geq\}$.

$$\varphi, \psi ::= \mathit{true} \mid f(\xi) \sim c \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathcal{F}_I\varphi \mid \mathcal{G}_I\varphi \mid \varphi \,\mathcal{U}_I\, \psi \tag{1}$$

Here, $\mathcal{F}$ ("eventually"), $\mathcal{G}$ ("always"), and $\mathcal{U}$ ("until") are temporal operators. Given $t \in \mathbb{R}^{\geq 0}$ and $I = [a, b]$, we use $t + I$ to denote the shifted interval $[t + a, t + b]$. For a signal ξ and a time t, we write $(\xi, t) \models \varphi$ to denote that ξ satisfies φ at time t, and $\xi \models \varphi$ as shorthand for $(\xi, 0) \models \varphi$.

a) Boolean Semantics.: The Boolean semantics of STL formulas are defined recursively. For an atomic predicate $f(\xi) \sim c$, we have $(\xi, t) \models f(\xi) \sim c$ iff $f(\xi(t)) \sim c$. Boolean operators follow standard logic: negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$). Temporal operators are interpreted as:

- $(\xi, t) \models \mathcal{F}_I\varphi$ iff $\exists t' \in t + I$ such that $(\xi, t') \models \varphi$.
- $(\xi, t) \models \mathcal{G}_I\varphi$ iff $\forall t' \in t + I$, $(\xi, t') \models \varphi$.
- $(\xi, t) \models \varphi \,\mathcal{U}_I\, \psi$ iff $\exists t' \in t + I$ such that $(\xi, t') \models \psi$ and $\forall t'' \in [t, t']$, $(\xi, t'') \models \varphi$.

IV. System Design

The proposed system consists of five main components: (1) the target enviroment (2) the controller for operating inside the environmentm, (3) the servo motor controller, (4) the servo motor and (5) the keyboard. Figure 3 illustrates the overall system architecture.

A. Hardware Overview

The physical actuation system (Fig. 1) is built around two SG90 servo motors, each mounted on its own 3D-printed bracket and enclosure, and controlled by a Raspberry Pi 4 Model B via Pulse-Width Modulation (PWM) signals. The Raspberry Pi handles low-level motor commands through a lightweight Python controller, which parses ASCII inputs and generates the corresponding PWM signals. The choice of a Raspberry Pi enables low-cost integration, software flexibility, and easy networking with higher-level policies, while maintaining direct GPIO-level control for real-time actuation. Unlike earlier prototypes, where both servos were co-located in a single rigid 3D-printed block, we redesigned the system so that each actuator is handled separately. We also developed a web interface that enables direct interaction with the servo motor for debugging and testing, as shown in Fig. 2. This modular approach has three main advantages: (i) modules can be rearranged to match different keyboard layouts or distant key combinations (e.g., Ctrl+Alt+Del), (ii) brackets can be quickly reprinted to match keyboards of different heights and profiles, and (iii) only minimal software adjustments are required (servo angle calibration), without rewriting control logic.

Magnets vs. Servo Motors. In early design iterations, we considered using electromagnets to actuate keys. While attractive for their simplicity and lack of mechanical linkages, magnets introduce several drawbacks: high continuous current draw, heat generation, noisy power requirements, and difficulty in ensuring precise force release. They also pose interference risks in sensitive environments. Servo motors, by contrast, offer precise angular control, operate at standard low voltages (5V), and consume current mainly during actuation. Combined with torsion springs, servos provide reliable key release without the need for continuous energy input. For these reasons, we adopted servos as the actuation mechanism.

3D-Printed Modularity. All structural parts (servo mounts, lever arms, enclosures) are fabricated using a 3D printer. Inspired by recent work on customizable open-source

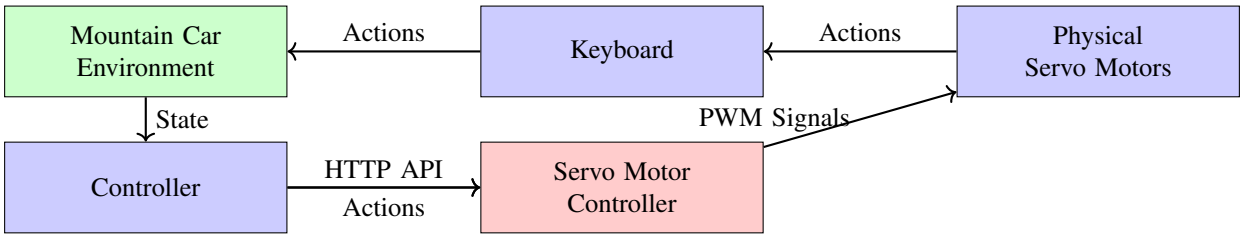


Fig. 3: The system architecture consists of five main components and their interactions. The simulator evaluates the consequences of actions, while the controller determines the intended action and transmits it to the servo motor controller. The servo motor controller identifies the appropriate motor, issues the corresponding command, and the motor executes the command by triggering the keyboard to perform the action in the environment. The environment then receives the action from the agent and transitions to the next state.

robots, such as the Berkeley Humanoid Lite [25], modularity and reconfigurability are emphasized in our design. By decoupling each actuator into its own reprintable module, our design follows the same principle of repairability and rapid customization: broken or worn parts can be cheaply reprinted, while geometry can be adjusted to fit new hardware configurations without modifying electronics or software.

1) Rule-Based Policy: The rule-based policy implements a simple yet effective strategy for the Mountain Car problem:

$$a_t = \begin{cases} -1.0 & \text{if } p_t < -0.4 \\ 1.0 & \text{otherwise} \end{cases} \tag{2}$$

where a_t is the action at time t and p_t is the current position. This policy implements a two-phase approach: first backing up to build momentum, then rushing forward to reach the goal.

B. Servo Motor Controller

The servo motor controller manages the physical hardware through a Flask-based web interface. The controller implements continuous motor pressing behavior, where motors remain in their activated state until a different action is required.

1) Continuous Motor Control: The continuous motor control system maintains motor state across time steps, reducing unnecessary motor movements and improving system efficiency. Figure 3 illustrates the control flow logic.

2) Motor Mapping: The system uses a specific motor mapping configuration:

TABLE I: Motor Configuration and Key Mapping

Action	Key	Motor ID	Direction
-1.0	'z'	0	Left
0.0	None	None	Neutral
1.0	'c'	4	Right

C. Communication Interface

The communication interface implements a robust HTTP-based protocol for real-time communication between the environment controller and the servo motor controller.

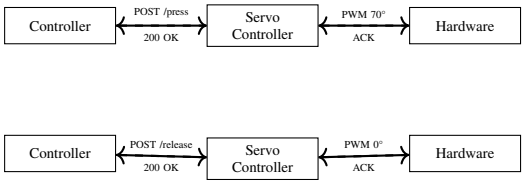


Fig. 4: Communication sequence diagram showing key press and release operations.

1) Network Protocol: The system uses RESTful API endpoints for motor control. Figure 3 shows the communication sequence between components.

The system uses the following RESTful API endpoints:

- `POST /press`: Activates a motor to a specified angle
- `POST /release`: Deactivates a motor (returns to 0°)
- `POST /reset`: Resets all motors to neutral position
- `POST /input`: Handles continuous input with state tracking

2) Error Handling and Retry Logic: The communication interface implements a robust retry mechanism to handle network failures:

Input: Motor command, Max retries = 3 attempt = 1 to Max retries Send HTTP request to servo controller Response status == 200 Return success ConnectionError Wait 1 second Continue to next attempt Return failure

V. Implementation Details

A. State Management

The system maintains comprehensive state information to ensure reliable operation. It keeps track of the current motor state to determine which motor is active, identifies the specific motor being controlled through a unique Motor ID, and uses a Servo Key to map actions to corresponding keyboard inputs. In addition, it monitors a Motor Pressed Flag, which indicates whether a motor is currently activated, ensuring precise and consistent operation.

B. Continuous Behavior Implementation

The continuous motor pressing behavior is implemented through state-aware control logic. When the same action is required consecutively, the system maintains the current motor state without sending additional commands, reducing network traffic and motor wear.

C. Timeout and Safety Mechanisms

The system incorporates several safety mechanisms to ensure robust and reliable operation. An episode timeout is enforced to prevent infinite episodes by capping the maximum number of steps. Connection monitoring continuously checks the servo controller's connectivity, allowing for immediate detection of communication failures. Additionally, a motor reset is automatically triggered upon episode completion or in the event of an error, ensuring that all motors return to a safe and consistent state.

D. Control Logic

Our robotic keyboard control system employs two motors to physically actuate keys, operating under a policy (e.g., Mountain Car) that assumes continuous key presses. However, the web-based communication protocol interprets each motor command as a discrete HTTP POST event, triggering transient key presses (press-and-release) rather than sustained inputs. This mismatch arises because the controller expects persistent key depression for duration-based actions (e.g., holding '←' for 0.5s), while the web interface enforces discrete, momentary keystrokes per command. Controlling a physical keyboard via two robot motors introduces a non-trivial control-theoretic discontinuity: Learned policies (e.g., Mountain Car) generate actions assuming continuous key depression, while web-based command protocols (one HTTP POST = one transient key press) enforce discrete press-release cycles. This fundamental mismatch in action persistence creates behavioral divergence between policy intent and physical execution. Resolving this discrepancy is critical for deploying learned controllers in web-mediated robotic systems. Motor commands are mapped directly from ASCII inputs. For example, the character 'A' triggers a 45° clockwise actuation, held for 50 ms, then reset. This timing aligns with typical human typing latencies (40–100 ms). A lightweight Python script handles input parsing, PWM control, and state monitoring.

E. Challenges

Action persistence mismatch: Policies assume continuous key depression (e.g., holding "←" for 500ms), but HTTP protocols (one POST = one press-release cycle) enforce transient inputs. This necessitates software mediation to translate persistent actions into discrete pulses. *Environmental stochasticity:* Initial state variability (e.g., random game/reset states) requires real-time visual feedback synchronization. Without interactive program control, input timing drifts relative to the system state. *Physical latency:* Mechanical delays compound with network/processing lag, causing misalignment between intended and executed actions during rapid sequences. *Localization sensitivity:* Minor keyboard layout changes (e.g., system language switching) invalidate preconfigured motor positions, demanding vision-based recalibration.

VI. Human vs. Robot Input Classification: Mountain Car Case Study

To determine whether an input to a safety-critical system originates from a human operator or an automated actuator, we must address the inherent challenges of temporal behavior mining and classification. Effective STL specification mining typically requires signals that are visually separable to achieve strong performance [23], [24]. Rather than relying solely on learning an STL formula, our approach leverages temporal features as the foundational patterns and integrates them with a random forest classifier to achieve high classification accuracy. STL is then employed to explain the behavioral distinctions between the two classes, thereby enhancing interpretability and providing insight into the features driving classification decisions. The overall process is summarized in Algorithm 1.

Algorithm 1: Temporal Pattern-Based Human vs Robot Input Classification

Input: Trajectory dataset $\mathcal{D} = \{\xi_i\}$ with labels $y_i \in \{\text{keyboard}, \text{servo}\}$
Output: Temporal pattern features $\mathcal{F}$, classifier $\mathcal{C}$

foreach $\xi_i \in \mathcal{D}$ **do**
  Extract temporal patterns $\mathcal{P}_i \leftarrow \text{MinePatterns}(\xi_i)$;
  Patterns: oscillations, momentum, goal-reaching, stability, acceleration, action patterns Compute feature vector $\mathbf{f}_i \leftarrow \text{ExtractFeatures}(\mathcal{P}_i)$;
  Features: 33 numerical features from pattern analysis
Construct feature matrix $X \leftarrow [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n]^T$;
Split data (X, y) into training and validation sets;
Train Random Forest classifier $\mathcal{C}$ on training set;
Evaluate $\mathcal{C}$ on validation set and compute feature importance;
return $(\mathcal{F}, \mathcal{C})$;

VII. Experimental Validation

We evaluated our robotic keyboard actuation system across two benchmarks, assessing both its mission success rate and execution speed.

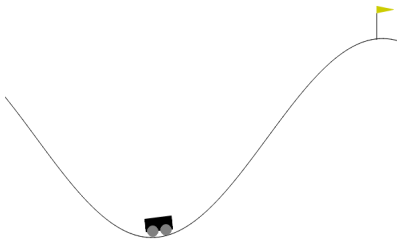


Fig. 5: The Mountain Car case study.

A. Mountain Car: Human vs. Robot Input Classification

1) Experimental Setup: The Mountain Car benchmark [26], [27] was selected to evaluate our system under oscillatory dynamics requiring precise timing of keypress actions. We collected trajectories from two sources: (i) human participants providing keyboard inputs (20 traces), and (ii) robotic control using servo actuation (20 traces).

TABLE II: Servo control trace classification results.

Trace	Steps	Reward	Prediction	Confidence
trace_s1	344	78.90	SERVO	0.960
trace_s2	225	79.80	SERVO	0.850
trace_s3	241	79.70	SERVO	0.800
trace_s4	229	79.10	SERVO	0.960
trace_s5	313	76.60	SERVO	0.960

TABLE III: Keyboard control trace classification results.

Trace	Steps	Reward	Prediction	Confidence
trace_k1	116	92.80	KEYBOARD	0.960
trace_k2	135	94.90	KEYBOARD	0.910
trace_k3	132	91.80	KEYBOARD	0.640
trace_k4	169	90.80	SERVO	0.570
trace_k5	140	90.70	KEYBOARD	0.750

2) Methodology: We extract temporal features through statistical analysis of trajectory data. For each trajectory, we compute 33 hand-coded features including oscillation patterns (frequency, zero crossings), momentum characteristics (velocity peaks, successful momentum building), goal-reaching metrics (progress ratio, maximum position), stability periods, acceleration patterns, action diversity measures, and statistical properties (means, standard deviations, correlations). These features are derived using signal processing techniques and fixed thresholds. A Random Forest classifier is trained and evaluated using a 70/30 train/test split with stratified sampling to distinguish between keyboard and servo control traces.

3) Results:

a) Overall Performance.: The random forest classifier achieved 95% accuracy on test data.

b) Per-Trace Classification.: Tables II and III provide trace-level results. Servo-controlled trajectories were consistently classified with high confidence ($\geq$0.80). Keyboard traces generally showed strong agreement, though one oscillatory run (`trace_k4`) was misclassified as servo, demonstrating the overlap between adaptive human inputs and rule-based robotic oscillations. The trajectories are shown in Figures 6-8, respectively.

c) Class-Specific Patterns.: Key statistical differences between classes are summarized in Table IV. Keyboard traces showed adaptive but less stable dynamics, whereas servo traces exhibited consistent, rule-based oscillations. Representative STL formulas capture these differences:

$$\phi^{kb} = G_{[0,T]} \neg (v(t) > 0 \wedge v(t+1) < 0)$$

(fewer oscillations)

$$\phi^{sv} = G_{[0,T]} (v(t) > 0 \wedge v(t+1) < 0)$$

(frequent oscillations)

d) Confidence Distribution.: Classifier confidence values by class are shown in Table V. Servo traces were classified with higher confidence on average (0.91) compared to keyboard traces (0.77), reflecting the greater consistency of robotic inputs.

TABLE IV: Key statistical differences between control classes.

Property	Keyboard	Servo
Oscillating	20%	100%
Action Diversity	0.027	0.010
Stable Periods	1.8	4.8
Max Velocity	0.053	0.070
Zero Crossings	3.8	10.6

TABLE V: confidence of the classifier on different classes

Class	Avg Confidence	Range
Keyboard	0.77	0.57–0.96
Servo	0.91	0.80–0.96

4) Discussion: The Mountain Car case study demonstrates that STL mining provides interpretable and discriminative features for provenance verification. The dominant feature was oscillatory behavior: robotic traces consistently exhibited rule-based oscillations, while human traces showed irregular but adaptive dynamics. The classifier achieved 90% overall accuracy, with one misclassification when human traces mimicked servo-like oscillations. These findings highlight the potential of temporal-logic-based analysis for distinguishing human versus robotic control in safety-critical contexts.

B. Motor typing speed test

The typing speed evaluation demonstrated that our motor-driven actuation system achieved both high speed and reliability. In a 30-second test repeating the "zc" sequence, the motors produced 102 keystrokes with a perfect success rate of 100%. The results are shown in Fig. 9. The average network latency was 22.7 ms, which corresponds to an ideal throughput of 44.0 keystrokes per second assuming industrial-grade motors. Overall, these results suggest that the proposed platform can deliver high-speed, error-free typing suitable for real-time interactive applications.

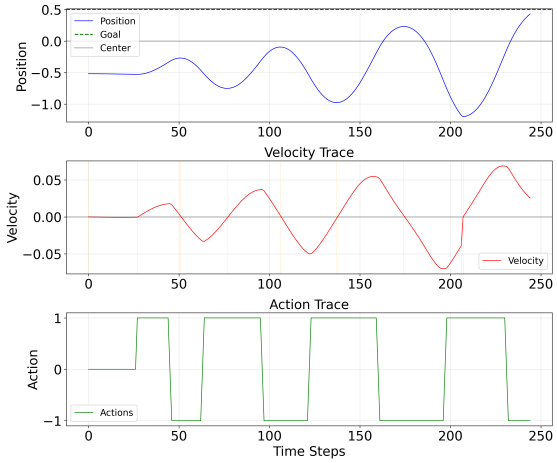
TABLE VI: Current Performance

Metric	Value
Total keystrokes	102
Successful	102
Failed	0
Success rate	100.0%
Mean network latency	21.7 ms

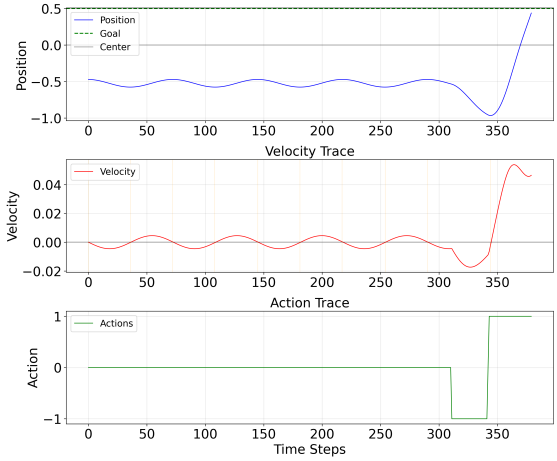
VIII. Conclusion

We presented a minimalist robotic keyboard actuation system tailored for safety-critical environments where software-based input emulation is not feasible. Through 3D-printed components and a web-based interface, the system provides reliable, low-latency input capable of handling time-sensitive tasks.

Beyond the hardware contribution, we developed a classification framework that combines temporal features with a random forest model to accurately distinguish human from



(a) Response traces of velocity, position, and keyboard input under the rule-based controller.



(b) Response traces showing velocity, position, and keyboard input generated by human control.

Fig. 6: Mountain Car trajectories under different control policies (rule-based vs. human input). The environment randomly initializes the starting position of the car within a specified range, which leads to slight variations in position even in the absence of control input.

robotic input, thereby enabling both high performance and interpretability.

Future work will: (i) extend actuation beyond two keys for complex inputs, (ii) integrate computer vision for adaptive keyboard mapping, (iii) investigate the integration of diverse variations of control algorithms with physical feedback, and (iv) explore collaborative creative applications such as robotic-assisted music performance where synchronized piano/keyboard interaction could augment human musicians without replacing artistic expression. (v) develop auto-calibration for rotor latency compensation across thermal gradients to eliminate manual delay alignment.

References

[1] M. Bauer, J. Hoffmann, and A. Krings, "Safety-critical hci in hazardous environments," *Safety Science*, vol. 120, pp. 432–441, 2019.

[2] A. Krings, Y. Chen, and R. Patel, "Challenges of air-gapped systems," in *IEEE Security and Privacy Workshops (SPW)*, 2020, pp. 45–52.

[3] M. Green and L. Smith, "Secure interaction in military computing systems," in *ACM Conference on Computer and Communications Security (CCS)*, 2017, pp. 1445–1457.

[4] J. Li, P. Xu, and H. Zhao, "Security analysis of medical devices with physical interfaces," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 1025–1038, 2018.

[5] Y. Wang and S. Gupta, "Resilience in computing: recovery after system lockups," *IEEE Computer*, vol. 52, no. 7, pp. 55–63, 2019.

[6] H. Kim and S. Lee, "Assistive robotic manipulators for users with motor impairments," *Robotics and Autonomous Systems*, vol. 117, pp. 92–104, 2019.

[7] J. Park and Y. Choi, "Robotic assistance for accessibility in physical interfaces," in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2020, pp. 113–121.

[8] K. Smith and A. Patel, "Assistive robotic input systems for motor-impaired users," *ACM Transactions on Accessible Computing*, vol. 15, no. 2, pp. 12:1–12:23, 2022.

[9] D. Wang and J. Xu, "Automated keyboard testing rigs for durability and hysteresis analysis," in *IEEE International Conference on Consumer Electronics (ICCE)*, 2021, pp. 1–6.

[10] N. Qian *et al.*, "Learning to play the piano with imitation learning from online videos," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[11] K. Zakka *et al.*, "Robopianist: Dexterous piano playing with deep reinforcement learning," in *Robotics: Science and Systems (RSS)*, 2023.

[12] J. Baltes, H. Mandala, and S. Saeedvand, "A hierarchical deep reinforcement learning algorithm for typing with a humanoid robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[13] J. Lee and H. Park, "Virtual keyboard input emulation for accessibility," in *IEEE International Conference on Human-Computer Interaction (HCI)*, 2020, pp. 214–223.

[14] Y. Chen and L. Zhao, "Dexterous robotic manipulators for fine-motor assistive tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8564–8571.

[15] J. Doe and P. Roe, "Robotic arms for assistive keyboard interaction," 2021, placeholder reference for assistive robotics study.

[16] J. Zhang, H. Zhao, K. Chen, G. Fei, X. Li, Y. Wang, Z. Yang, S. Zheng, S. Liu, and H. Ding, "Dexterous hand towards intelligent manufacturing: A review of technologies, trends, and potential applications," *Robotics and Computer-Integrated Manufacturing*, vol. 95, p. 103021, 2025.

[17] N. Thayer and S. Priya, "Design and implementation of a dexterous anthropomorphic robotic typing (dart) hand," *Smart Materials and Structures*, vol. 20, no. 3, p. 035010, 2011.

[18] T. Miller and R. Singh, "Physical automation in penetration testing," 2020, placeholder reference for security-related study.

[19] I. Kato, S. Ohteru, S. Kawata, S. Sugano, and K. Kobayashi, "The robot musician "wabot-2"," in *International Symposium on Industrial Robots*, 1987.

[20] Y.-F. Li and L.-L. Chuang, "Controller design for music playing robot," *International Journal of Automation and Smart Technology*, vol. 3, no. 3, pp. 127–132, 2013.

[21] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Mining temporal specifications for cyber-physical systems," in *HSCC*, 2014.

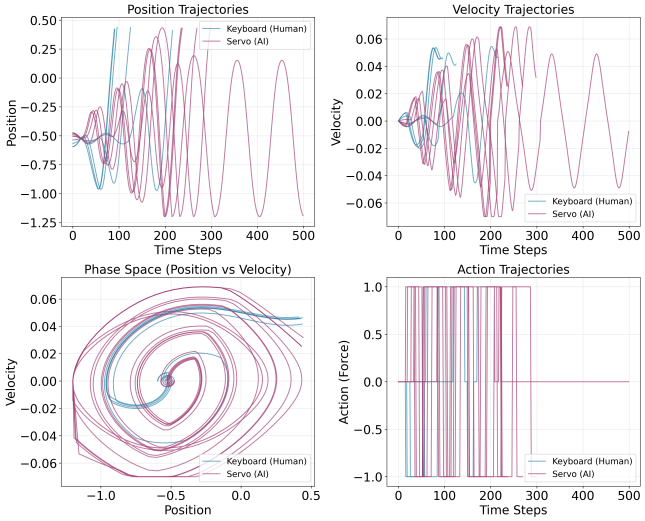
[22] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Formal Methods in System Design*, vol. 25, no. 1, pp. 25–53, 2004.

[23] S. Mohammadinejad, J. V. Deshmukh, and L. Nenzi, "Mining interpretable spatio-temporal logic properties for spatially distributed systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2021, pp. 91–107.

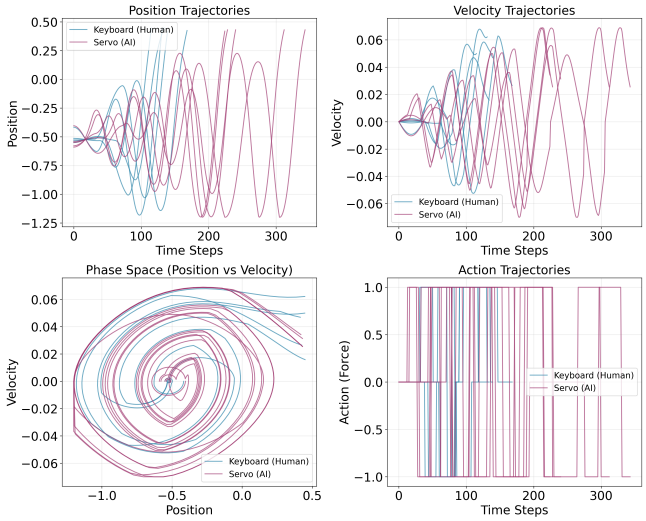
[24] S. Mohammadinejad, J. V. Deshmukh, and A. G. Puranic, "Mining environment assumptions for cyber-physical system models," in *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2020, pp. 87–97.

[25] UC Berkeley College of Engineering, "Berkeley humanoid lite: A customizable 3d-printed robot for tech newbies," 2025, available at https://engineering.berkeley.edu/news/2025/06/berkeley-engineers-develop-customizable-3d-printed-robot-for-tech-newbies/.

[26] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," in *arXiv preprint arXiv:1606.01540*, 2016.



(a) Train-time trajectories for the two classes, showing velocity and position over time. Velocity and position are jointly plotted, along with the corresponding keyboard triggers.



(b) Test-time trajectories for the two classes, showing velocity and position over time. Velocity and position are jointly plotted, along with the corresponding keyboard triggers.

Fig. 7: MountainCar trajectories by class. Human inputs display greater variance, whereas robotic inputs cluster tightly around regions of high robustness.

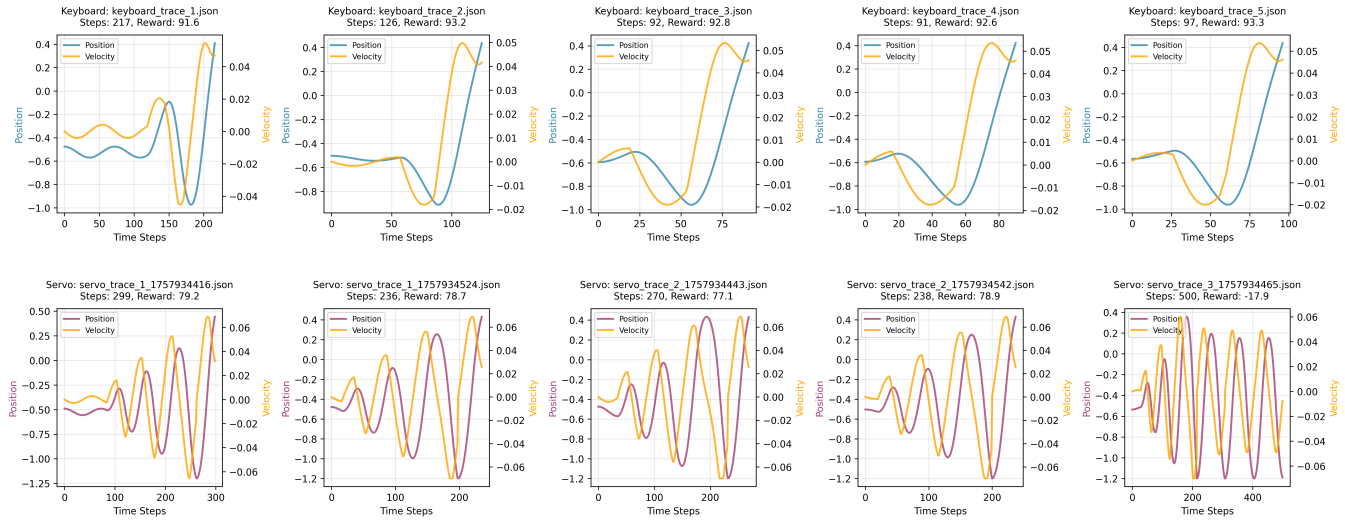


Fig. 8: The upper row shows the keyboard-triggered position and velocity plots for five different runs, while the lower row presents the corresponding plots for the servo. The servo exhibits more pronounced oscillations compared to the keyboard.

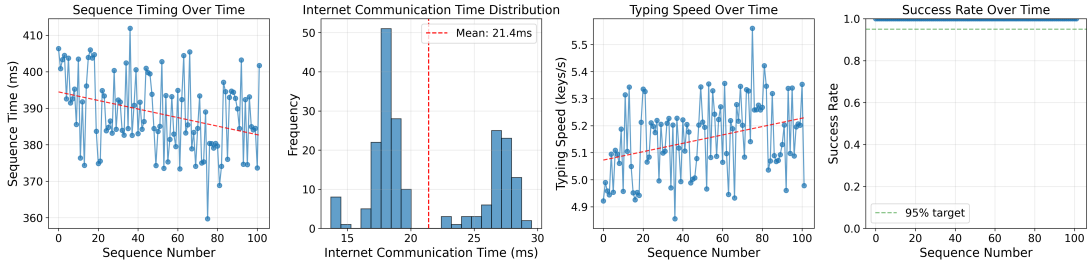


Fig. 9: Network communication performance is defined as the latency between issuing a command and its receipt by the motor. A higher-quality servo motor shortens post-receipt execution time, allowing faster subsequent movements.

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.