Application of Greedy Algorithm for Adaptive Mouse Sensitivity Adjustment Based on Accuracy and Shoot-Deviation Evaluation in Valorant

Al Farabi
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
NIM: 13524086
Email: 13524086@std.stei.itb.ac.id, alfarabi0642@gmail.com

Abstract—Mouse sensitivity in a tactical shooter is usually tuned by trial and error. This paper presents a small 2D calibration program that treats the tuning step as a greedy decision problem. The program records the virtual crosshair trajectory for every target, classifies the primary movement as overshoot, undershoot, direct, lateral error, or invalid, then recommends the next Valorant-style sensitivity after each round. The DPI is fixed and only the simulated crosshair sensitivity changes. The greedy rule uses the latest round only: undershoot evidence increases sensitivity, overshoot evidence decreases it, and low-confidence evidence keeps the value unchanged. The adjustment size is bounded by a round schedule but scaled by directional severity, directional confidence, and a composite performance score. The final implementation uses a fixed report protocol of 1200 DPI, initial sensitivity 0.15, no warm-up, five recorded rounds, and fifteen targets per round. A single controlled session under the fixed report protocol produced a monotonic sensitivity increase from 0.15 to 0.209, driven by persistent undershoot evidence across all five rounds. The stability indicator was not reached, suggesting that the initial sensitivity was below the participant's preferred value for this task. The work shows that the adaptive greedy process can be implemented, observed, and audited without changing Windows, mouse firmware, or Valorant settings.

Index Terms—greedy algorithm, mouse sensitivity, Valorant, trajectory analysis, pointing task

I. Introduction

A player can change mouse DPI, in-game sensitivity, Windows pointer settings, or all three. In a game such as Valorant, this makes a simple question surprisingly awkward: should the player raise sensitivity, lower it, or leave it alone after missing a target? The common answer is manual tuning. The player changes a number, plays a few rounds, and guesses again.

This paper takes a narrower path. It asks whether a greedy algorithm can be used to make that next local choice in a controlled 2D calibration program. The program does not attach itself to Valorant. It does not edit any operating-system setting. It shows targets on a 1920 by 1080 Pygame window, records a virtual crosshair path, and recommends the next simulated sensitivity. DPI is treated as fixed hardware input. Sensitivity is the only decision variable.

The registered title mentions accuracy and shoot-deviation evaluation. Accuracy is still measured, but the main design issue turned out to be earlier than the click. A click can be accurate after a bad first movement. For example, the crosshair may pass the target, return, and then click near the center. A program that only stores the final click sees a good shot. A program that stores the trajectory sees overshoot and correction. That distinction matters because sensitivity tuning is about movement tendency, not only the final point.

The contribution of this work is a working adaptive greedy calibration pipeline with four parts. First, it maps Valorant-style sensitivity into a 2D virtual crosshair. Second, it analyzes the primary movement toward each target. Third, it turns the latest round into one greedy action: increase, keep, or decrease. Fourth, it stores data in CSV files that can be checked before being used in a report.

The claim is intentionally limited. This paper does not prove a globally optimal sensitivity. It also does not claim that a 2D Pygame target task is the same as aiming inside Valorant. The result is an implementation proof: the proposed greedy process exists as runnable software, the trajectory logic is testable, and the data pipeline can reject logs that do not match the report protocol.

II. Background

A. Greedy decision making

A greedy algorithm builds a solution by taking a local choice that looks best at the current step. It does not search every future sequence, and it does not undo previous decisions unless the problem design explicitly permits that behavior. Standard algorithm texts describe greedy algorithms through this local-choice property and through problems where a locally chosen component can lead to a valid solution [1].

The calibration problem in this paper is not an optimization problem with a known proof of optimality. There is no closed form saying that sensitivity S is best for a player. The greedy part is therefore practical rather

than proof-theoretic. At the end of round i, the algorithm only sees the current sensitivity and the summary of that round. It then chooses one action:

$$\{\text{increase}, \text{keep}, \text{decrease}\}.$$

The method is greedy because it applies the chosen action immediately to round $i + 1$ without testing all possible sensitivity sequences.

B. Pointing task measures

The target task is a pointing task. Fitts' law is the standard starting point for reasoning about aimed movement because it relates movement time to target distance and target width [2]. Later HCI work adapted the model to computer pointing and discussed the use of throughput, effective width, and error-aware evaluation [3], [4]. Two-dimensional pointing needs extra care because a target has area and a movement has both longitudinal and lateral components [5].

This paper does not fit a Fitts' law regression. The task has only one target radius and the goal is not to compare devices. Still, the same speed-accuracy tension appears. The program records accuracy, click deviation, and response time for every target. These metrics are used in the round score. Directional sensitivity changes, however, come from trajectory evidence: overshoot means the first movement went too far, and undershoot means it stopped short.

C. Sensitivity units

Effective DPI, or eDPI, is often described as mouse DPI multiplied by in-game sensitivity. GamingSmart describes eDPI as a game-specific value because each game scales sensitivity differently [7]. Sensitivity converters usually preserve cm/360, the physical mouse distance needed for a full turn. Aiming.Pro states this explicitly for Valorant conversion: converted sensitivity keeps cm/360 the same [6].

The program uses that idea as a scale anchor. The 2D simulator applies a Valorant-style yaw constant of 0.07 degrees per mouse count. For mouse delta Δ, sensitivity S, and screen width W, the program computes:

$$\theta = \Delta \cdot S \cdot 0.07,$$

$$\Delta_{px} = \theta \cdot \frac{W}{103}.$$

The value 103 is used as a horizontal field-of-view approximation for the 2D screen mapping. The cm/360 diagnostic value is:

$$cm/360 = \frac{360 \cdot 2.54}{DPI \cdot S \cdot 0.07}.$$

For the report protocol, $DPI = 1200$ and $S = 0.15$, so $cm/360 \approx 72.57$. This is not raw Valorant input. Pygame provides relative mouse movement through pygame.mouse.get_rel(), and its documentation also notes that hidden cursor plus grabbed input creates a virtual input mode where relative movement is not stopped by screen borders [8]. The simulator uses that mode.

III. Adaptive greedy formulation

A. Trajectory representation

For one target, let p_0 be the crosshair position when the target appears and let c be the target center. The target radius is r. The initial distance is:

$$D = ||c - p_0||.$$

The ideal direction is:

$$u = \frac{c - p_0}{D}.$$

Each sampled crosshair position p_k is projected onto the ideal direction:

$$q_k = (p_k - p_0) \cdot u.$$

The target center lies at $q = D$. The near and far longitudinal edges are approximated by $D - r$ and $D + r$. Lateral error is the distance from p_k to this ideal line after removing the projected component.

The program records one initial sample for each target, then appends samples until the player clicks or the target times out. Each motion sample stores session id, round index, target index, timestamp, crosshair coordinates, and raw mouse delta.

B. Primary movement

The algorithm separates primary movement from correction movement. In the implementation, the primary endpoint is chosen from the smoothed projection sequence unless the raw sequence shows a clear fast overshoot. The logic watches for two events: a longitudinal reversal beyond the correction threshold, or a short pause before the target. The implementation also keeps a raw maximum projection so that fast overshoot is not hidden by smoothing.

This choice was made because of a concrete failure mode found during testing. When the user intentionally moved past the target and corrected back, the earlier classifier sometimes labeled the shot as lateral or undershoot. Prioritizing longitudinal overshoot over lateral drift fixed that class of error. Lateral drift is only chosen after clear overshoot and undershoot checks fail.

C. Target classification

The target classifier returns one label.

Overshoot:

$$\max(q_k) > D + r + \epsilon.$$

Undershoot:

$$q_{primary} < D - r - \epsilon.$$

Direct movement means the primary movement reaches the target interval without a large correction. Lateral error means the longitudinal movement is not clearly too far or too short, but the crosshair drifts sideways beyond the lateral threshold. Invalid means the trajectory cannot vote, usually because there are too few samples or the initial target distance is too small.

Directional error is signed:

$$e_j = \begin{cases} \frac{D-r-q_{primary}}{D}, & \text{undershoot,} \\ -\frac{\max(q_k)-(D+r)}{D}, & \text{overshoot,} \\ 0, & \text{direct or lateral.} \end{cases}$$

Positive error asks for higher sensitivity. Negative error asks for lower sensitivity.

D. Round summary

For each round, the program stores hit count, miss count, timeout count, accuracy, average deviation, average response time, class counts, correction average, directional confidence, severity, and net directional error.

The confidence value is:

$$C = \frac{|\sum e_j|}{\sum |e_j|}.$$

Only overshoot and undershoot samples enter the denominator. Direct and lateral targets do not lower confidence because they do not vote for a direction. If overshoot and undershoot cancel each other by magnitude, confidence becomes low. If one signed direction dominates, confidence approaches one.

Severity is:

$$V = \frac{|\sum e_j|}{n_d},$$

where n_d is the number of overshoot and undershoot targets.

IV. Greedy recommendation rule

The greedy rule starts with direction. If C is below the configured threshold, the action is keep. If net directional error is positive, the action is increase. If it is negative, the action is decrease.

The maximum adjustment schedule is:

$$d_i = \begin{cases} 0.20, & i = 1, \\ 0.10, & i = 2, \\ 0.05, & i \geq 3. \end{cases}$$

Those numbers are upper bounds, not direct changes. The actual adjustment ratio is computed from severity, confidence, and performance pressure:

$$severity\_factor = \min(1, \frac{V}{V_{full}}),$$

$$performance\_error = 1 - G,$$

Round flow
Fixed profile: DPI 1200, sensitivity 0.15
↓
Show 15 targets and record motion samples
↓
Classify each trajectory
↓
Summarize accuracy, deviation, time, direction
↓
Greedy action: increase, keep, or decrease
↓
Use recommended sensitivity in the next round

Fig. 1. Report-session flow used by the final program.

$$R = \min(1, 0.50 \cdot severity\_factor + 0.30 \cdot C + 0.20 \cdot performance\_error). \quad (1)$$

Here G is the composite score:

$$G = 0.5A + 0.3P + 0.2T,$$

where A is accuracy, P is precision score, and T is response-time score. The next sensitivity is:

$$S_{i+1} = \begin{cases} S_i(1 + d_i R), & \text{increase,} \\ S_i(1 - d_i R), & \text{decrease,} \\ S_i, & \text{keep.} \end{cases}$$

The final value is clamped to $[0.01, 2.00]$ and rounded before storage.

V. Program design

A. Runtime

The final application is written in Python 3.12 with Pygame and Matplotlib. The report collector has a fixed protocol: one anonymous session, 1200 DPI, initial Valorant sensitivity 0.15, no warm-up, five recorded rounds, and fifteen targets per round. The normal application still has a setup screen, but the report path uses fixed values so the paper dataset is reproducible.

The simulator starts each round with the crosshair at the screen center. It does not reset the crosshair between targets inside a round, so the next target starts from wherever the last target ended. This makes the path closer to an aiming drill than to fifteen independent clicks from the center.

B. Storage

The data model has four CSV layers. Raw target events store target position, click position, hit state, deviation, response time, timeout state, DPI, sensitivity, and sequence id. Motion samples store the full path while a target is active. Target summaries store the classifier output and directional error. Round summaries store the greedy decision and aggregate metrics.

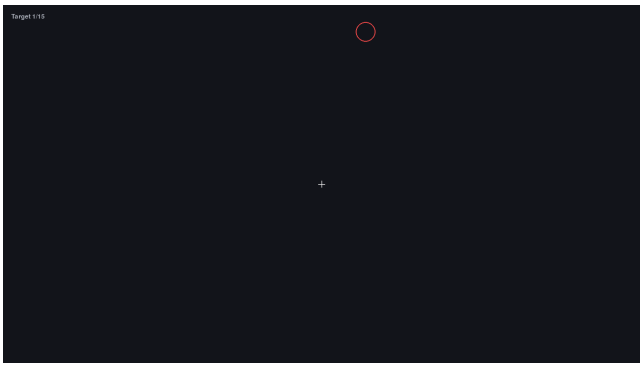

Fig. 2. Pygame target screen captured from the implemented calibrator.

TABLE I
CSV outputs used by the report pipeline

File	Main contents
sessions.csv	DPI, initial sensitivity, final sensitivity, round count, status
raw_events.csv	Target position, click position, hit, deviation, response time
motion_samples.csv	Timestamped crosshair coordinates and mouse delta
target_summaries.csv	Classification, primary projection, signed directional error
round_summaries.csv	Scores, class counts, confidence, severity, greedy decision

TABLE II
Fixed report protocol

Parameter	Value
Participant count	One
DPI	1200
Initial sensitivity	0.15
Rounds	5
Targets per round	15
Warm-up	Disabled
Target radius	30 px
Timeout	2 s per target
Sensitivity bounds	0.01 to 2.00

The report validator rejects incomplete report data. It expects exactly one complete session, exactly five rounds, no warm-up events, fifteen target events per round, motion samples for all targets, complete target summaries, one round summary per round, and final sensitivity equal to the last recommendation. This matters because the paper should not accidentally mix rehearsal logs with the final report protocol.

VI. Report pipeline

The program separates calibration, validation, plotting, and table extraction. This split is intentional. The runner records data, but it does not decide whether a session is usable for the paper. A separate validator checks the files after the session ends. If a player aborts early, if warm-up rows appear in the report folder, or if one of the five rounds has fewer than fifteen targets, the dataset is rejected before any figure is generated.

The report commands are short enough to repeat:

```
python collect_report_data.py
python validate_dataset.py
python plot_results.py
python generate_report_tables.py
```

The collector writes only to data/report. Rehearsal data and older application logs stay in data/application. This prevents a common reporting mistake: running the program during development, changing the algorithm, and then mixing old logs with final logs. A session may look complete to a human reader but still fail the validator because it belongs to the wrong protocol.

The plotting script creates four PDF figures: sensitivity over rounds, trajectory-classification counts, selected metrics, and greedy decisions. The table generator creates one CSV summary with initial sensitivity, final sensitivity, round count, the sensitivity sequence, mean accuracy, mean deviation, mean response time, class totals, and final recommendation. Those generated files are used in the results section after the completed report session passed validation.

This pipeline also protects the algorithm claim. The paper can say that the greedy rule was implemented, checked, and observed in one completed report session. It still cannot claim that the player's aim improved in general. The observed evidence supports a narrower statement: sensitivity increased monotonically because undershoot dominated the recorded session.

VII. Results and analysis

One complete report session was collected under the fixed protocol in Table II. The session used 1200 DPI and initial sensitivity 0.15, ran five rounds of fifteen targets without warm-up, and completed without abort. The automated validator confirmed the dataset before any figure or table was generated. Over the five rounds, sensitivity increased monotonically from 0.15 to 0.2089, a net change of 39.3%. Mean accuracy was 94.67%, mean click deviation 19.02 pixels, and mean response time 0.777 seconds. The stability indicator was not reached.

A. Sensitivity trajectory

Fig. 3 shows the recommended sensitivity at each round boundary. Every round produced an increase decision because undershoot evidence consistently outweighed overshoot evidence. The largest single-round change occurred in round 1, where the 20% schedule cap allowed sensitivity to move from 0.15 to 0.1712. Later rounds used the 10% and 5% caps and produced smaller steps.

B. Trajectory classifications

Table III and Fig. 4 show the trajectory classification counts per round. Undershoot was the dominant class in every round, totaling 50 out of 75 targets (66.7%).

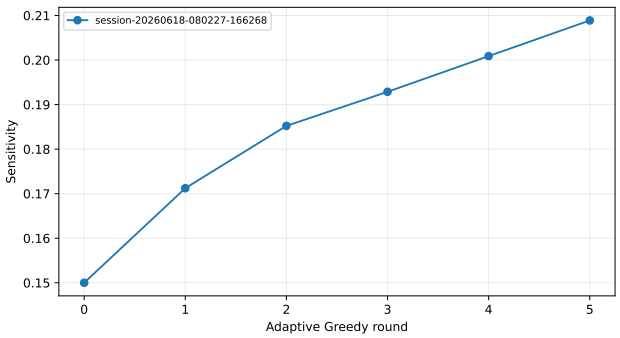

Fig. 3. Sensitivity trajectory across five adaptive greedy rounds.

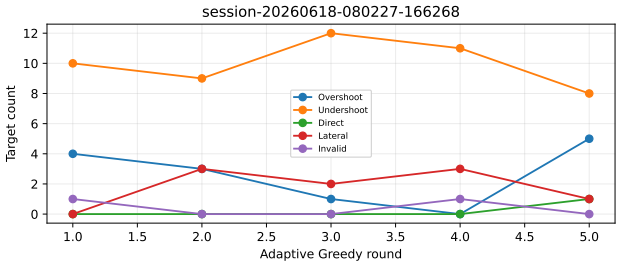

Fig. 4. Trajectory classification counts per round.

Overshoot totaled 13 targets (17.3%). Round 4 had the strongest undershoot dominance: 11 undershoot, 0 overshoot, confidence $C = 1.0$. Round 5 showed a partial shift with 5 overshoot targets as sensitivity reached 0.2009, though undershoot still outnumbered overshoot. Lateral classifications appeared in rounds 2 through 5, totaling 9 targets, and did not vote on sensitivity direction.

C. Performance metrics

Fig. 5 shows accuracy, average click deviation, and average response time across rounds. Accuracy was 100% in round 1 and 93.3% in rounds 2 through 5, each recording exactly one miss. Average deviation was stable between 13.90 and 14.33 pixels in rounds 1 through 3, spiked to 34.62 pixels in round 4, then returned to 18.17 pixels in round 5. Response time ranged from 0.725 to 0.828 seconds with no monotonic trend.

D. Greedy decisions

Fig. 6 shows the greedy decision for each round. All five decisions were increase. Confidence was moderate in round 1 ($C = 0.620$) because four overshoot targets partially cancelled ten undershoot targets. Confidence rose in rounds 2 through 4, reaching $C = 1.0$ in round 4, then fell to $C = 0.890$ in round 5 as overshoot reappeared. The adjustment schedule bounded each change: 20% maximum in round 1, 10% in round 2, and 5% in rounds 3 through 5. The actual adjustment ratio was further reduced by severity, confidence, and composite score according to the formula in Section IV.

E. Software verification

The automated tests were run before data collection to confirm correct program behavior. Table IV lists the checks performed after the final program update.

These tests also caught regressions during development. One earlier classifier treated intentionally exaggerated overshoot as lateral movement. Another version detected some overshoots as undershoots after smoothing. The final classifier uses raw maximum longitudinal projection before the undershoot test, which is why an overshoot followed by a corrected click still becomes an overshoot vote.

VIII. Discussion

The greedy structure is simple, but the input to the greedy rule is not just click accuracy. That choice is the main point of the implementation. If the program reacted only to misses, a round with 0 percent accuracy and large deviation would still be ambiguous. The player may need lower sensitivity, higher sensitivity, or better lateral control. The trajectory split narrows that ambiguity: longitudinal overshoot and undershoot vote on sensitivity, while lateral error is reported without forcing a sensitivity change.

The confidence formula also avoids a common mistake. Direct and lateral targets are not votes. If one target is a strong overshoot and fourteen targets are direct, the directional confidence remains high for the one directional sample, although the severity and performance score still control how much the sensitivity can move. If overshoot and undershoot both appear with similar magnitude, confidence drops and the greedy rule keeps the current sensitivity.

There is a tradeoff. A greedy algorithm responds quickly, which is useful in an interactive calibration tool. It also inherits the weakness of local decisions. Fatigue, learning, and target order can affect a single round. The schedule reduces movement from 20 percent to 10 percent to 5 percent, but it does not remove noise. A future comparison against grid search, binary search, or Bayesian optimization would be needed before making stronger claims about calibration quality.

IX. Limitations

The simulator is not Valorant. It uses a 2D target field, a fixed target radius, and relative mouse input from Pygame. Valorant has camera rotation, weapons, recoil, movement, map geometry, opponents, and raw-input behavior that this program does not model.

The yaw constant and 103-degree horizontal FOV are treated as practical assumptions. They make the sensitivity scale more meaningful than a relative multiplier, but they do not guarantee identical game feel. The program also assumes that one participant and five rounds are enough for a report on implementation feasibility. That is enough for the contribution claimed here. It is not enough for a population-level claim.

TABLE III
Round-by-round observed results for the report session

Round	Sens.	Acc.	Dev.	RT	Over	Under	Direct	Lat.	Inv.	Decision
1	0.1500	1.000	13.90	0.788	4	10	0	0	1	increase
2	0.1712	0.933	14.07	0.805	3	9	0	3	0	increase
3	0.1852	0.933	14.33	0.738	1	12	0	2	0	increase
4	0.1929	0.933	34.62	0.725	0	11	0	3	1	increase
5	0.2009	0.933	18.17	0.828	5	8	1	1	0	increase

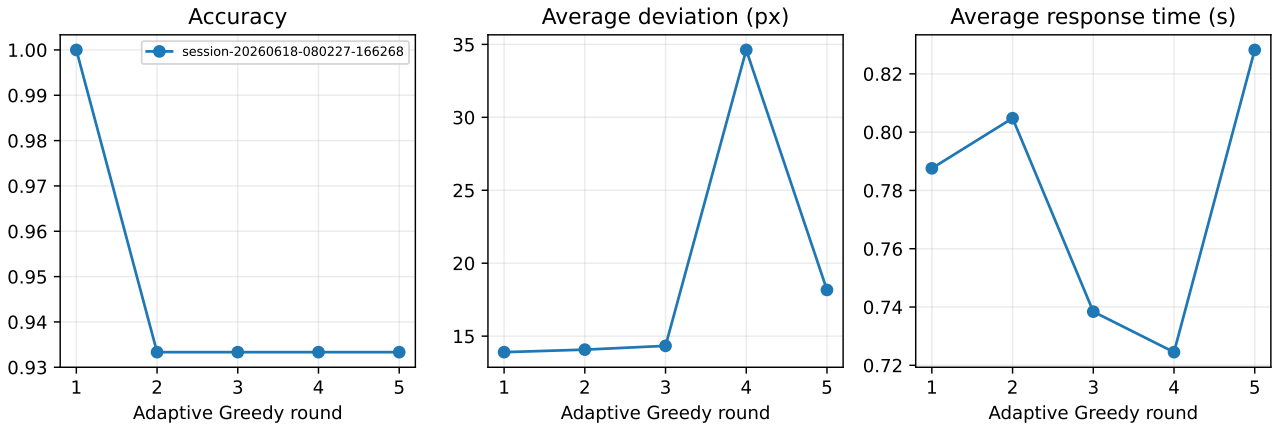

Fig. 5. Accuracy, average click deviation, and average response time per round.

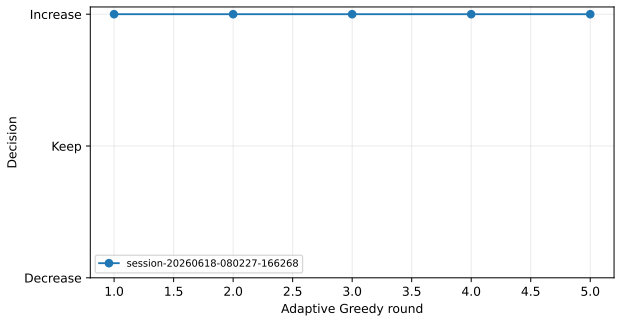

Fig. 6. Greedy decision per round (−1 = decrease, 0 = keep, +1 = increase).

TABLE IV
Software verification result

Check	Result
pytest tests -q	46 tests passed
compileall on entry scripts and package	Passed
Valorant cm/360 default test	72.57 cm/360
Report validator behavior	Tested
Plot and table generators	Tested

The session did not reach the stability indicator within five rounds. The final recommended sensitivity of 0.2089 is therefore a recommendation in progress, not a converged value. Additional rounds or sessions would be needed to determine whether convergence occurs.

X. Conclusion

This work produced a runnable adaptive greedy sensitivity calibrator for a Valorant-style 2D aiming task. The program records full crosshair trajectories, classifies each target into directional or non-directional movement classes, computes round-level confidence and severity, and applies a greedy increase, keep, or decrease decision to the next round. The implementation keeps DPI fixed and changes only the simulator's virtual crosshair sensitivity.

A completed report session under the fixed protocol showed a monotonic sensitivity increase from 0.15 to 0.2089 across five rounds, with undershoot dominating every round (50 out of 75 targets). All five greedy decisions were increase. The stability indicator was not reached, indicating that additional rounds would be needed to test convergence. Accuracy remained above 93% throughout the session. Future work could extend the protocol to more rounds, include multiple participants, or compare the adaptive greedy rule against grid search, binary search, or Bayesian optimization.

Acknowledgment

The author thanks the IF2211 Strategi Algoritma teaching team for the technical-report assignment format and topic framing.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[2] P. M. Fitts, “The information capacity of the human motor system in controlling the amplitude of movement,” Journal of Experimental Psychology, vol. 47, no. 6, pp. 381-391, 1954.
[3] I. S. MacKenzie, “Fitts’ law as a research and design tool in human-computer interaction,” Human-Computer Interaction, vol. 7, no. 1, pp. 91-139, 1992.
[4] R. W. Soukoreff and I. S. MacKenzie, “Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts’ law research in HCI,” International Journal of Human-Computer Studies, vol. 61, no. 6, pp. 751-789, 2004.
[5] I. S. MacKenzie and W. Buxton, “Extending Fitts’ law to two-dimensional tasks,” in Proc. SIGCHI Conf. Human Factors in Computing Systems, 1992, pp. 219-226.
[6] Aiming.Pro, “Valorant mouse sensitivity converter calculator,” 2026. [Online]. Available: https://aiming.pro/mouse-sensitivity-calculator/valorant. Accessed: Jun. 16, 2026.
[7] GamingSmart, “eDPI calculator,” 2026. [Online]. Available: https://gamingsmart.com/edpi-calculator/. Accessed: Jun. 16, 2026.
[8] Pygame Developers, “pygame.mouse,” Pygame v2.6.0 documentation, 2023. [Online]. Available: https://www.pygame.org/docs/ref/mouse.html. Accessed: Jun. 16, 2026.

Pernyataan

I, the undersigned, hereby declare that the paper I have written is my own work, not an adaptation, not a translation of another person’s paper, and not plagiarism.

Bandung, 18 June 2026

Al Farabi
13524086