

KiCaT: Fast Visual Keystroke Tracking on Any Keyboard Image

Lawrence Amadi, Andrew Lu, Chih-Hsien Chou, Ning Lu; Futerwei Technologies; Austin, USA

Abstract

Typing remains a primary mode of interaction in extended reality (XR), yet most XR systems still depend on physical keyboards or specialized sensing hardware. We present a vision-only keystroke tracking pipeline for surface typing that combines visible-key detection, recovery of hand-occluded keys, and finger keypress event detection on monocular video. The system first detects the keyboard and visible keys with a YOLO-based detector trained on oriented key boxes and synthetic hand-occlusion augmentation. Occluded or missed keys are then recovered with lightweight per-key regressors driven by an efficient 3–2 recovery-key selection strategy. For keypress detection, we use monocular 3D hand reconstruction with HaMeR and an autoregressive transformer that processes temporal image sequences, recovered hand meshes, finger keypoint masks, and camera poses to estimate press-down probabilities for each finger. On the Keyboard Key Detection dataset, our visible-key detector achieves 97% mAP@75 with mean IoU around 0.9, while occluded-key recovery sustains 30 fps. On typing videos from the MSU Typing Behavior Database, the keypress module reaches up to 90% event detection accuracy with 28 ms latency. Together, these components form a practical end-to-end vision system for hardware-free keystroke tracking on printed, projected, or rendered keyboard layouts.

Introduction

Typing is still one of the most important interaction primitives for productivity, search, messaging, and command entry. This need does not disappear in extended reality (XR); in fact, it becomes more pronounced because immersive systems increasingly aim to support mobile and untethered work. However, current XR text-entry workflows still depend heavily on physical keyboards, handheld controllers, or auxiliary sensing hardware. Those choices preserve typing performance in controlled settings, but they also reintroduce the same setup constraints that XR otherwise tries to remove. A compelling alternative is to let a user type on any convenient surface as long as a keyboard layout is visually reproduced on that surface as demonstrated in Fig. 1. The layout may be printed, projected, or rendered, while one or more cameras observe the session and infer the pressed key directly from vision. This formulation is attractive for mobile XR because it preserves the familiar keyboard typing behavior and finger contact feedback while avoiding the physical constraint of traditional keyboard peripherals. The challenge is that an end-to-end vision solution must simultaneously solve three problems: detect the visible keys precisely, infer the positions of keys hidden by the hands, and decide when a finger has actually pressed a key.

Our work addresses these three components in one unified Keyboard-image typing Camera Tracker (KiCaT) pipeline. We first train a YOLO-based detector to localize keyboards and visible keys using oriented bounding boxes and hand-occlusion augmentation. We then recover occluded or missed keys using

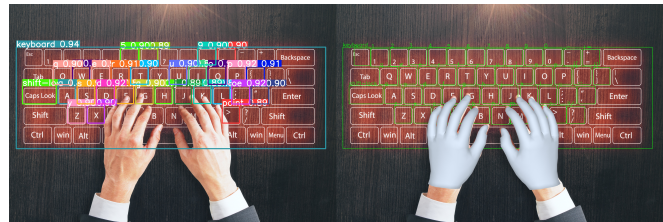


Figure 1. Illustration of surface typing on a rendered keyboard layout which involves real-time keyboard and visible key detection (left), recovery of hand-occluded keys, hand tracking, and finger press-down detection (right).

lightweight per-key regressors whose inputs are selected by a fast 3–2 lookup strategy derived from ranked five-key combinations. Finally, we reconstruct the hands in 3D from monocular input and infer press-down events with an autoregressive transformer over temporal visual and geometric cues. The resulting system supports keystroke logging on keyboard layouts reproduced on arbitrary surfaces. The main contributions of this paper are three-fold. First, we present a vision-only pipeline that integrates key detection, occluded-key recovery, and finger keypress detection for surface typing. Second, we show that oriented-box detection plus synthetic hand occlusion substantially improves visible-key robustness while preserving high detection precision. Third, we pair lightweight recovery models with a practical 3–2 key-selection strategy and a transformer-based press detector to obtain an end-to-end system that is both accurate and responsive enough for real-time XR interaction.

Related Work

Text entry spans physical keyboards, mid-air input, and virtual keyboard systems. A common finding across prior work is that typing performance improves when the interaction stays close to familiar keyboard behavior, but portability and robustness suffer when systems depend on specialized hardware or tightly controlled sensing.

Physical keyboards in XR. Physical keyboard typing remains a strong baseline in immersive systems. McGill *et al.* showed that VR typing can approach normal performance when the keyboard is visually integrated into the scene [3], while Walker *et al.* highlighted the importance of visual feedback and the disruptive effect of occlusion when users type on a partially obscured physical keyboard [4]. Grubert *et al.* further studied text entry on standard keyboards in head-mounted-display VR and demonstrated the value of preserving familiar motor behavior [5]. These systems, however, still require a physical keyboard and therefore do not directly address mobile or ad hoc typing.

Gesture-based typing. Touchless text entry methods avoid a physical surface but often trade off speed, comfort, or learnability. Didehkhoshid *et al.* used a tracked tablet for text entry

in VR [1]. Amma *et al.* recognized handwritten characters in mid-air with wearable motion sensors [7], and Yi *et al.* explored ten-finger freehand typing with 3D hand tracking [8]. Lu *et al.* proposed iText, a hands-free imaginary keyboard for AR systems [2]. While these methods reduce hardware footprint, they typically sacrifice tactile contact or depend on interaction techniques that differ substantially from ordinary typing.

Virtual keyboards. Several systems detect taps or touch events with additional sensors. TapSense [10, 21] and TapID [11] detect taps using acoustic or inertial sensors, but require specialized hardware calibration and offer limited scalability. Alternatively, vision-based approaches show promise. Ponser *et al.* used cameras to detect key interactions on printed keyboards. Their shadow-based touch detection relied strongly on imaging geometry and controlled conditions but could not handle occlusion in real-time [9]. In contrast, we target a purely vision-based pipeline that does not require acoustic, inertial, or depth instrumentation.

Occlusion reasoning and hand reconstruction. Occlusion remains a central obstacle for vision-based typing because hands often hide the very keys that are likely to be pressed. General occlusion-aware detection and segmentation methods such as Mask R-CNN [12] and hand-specific reconstruction models such as HandOccNet [15, 13, 14] address related robustness problems, but not the structured recovery of missing keys in a keyboard layout. Rezanezhad *et al.* work on document layout analysis with deep learning and heuristics [17] is a relevant example of recovering regular layouts from partial observations. For hand motion, we build on HaMeR, a transformer-based model for monocular 3D hand reconstruction [16]. Our work combines these strands into a single keystroke-tracking system specialized for structured keyboard geometry and finger-surface interaction.

Method

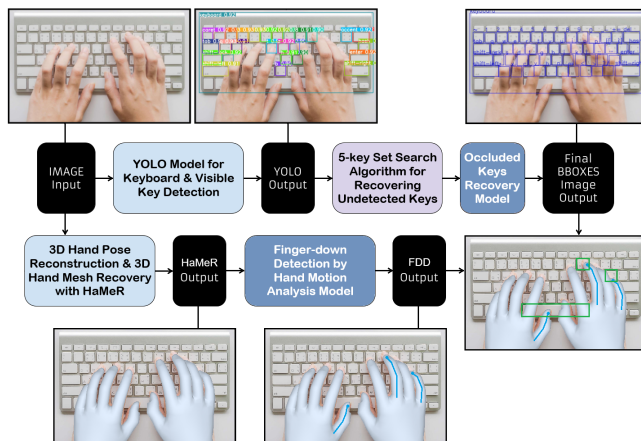


Figure 2. End-to-end vision-only keystroke tracking pipeline. The system detects the keyboard and visible keys, recovers occluded keys from visible-key geometry, reconstructs the hands in 3D, detects finger press-down events, and assigns each event to the corresponding key.

Our pipeline takes monocular RGB images of a reproduced keyboard layout and outputs time-stamped keypress events. As shown in Figure 2, it has four stages: (1) visible-key detection, (2) occluded-key recovery, (3) 3D hand reconstruction and finger

motion tracking, and (4) keypress event detection with finger-to-key assignment. The key detection branch of the pipeline provides a complete geometric layout even when hands block part of the keyboard; the hand tracking branch determines when a finger crosses into a press state. A keystroke is emitted when a detected press event is associated with the most plausible key under the corresponding finger.

Keyboard and Visible-key Detection

We treat the keyboard and its keys as fine-grained objects and train a YOLO detector for keyboard segmentation and key recognition [19]. We chose the YOLO11 network architecture with improved accuracy and speed with fewer parameters. Its low-latency design makes it well suited for real-time applications while delivering state-of-the-art performance. Training uses the Keyboard Key Detection (KKD) dataset [20], which we clean by removing poor-quality or ambiguous samples from the original release and by converting key annotations from axis-aligned bounding boxes to oriented bounding boxes. This oriented representation is important because many keyboard images are slightly rotated, and axis-aligned boxes can introduce overlap and excess background that are undesirable for downstream key assignment.

To improve robustness under typing, we augment the cleaned KKD images with synthetic hand occlusions. Cropped hand images are overlaid on the keyboards at random positions, scales, and orientations, together with conventional image augmentations such as rotation, crop, reflection, and brightness or contrast changes. The detector therefore learns to localize keys not only in clean views but also in conditions closer to real typing, where fingers and palms partially hide the layout. The outcome of these steps are illustrated in Fig. 3.



Figure 3. YOLO key detection on different types of keyboard images from MSU Typing Behavior Dataset [22] (top) and captured (bottom left). Example of key detection and hand-occluded key recovering (bottom right).

Recovering Occluded Keys from Visible Keys

Even with occlusion augmentation, some keys remain too obscured to detect directly. We therefore recover their locations from the spatial configuration of visible keys. For each recoverable key, we train a lightweight four-layer fully connected regressor that takes the coordinates of five detected keys and predicts the oriented bounding box of the missing key. This produces one compact per-key recovery model for each target key in the recoverable subset.

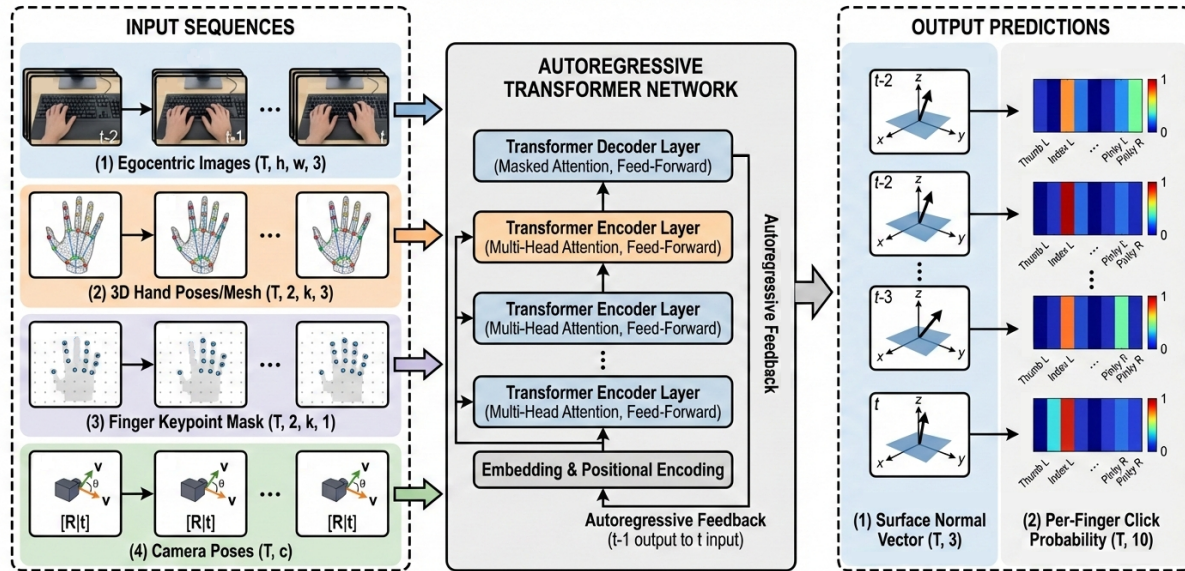


Figure 4. Autoregressive transformer-based training pipeline for click event detection. The model processes temporal sequences of images, 3D hand meshes, finger keypoint masks, and camera poses to estimate the typing surface normal and per finger press-down probabilities.

We had initially explored rule-based heuristics. However, their reliance on numerous variables—such as bounding box variation, viewing angle, and keyboard design—limited generalizability. In contrast, the proposed data-driven approach adapts better to real-world variability. For simplicity, we excluded 14 “special” keys (e.g., space, enter, tab, shift) and focused on recovering 45 alphabetic, numeric, and symbol keys.

Per-Key Recovery Model from Key Combinations

We trained a set of small models to recover the location of a specific key based on the location of detected keys. This resulted in 45 lightweight models, each taking as input the bounding-box coordinates of $x = 5$ high-confident keys predicted by the YOLO key detection model. This approach significantly improved key recovery performance, although each model is run once for every occluded, low-confident, or undetected key. We implemented a simple 4-layer fully connected network architecture for the Key Recovery models to support real-time performance. The number of recovery keys used to locate an occluded key is a hyperparameter we tuned during training experiments and discovered that the key recovery models generally performed better with a higher number of recovery keys. However, setting a high requirement for the number of recovery-keys would make this approach unusable if the key detection model often fails the detect that many keys when the hands naturally occlude significant portions of the keyboard while typing. We found that five keys are adequate for accurate key recovery and few enough for practical use.

Choosing the Combination of Recovery-Keys at Inference

The main challenge is not only learning the regressors, but also choosing which five visible keys to feed them at inference time. We found that some five-key combinations yield much more accurate recovery than others. Exhaustively searching all possible five-key subsets is too expensive, so we pre-rank combinations of five and organize them into a 3–2 key selection procedure. For

each target key, the ranked five-key combinations are decomposed into one main lookup table over the best available three-key subsets and conditional secondary lookup tables over the final two keys. At inference, we first choose the highest-ranked three-key subset present among the confident detections, then select the best compatible two-key completion. This preserves the accuracy benefits of optimized recovery-key selection while reducing search overhead enough for real-time use—effectively reducing the search space from over 1.2M to 15k.

3D hand reconstruction and keypress detection

Finger motion is analyzed from monocular video using HaMeR [16], which reconstructs the 3D hand pose and MANO mesh for each visible hand. The resulting representation provides 21 hand keypoints and a dense mesh per frame, supplying both fingertip trajectories and richer geometric context around finger-surface interaction.

On top of these reconstructions, we train an autoregressive transformer-based keypress detector illustrated in Fig. 4. The model processes temporal windows of RGB frames, recovered 3D hand meshes, finger keypoint masks derived from hand pose confidence maps, and camera poses to estimate the typing surface normal and per-finger press-down probabilities. This formulation is designed to handle variations in typing style and speed, hand shapes, camera viewpoint, and surface orientation. Training and evaluation use typing videos from the MSU Typing Behavior Database [22, 21]. 110 videos are used for training and validation and 30 videos are reserved for test.

Once a press-down event is detected for a finger, the system assigns the event to a key using the recovered keyboard geometry (2D image locations of keys) and the predicted finger-tip location (from 2D hand pose estimator) at the event time. The combined output is a keystroke log with both the event timestamp and the most likely key identity.

Visible-key detection results on the KKD dataset. The table compares models trained with or without synthetic hand occlusion and with axis-aligned or oriented key boxes. The best performance is obtained with oriented boxes and occlusion-aware training.

YOLO Key Detection	Bounding Box		Avg. IOU ↑	mAP		
	Axis-Aligned	Oriented		@50 ↑	@75 ↑	@50-95 ↑
Trained w/o occlusion → no-occlusion test	✓		0.9068	99.7%	97.6%	93.9%
Trained with occlusion → no-occlusion test	✓		0.9129	99.6%	98.2%	94.5%
Trained with occlusion → no-occlusion test		✓	0.9334	99.9%	99.3%	96.7%
Trained w/o occlusion → occlusion test	✓		0.7292	79.6%	75.9%	71.4%
Trained with occlusion → occlusion test	✓		0.8892	97.9%	95.0%	93.3%
Trained with occlusion → occlusion test		✓	0.9078	99.1%	97.8%	95.9%

Results

We evaluated the key detection accuracy on **average intersection over union (IOU)** between the ground truth and predicted bounding boxes of each key in the KKD test set. We use the same IOU metric in Eq. (1) to evaluate the precision of recovered occluded keys bounding boxes. Where A' and A are the area enclosed by the ground truth and predicted bounding boxes, respectively, given the set of I images and K detected keys per image.

$$IOU = \frac{1}{I \times K} \sum_i \sum_k \frac{A'_{ik} \cap A_{ik}}{A'_{ik} \cup A_{ik}} \quad (1)$$

The key detection model outputs the bounding box coordinates, key identity, and detection confidence score for expected keys. Typically, at inference, a preset detection confidence threshold dictates whether a predicted output is taken as a valid detection or ignored. We evaluated the recall of key detections at different confidence thresholds to study how well our models detect keys that should be found. We also assess the **mean average precision (mAP)** of key detections at different IOU thresholds. This is a popular object detection metric that measures the average precision and recall values of all boxes that are detected at a given IOU threshold, averaged over all key classes. For example, mAP@75 computes the mean precision and recall of keys that are detected with an IOU of 75% or more.

Visible-key Detection

The KKD experiments in Table 1 show two consistent trends. First, oriented boxes improve key localization quality over axis-aligned boxes. On the occlusion test split, the oriented-box detector reaches 97.8% mAP@75 and 0.9078 average IoU, outperforming the axis-aligned counterpart. Second, training with synthetic hand occlusion materially improves robustness on occluded inputs without harming clean-input performance. These results justify the detector design used throughout the pipeline. The table also shows that key detection is significantly more difficult when more keys on the keyboard are obscured from the camera view. Not only do the average IOU of the detected bounding boxes drop from 0.933 to 0.908, but also the detection accuracy of mAP@50, mAP@75 and mAP@50-95 drops by .8%, 1.5%, and .8%, respectively. Evidently, training the key detection model with synthesized hand-occluded images improves model performs on both images with hand occlusions and images without occlusions.

Occluded-key Recovery

Recovery is needed because some keys are fully hidden at the exact moments that matter most for keystroke tracking. In our current configuration, the per-key recovery stage sustains 30 fps on average while preserving the geometric regularity needed to reconstruct a complete keyboard layout from the visible subset. The 3–2 key selection procedure is central to that operating point: it retains the benefit of ranked recovery-key combinations while avoiding the cost of exhaustive five-key search. In practice, this makes recovery fast enough for real-time use rather than a post-processing step. In Fig. 5, we compare the speed of key recovery for random five-key selection, 3–2 key selection, and exhaustive five-key search.

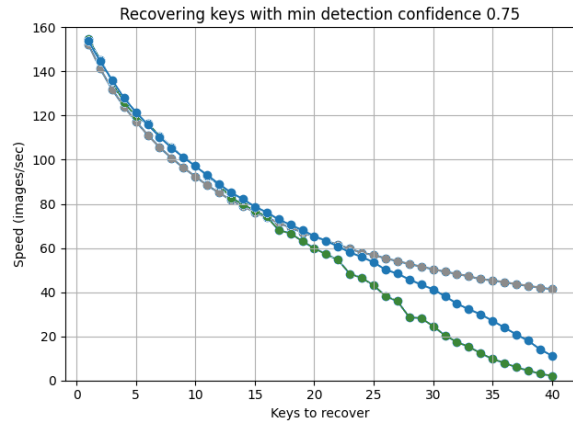


Figure 5. Average key recovery speed at inference. The line plots represent the frame rate of random selection of 5 recovery-keys (gray), the optimized 3–2 key search selection (blue), and the exhaustive five-key search (green).

Figure 6 presents the recovered keys' IOU precision of the different recovery-key selection methods evaluated at three detection confidence thresholds. The most accurate bounding box predictions are obtained when the best available set of five keys, detected with a confidence score of 0.9 or more, are used to recover occluded keys. However, the optimal set of five keys is selected using slow inefficient brute-force search. In a typical typing scenario, the hands can occlude 20–30 keys at any given time. Hence, key recovery with exhaustive best-available five-key search would run at 20 FPS or more. Although this produces

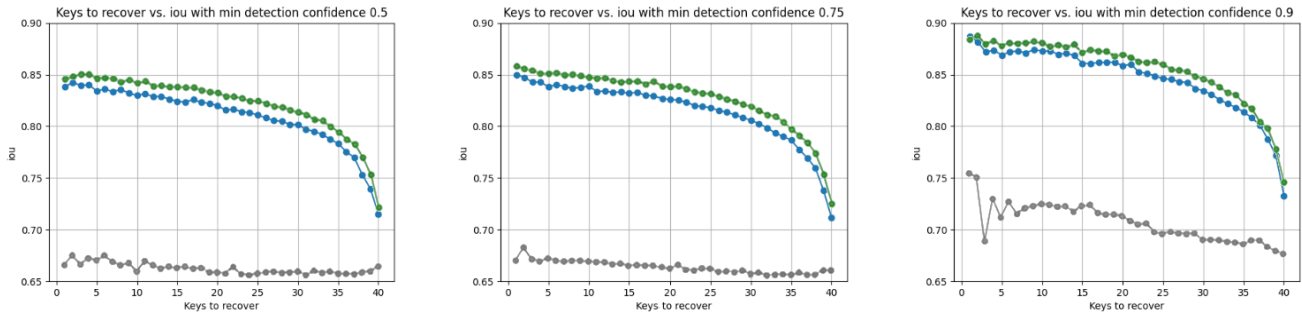


Figure 6. Key recovery on KKD test set evaluated on mean IOU. The line plots represent the outcome of the random selection of 5 recovery-keys (in gray), the optimized 3-2 keys selection (in blue), and the optimal search selection of 5 best available recovery-keys (in green). The set of recovery-keys initially detected by YOLO are determined by a minimum detection confidence of 0.5, 0.75, and 0.9, from left to right.

most accurate key locations, it does not run in real-time when 28 or more keys are to be recovered. However, the optimized 3-2 recovery key search using two lookup tables can recover 20-30+ keys in real-time (60-30 FPS in Fig. 5). Although this comes at a slight drop-off in bounding-box IOU precision. Since the YOLO key detection model run at a frame rate of approximately 200 FPS, the speed of key recovery and finger tracking determines whether the end-to-end pipeline runs in real-time.

Keypress Event Detection.

On videos from the MSU Typing Behavior Database, the keypress module reaches up to 90% event detection accuracy with 28 ms latency. These results are important because accurate key geometry alone is not sufficient for keystroke logging; the system must also determine *when* a finger actually presses. The transformer-based key press-down event detector benefits from the temporal context of image sequences and the geometric stability provided by monocular 3D hand reconstruction. Combined with the recovered keyboard layout, this enables end-to-end finger-on-key event inference using monocular vision alone.

Discussion. The revised pipeline reflects a deliberate decomposition of visual keystroke tracking into geometric layout reasoning and temporal motion reasoning. Visible-key detection and occluded-key recovery solve a structured localization problem, while the press detector solves a temporal state-change problem. This separation is useful because the two tasks fail for different reasons. Key detection is most sensitive to heavy occlusion, blur, and annotation quality. Press detection is more sensitive to reconstruction noise, fast motion, and changes in viewpoint or typing style. Our experiments suggest that the system becomes notably more stable when these two sources of uncertainty are handled by dedicated modules rather than by a single monolithic predictor.

The present paper also has clear limitations. We report keypress event detection accuracy and latency, but not the full per-finger or per-layout breakdown which we reserve for a longer article version. Also, the current recovery formulation targets a fixed key set limited to 45 keys. Future work would aim to improve the five recovery-key selection algorithm to obtain optimal bounding-box precision while guaranteeing real-time performance for worse-case scenarios where 40 or more undetected keys must be recovered. However, note that the frame rates in Fig. 5 are obtained by performing sequential key recovery and a

3X speed-up can be easily obtained by implementing concurrent execution. Even so, the combined results presented in this paper already support the main claim of this work: a camera-only system can recover a full keyboard state and detect press events accurately enough to make practical surface typing possible.

Conclusion

We presented a camera-only keystroke tracking pipeline for surface typing that unifies visible-key detection, hand-occluded key recovery, and finger press-down detection. The system uses oriented bounding-box YOLO detection trained with synthetic hand occlusions, per-key recovery regressors driven by an efficient 3-2 recovery-keys lookup strategy, and a transformer-based finger press-down detector over monocular hand reconstructions for key press event detection. On Keyboard Key Detection dataset, the visible-key detector achieves 97% mAP@75 with mean IoU near 0.91, while the recovery stage runs at 30 fps. On MSU typing videos, the keypress module reaches up to 90% keypress detection accuracy with 28 ms latency. These results show that end-to-end vision-only keystroke tracking is feasible for printed, projected, and rendered keyboards, and they move text entry in extended reality applications closer to hardware-free typing on arbitrary surfaces.

References

- [1] Seyed Amir Ahmad Didehkhorsid, Siju Philip, Elaheh Samimi, and Robert J. Teather, Text Input in Virtual Reality Using a Tracked Drawing Tablet, *Proc. Interacción* (2020).
- [2] Xueshi Lu, Difeng Yu, Hai-Ning Liang, and Jorge Gonçalves, iText: Hands-free Text Entry on an Imaginary Keyboard for Augmented Reality Systems, *Proc. UIST* (2021).
- [3] Mark McGill, Roderick Murray-Smith, Daniel Boland, and Stephen A. Brewster, A Dose of Reality: Overcoming Usability Challenges in VR Head-Mounted Displays, *Proc. CHI EA*, pg. 177 (2015).
- [4] James Walker, Bocho Li, Keith Vertanen, and Scott Kuhl, Efficient Typing on a Visually Occluded Physical Keyboard, *Proc. CHI*, pp. 5457–5461 (2017).
- [5] Jens Grubert, Lukas Witzani, Eyal Ofek, Michel Pahud, Matthias Kranz, and Per Ola Kristensson, Text Entry in Immersive Head-Mounted Display-Based Virtual Reality Using Standard Keyboards, *Proc. IEEE VR*, pp. 159–166 (2018).
- [6] Hao Zhang, Yafeng Yin, Lei Xie, and Sanglu Lu, AirTyping: a mid-air typing scheme based on leap motion, *Proc. UbiComp/ISWC Adjunct*, pp. 168–171 (2020).
- [7] Christoph Amma, Dirk Gehrig, and Tanja Schultz, Airwriting recognition using wearable motion sensors, *Proc. AH, Art. 10* (2010).
- [8] Xin Yi, Chun Yu, M. Zhang, Sida Gao, Ke Sun, and Yuanchun Shi, ATK: Enabling Ten-Finger Freehand Typing in Air Based on 3D Hand Tracking Data, *Proc. UIST* (2015).
- [9] Erez Posner, Nick Starzicki, and Eyal Katz, A single camera based floating virtual keyboard with improved touch detection, *Proc. IEEE EE in Israel*, pp. 1–5 (2012).
- [10] Chris Harrison, Julia Schwarz, and Scott E. Hudson, TapSense: enhancing finger interaction on touch surfaces, *Proc. UIST*, pp. 627–636 (2011).
- [11] Manuel Meier, Paul Streli, Andreas Fender, and Christian Holz, TapID: Rapid Touch Interaction in Virtual Reality using Wearable Sensing, *Proc. IEEE VR*, pp. 519–528 (2021).
- [12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick, Mask R-CNN, *Proc. ICCV*, pp. 2980–2988 (2017).
- [13] Hongyu Xiao, Hui He, Yifan Xie, and Yi Zheng, Occluded human pose estimation based on part-aware discrete diffusion priors, *Knowl.-Based Syst.*, 315, 113272 (2025).
- [14] Mallika Garg, Debashis Ghosh, and Pyari Mohan Pradhan, OccRobNet: Occlusion Robust Network for Accurate 3D Interacting Hand-Object Pose Estimation, *arXiv* (2025).
- [15] Joo Hyun Park, Yeong Min Oh, Gyeongsik Moon, Hongsuk Choi, and Kyoung Mu Lee, HandOccNet: Occlusion-Robust 3D Hand Mesh Estimation Network, *Proc. CVPR*, pp. 1486–1495 (2022).
- [16] Georgios Pavlakos, Dandan Shan, Ilija Radosavovic, Angjoo Kanazawa, David Fouhey, and Jitendra Malik, Reconstructing Hands in 3D with Transformers, *Proc. CVPR* (2024).
- [17] Vahid Rezanezhad, Konstantin Baierer, Mike Gerber, Kai Labusch, and Clemens Neudecker, Document Layout Analysis with Deep Learning and Heuristics, *Proc. HIP*, pp. 73–78 (2023).
- [18] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi, You Only Look Once: Unified, Real-Time Object Detection, *Proc. CVPR*, pp. 779–788 (2016).
- [19] Xuerui Zhang, A lightweight model FDM-YOLO for small target improvement based on YOLOv8, *arXiv*, abs/2503.04452 (2025).
- [20] Kaggle Datasets and Farmer Matzle, Keyboard Key Detection Dataset, *Kaggle dataset* (2024).
- [21] Joseph Roth, Xiaoming Liu, Arun Ross, and Dimitris Metaxas, Investigating the Discriminative Power of Keystroke Sound, *IEEE Trans. Inf. Forensics Secur.*, 10(2), 333–345 (2015).
- [22] Joseph Roth, Xiaoming Liu, and Dimitris Metaxas, On Continuous User Authentication via Typing Behavior, *IEEE Trans. Image Process.*, 23(10), 4611–4624 (2014).
- [23] Richard O. Duda and Peter E. Hart, Use of the Hough transformation to detect lines and curves in pictures, *Commun. ACM*, 15(1), 11–15 (1972).
- [24] Qi Han, Kai Zhao, Jun Xu, and Ming-Ming Cheng, Deep Hough Transform for Semantic Line Detection, *Proc. ECCV*, pp. 249–265 (2020).
- [25] Bhakti Baheti, Shubham Innani, Suhas S. Gajre, and Sanjay N. Talbar, Eff-UNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment, *Proc. CVPRW*, pp. 1473–1481 (2020).
- [26] Moritz Kassner, William Patera, and Andreas Bulling, Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction, *arXiv* (2014).

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



- **SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •**
- **INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •**

www.electronicimaging.org

