Through-Screen Finger Localization and Tracking using Reflected Light

Andrei Grigore Croitoru[1]

Supervisor: Dr. Qing Wang

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Andrei Grigore Croitoru
Final project course: CSE3000 Research Project
Thesis committee: Dr. Qing Wang, Braden Refalo, Dr. Julia Olkhovskaia

Abstract

Visible light positioning systems conventionally fix the light sources on a ceiling and let a receiver move through the scene. We invert this geometry by tracking a hovering finger above a transparent segmented OLED display placed above four photodiodes. From the four signals influenced by the reflected light from the finger we track its position. The motivating application is pre-touch sensing on mobile devices, where anticipating the user's next touch during the hover to touch window lets the system pre-load content. The central question is whether four under-screen photodiodes can localize and track a hovering finger in real time using only the microcontroller already driving the screen, avoiding the deep neural networks that prior through-screen sensing required. We collected a reflected light dataset of 199 finger captures across a 10×4 calibration grid and evaluated localization on a 5×2 cell grid. After subtracting a temporally interpolated no finger baseline, we build an 18-dimensional feature vector and classify the cell with a two-stage logistic-regression head that predicts column and row independently. This reaches 77.2% cell accuracy under a random split and 66.8% under a leave-one-calibration-dot-out protocol. The second one is more representative of deployment because every recording of the tested position is withheld from training. The complete pipeline runs on an Arduino Due that also drives the screen with sub-millisecond inference. We conclude that through-screen reflected light carries enough spatial information for cell-level finger localization without deep learning, on the same embedded hardware that runs the display.

1 Introduction

Many works show that visible light is a very promising medium for localization, monitoring and tracking in different indoor environments [1]. Sun et al. [2] show the high potential for accuracy that tracking in the visible light medium can provide. In that particular work the setup revolves around tracking a robot with a photoreceptor on top of it and five LEDs on the ceiling in a $5 \times 5 \times 2.84$ m room achieving mean errors ranging from 26 cm to 40 cm on various robot trajectories. One reason for the emergence of visible light as a tracking medium is its advantage over RF media of devices not competing for the same bandwidth [1].

Most current tracking and positioning using visible light (VLP) share a common geometry where light sources are fixed and with known positions on the ceiling or panels and a receiver or a disturbing object moves through the scene as presented in the Figure 1 of [3]. In this work we invert that geometry. We present a method that localizes the position of a finger hovering above a segmented OLED HUD using the reflected light from the screen onto four under-screen photodiode sensors. We envision deploying this technology in mobile phones to help predict where the user is pointing the finger with the intent to press on the screen. The main advantage of this configuration lies in the timing window between hover and touch. This window in which the system can predict the user's intentions can have various applications such as allowing the operating system to pre-load applications or pre-fetch the contents of links.

Pre-touch technologies, systems in which we determine the position of the finger before it touches the surface, already show great value. Hinckley et al. use self-capacitance hover sensing to anticipate the user's input and adapt the content on display before the touch lands [4]. Nonetheless, the self-capacitance hover panels their discussion relies on are not usually present in mobile phones, which motivates capturing the hover window by other means.

One such means is using optical sensing. Commercial touchless interfaces like Leap Motion [5] are dedicated hardware that uses near-infrared (NIR) cameras to achieve full hand tracking. Kamijo et al. [6] use a through-display sensing technology, where a custom made near invisible organic photodetector array is placed over a screen and used to sense motions of a hovering finger. Compared to them, we only rely on a small set of low cost photodiodes below the screen.

Zhang et al. present a clever way to localize and track the position of a finger gliding on a flat surface [7] using reflected light and ordinary components. They obtained a median finger localization accuracy of 0.70 cm, with a 90th-percentile error of 1.43 cm on a $7 \times 9\ cm$ surface. In a calibration step they model and store the received signal strength their photodiodes should read for every millimeter of the surface, and compare it with the real-time readings. The setup must be recalibrated for every new user or surface and their accuracy depends on dedicated hardware, composed of a precise custom built enclosure for an LED and two photodiodes. Moreover, their model assumes a point like light source and it is unclear how it would transfer to a more diffuse, larger light source like our OLED screen. We therefore do not reuse their physical model and instead ask whether a lightweight learned pipeline can recover the finger's position in our setting.

Closer to our setup, prior work has demonstrated that through-screen reflected-light sensing with a transparent OLED and a photodiode array carries sufficient spatial information to classify air written digits with up to 91% accuracy [8]. Although the classification accuracy is impressive the same paper shows the highest accuracy is obtained when the finger hovers 1 cm above the screen, dropping sharply as the distance increases. Furthermore, their pipeline relies on a deep neural network deployed on a Raspberry Pi 4 which is by no means a lightweight process.

It remains an open question whether or not the same geometry carries enough information to track a hovering finger over the screen's surface without the use of heavy deep-learning models. Can a lightweight method recover usable finger position from the reflected light and can all computation, from capture to real time tracking of the finger's position, happen on the same microcontroller managing the screen?

The open questions above motivate the main research question and three sub-research questions.

Main research question. *Can four under-screen photodiodes localize and track a hovering finger above a segmented OLED display in real time, using only the computation strength of the embedded microcontroller driving the screen?*

1. To what extent do preprocessing techniques and the feature engineering pipeline drive localization accuracy relative to the choice of classification model?
2. How does the localization accuracy obtained from the static capture translate to continuous tracking when restricting the capture window to reduce the latency of the tracking?
3. Does a model accurate enough to support finger localization fit the SRAM, flash and real time inference budget of an Arduino microcontroller?

This paper makes four contributions. First, we collect a reflected light dataset of hovering stationary finger positions above the OLED display across forty calibration positions on a 10×4 grid, evaluated on a 5×2 cell grid. Second, we identify a signal preprocessing and feature engineering pipeline that generates most of the accuracy for the static localization task. Third, we develop a two stage classification head, separating cell classification into row and column categorization, reaching 77.2% cell accuracy under a random split evaluation and a 66.8 % accuracy under a strict leave-one-calibration-dot-out (LODO) cross validation on the 5×2 cell grid. Fourth, we port the classifier to an Arduino Due and evaluate its continuous tracking accuracy under shorter data capture windows.

2 Methodology

In order to answer the main research question we first have to understand the setup and its particularities and the choices we made along the way to define the chosen method. The first particularity regarding the setup emerges from the fact that the transparent screen allows light to bleed onto the photodiodes below. In order to extract data about the finger, the light has to travel from the screen to the finger and then back to the photodiodes under the screen. Since the finger is not a perfectly reflective surface, part of the light intensity is lost. On top of that, the intensity of the light is inversely proportional to the square of the distance between the source and the receiver. We consistently observed that the contribution of the finger to the overall received signal is small compared to the light bleeding from the underside of the screen. In this section we describe what the setup looks like, what data collection protocol we used and how we managed to extract the most information about the finger position despite its relatively small contribution.

2.1 System Design

The sensing setup is composed of three components layered on top of each other. On top there is a SparkFun WiseChip OLED HUD display [9] measuring 10×3 cm, from which light is emitted on a hovering finger above. Below the screen there are four OPT101 photodiode sensors [10] placed at approximate (x, y) positions $(2.0, 0.5)$, $(7.8, 0.5)$, $(2.0, 2.8)$ and $(7.8, 2.8)$ cm relative to the top left corner of the screen, as presented in Figure 1. The values of the photodiode array are read simultaneously and converted into readings by the Arduino Due's [11] 12 bit on chip ADC, our third layer of the system. As mentioned above all predictions for finger position are computed on the Arduino Due which has 96 KB of SRAM and 512 KB of flash memory. This computation limitation is intentional and mirrors the ambition of deploying these localization and tracking methods on a generic mobile chip alongside the processes already running.

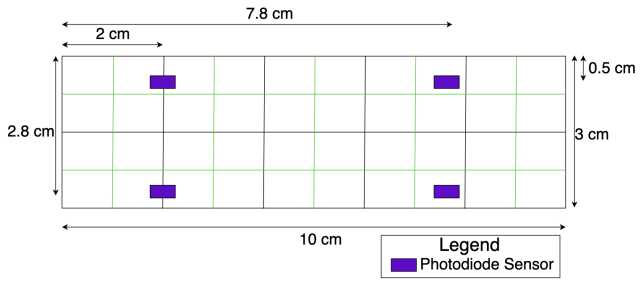

Figure 1: Schematic showing the placement of the photodiode sensors relative to the top left corner of the screen. With black outline we see the 5×2 evaluation grid and with green outline the 10×4 data collection grid

Before gathering meaningful data using this setup we had to find a configuration of the segmented HUD where there is no lit segment right above the photodiodes. If we didn't do so the screen bleed would dominate the finger related signal making the finger's contribution too weak to enable reliable localization. We turned off all the segments right above the photodiodes from the firmware, keeping the rest turned on to maximize the total light reaching the finger.

With this setup we collected a dataset of 199 finger captures across 40 calibration positions arranged on a 10×4 grid at $1\,\mathrm{cm}$ spacing across the screen surface as presented in Figure 1. Each calibration position was visited approximately five times. Ten no-finger baseline captures were interleaved with the finger captures throughout the collection to support the temporal baseline interpolation procedure described in Section 2.3. The data was collected across three separate sessions, each lasting between 25 and 35 minutes. The 10×4 calibration grid is four times denser than the 5×2 evaluation grid helping us gather up to 20 independent captures for each one of the ten evaluation cells. Throughout the session all captures gathered information about the finger hovering between 15 and 20 mm above the screen.

2.2 Signal Acquisition

Because the actual information about the finger is small, we had to find a way to make sure the noise from the screen's interference does not affect the finger's contribution to the signal even more. We looked at the signal's signature presented in Figure 2a and we can see that 80 of 1000 samples drop more than 100 ADC below the median in a representative 1-second capture. The corresponding magnitude spectrum (Figure 2b) is approximately broadband with no dominant periodic carrier. We therefore apply a per-capture median pooling rather than computing the mean, which is much more sus-

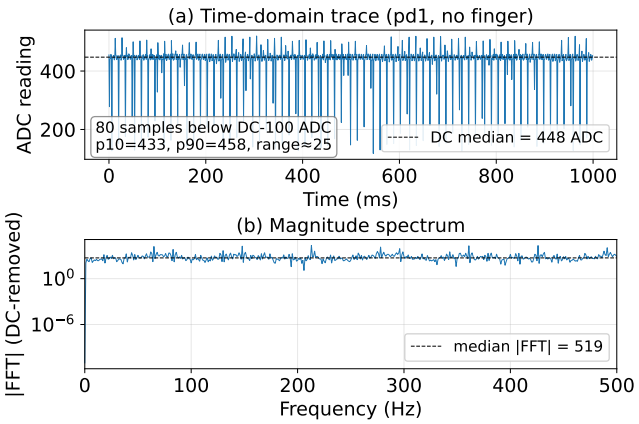

Figure 2: Photodiode signal at rest (no finger). (a) Time-domain trace of channel PD1 over a single 1-second capture showing a stable DC level despite the presence of large impulsive transients, (b) Magnitude spectrum (DC removed); the median spectral magnitude is shown as a reference. The spectrum is approximately broadband with no dominant periodic carrier, consistent with impulse noise rather than narrowband flicker or a constant PWM carrier.

ceptible to the large irregular drops in received strength. We sampled the photodiodes at 1 kHz for 1 second, thus collecting 1000 samples per median, sufficient to establish a stable estimate of the DC baseline. This sampling regime is used throughout for all subsequent captures.

2.3 Extracting the finger's contribution

In order to separate the finger's contribution from the light bleeding in from the screen, we defined the following process. In the same environment where we want to establish the position of the finger we read data from the sensors and, as mentioned before, extract its median value. We perform this reading five times and define the *baseline value* as the median of those five values. This procedure allows us to confidently characterize the value received by each of the photodiodes in the absence of the finger. A similar procedure is described by Deprez et al. [12], where they extracted the contribution of a human's presence in a setting by first measuring the intensity of the light with no human in the scene.

In the same way, the value characterizing the environment is subtracted from the median readings, $P_{\text{raw},i}$, for each of the photodiodes when the finger is introduced in the system, thus resulting in the raw contribution of the finger. We also observed that the baseline value can drift throughout a longer data-collection session and that it is essential to gather multiple baseline values throughout the session (Figure 3). Likely reasons for changes in baseline values are the temperature of the screen and changes in ambient lighting. In total we recorded 209 captures: 199 finger captures and 10 no-finger baseline values. We took 10 equally spaced baseline breaks, one at the start of the session, one every 20 finger captures, and one at the end. At each break we recorded five no finger readings whose median defines that break's baseline value. We then subtract the linearly interpolated baseline from each of the 199 finger captures.

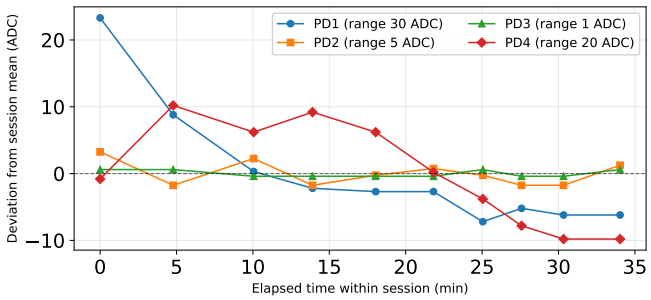

Figure 3: Baseline values and drift in a data-collection session. Drift of the baseline values throughout a session expressed as the difference from their per-PD mean value.

2.4 Feature Design

In order to further extract information about the position of the finger, we tried multiple feature representations. From the baseline-corrected per-PD signal vector $\mathbf{P}_{\text{corrected}} \in \mathbb{R}^4$, introduced in the previous section, we construct a higher-dimensional feature vector that combines four complementary representations: raw values, polynomial expansion, pairwise log-ratios, and a relative-intensity scheme. All the different feature representations tested are composed of one or more of these four components concatenated. Each of them targets a specific signal property: magnitude, polynomial separability, drift-canceling ratios, and common-mode-canceling normalization. When all four feature representations are concatenated we reach a total of 18 dimensions.

The most direct features are the baseline-corrected values, one for each of the PDs. Under the Lambertian VLC channel model [13], the optical power received at a photodiode is a deterministic function of the source-to-PD geometry. The intensity of the received signal is inversely proportional to the square of the distance between the source and the receiver, and is shaped by the emission and incidence angles as

$$P_r \propto \cos^m(\phi)\cos(\psi)/d^2, \tag{1}$$

where d, ϕ, and ψ are the source-to-PD distance and the emission and incidence angles, respectively. In our setup, each of the four photodiodes sits at a fixed corner of the sensing area, so a finger at a given position above the screen induces a different (d_i, ϕ_i, ψ_i) tuple at each PD and therefore a different received-power component $P_{r,i}$. The 4-vector $\mathbf{P}_{\text{corrected}}$ is thus a position-dependent spatial signature, motivating its inclusion as a feature.

Because the relation between the finger's position and the received signal strength (RSS) is not linear, a purely linear classifier will lead to misclassifications not aligned with the true finger's position. This is the reason why we append the squared values of $\mathbf{P}_{\text{corrected}}$, thus extending the total feature vector to 8 dimensions. The classifier remains linear in the extended feature space, while allowing us to fit curved boundaries in the original received-intensity space.

The baseline subtraction presented in section 2.3 eliminates the OLED bleed onto the PDs, and the contribution of the finger dominates the corrected signal. This signal does not only contain information about the distance and angle

from the finger to each of the PDs. It also contains information about the reflective nature of the finger, how well the light from the screen reflects off it, and small common-mode ambient-light shifts. Since these factors are sensed by all photodiodes in similar ways, the previous two features described will not help with mitigating them. To further extract information about the finger's position we decided to combine all four $P_{\text{raw},i}$ values, creating $\binom{4}{2} = 6$ new features. Raes et al. state that this feature design approach increases the robustness of RSS-based visible light positioning applications [14]. We also wanted to see if it can improve accuracy. In our feature design methodology we adopted log-differences $\log P_i - \log P_j$, choosing logarithms over raw ratios for numerical stability when one PD reads near zero.

On top of the log-ratio features helping us to further extract information about the finger's position, we also tried other relative-intensity representations with varying degrees of success. The sum-normalized representation

$$pn_i = P_{\text{raw},i} / \sum_j P_j \tag{2}$$

maps each capture's four-vector to the $(0, 1)$ interval, with the four components summing to 1. This is the same scale-invariance the log-ratios provide, presented in a different geometric form. As with the log-ratios, this features only cancels effects that scale all four photodiodes uniformly.

Besides adding to the set of features used as input for our model, we wanted to try preprocessing techniques as well. When consulting the VLP literature, we found that Sigmoid Function Data Preprocessing (SFDP) was presented by Wu et al. [15] as a way to increase accuracy when paired with kernel ridge regression. SFDP standardizes each feature on the training fold and applies the logistic squash.

$$f(z) = \frac{1}{1 + e^{-b\frac{z-\mu}{\sigma}}} \tag{3}$$

We managed to replicate their findings, but we also observed that adding SFDP to any feature representation and feeding the results as inputs for any other model than kernel ridge regression (KRR) and the RBF Support Vector Classifier (SVC) produced no improvement.

2.5 Model Design

When trying to find the best way to localize the finger position, we first trained a continuous regression on the (x, y) labels. The continuous predictions were then snapped to the 5×2 evaluation grid in order to identify whether or not the model predicted the correct cell, our primary success metric. Results hovered around the 64% cell-accuracy mark, which fell short of our expectations. We then computed the per-axis confusion matrices and found that the dominant confusions were along the X-axis (Figure 4).

To further increase the accuracy on the X-axis we changed our approach from snapping the results of regression models to the evaluation grid to extracting the column and row position from the features separately. We introduced a 5-way classifier for the column and a 2-way classifier for the row, recombining them at inference as cell = row $\times$ 5 + col. With this new approach we achieved a 77.2% cell-accuracy

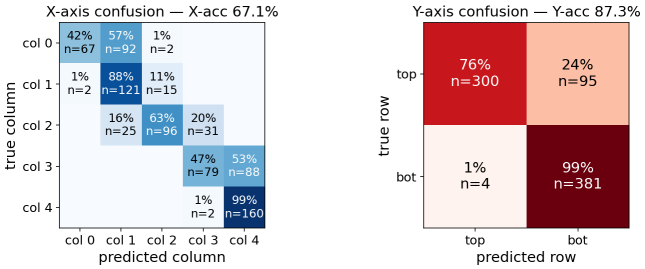

Figure 4: Per-axis confusion matrices for the continuous-regression-then-snap pipeline, for the best performing model. The X-axis (column index) is the cell-accuracy bottleneck: regressor collapses left-edge predictions toward the center of the photodiode array, leaving the Y-axis (row index) nearly solved.

result. We opted for this approach as opposed to a 10-way classifier since the per axis confusion is largely decoupled so a joint 10-way classifier would thus suffer twice the confusion. For this two stage classification approach we then tested five classifiers, namely logistic regression, random forest classifier, Extreme Learning Machine (ELM), k-NN and SVC. The results for all five families of classifiers are presented in Table 3. The best performing family, logistic regression, will be described here. All models were implemented using scikit-learn 1.8.0. Each stage is a separate `sklearn.linear_model.LogisticRegression` estimator with `max_iter=2000` and `class_weight="balanced"`. The inverse-regularization strength C is the only tuned hyperparameter selected by grid search over $C \in \{10^{-3}, 10^{-2.5}, \ldots, 10^{3}\}$, with a 5-fold inner cross-validation loop inside `GridSearchCV` using per stage classification accuracy as the selection criterion. The fold splitter is matched to each evaluation regime and the estimator is refit on the full training fold at the best C.

3 Evaluation

In this section we define and motivate the success metrics for finger localization and tracking respectively. We will explain the evaluation methodology of both the localization and tracking objectives and we will present the headline results, comparing them to different model and feature representation variants.

On top of that we will analyze which cells are most commonly misclassified and why, we will talk about the robustness of the setup with regards to finger hovering height and capture window and in the end about the deployment of the models on the device.

3.1 Evaluation Methodology

All feature representation options presented above were given as input to various regressor and classifier models in order to identify which combination of model and feature representation could most consistently identify the finger's position.

We define the primary success metric as the cell accuracy:

$$\text{cell accuracy} = \frac{1}{|S|} \sum_{i \in S} \mathbb{1}[\hat{c}_i = c_i], \tag{4}$$

where S is the set of evaluation samples, $\hat{c}_i$ is the cell predicted by the model for sample i, c_i is the ground-truth cell, and $\mathbb{1}[\cdot]$ is the indicator function that returns 1 when its argument is true and 0 otherwise.

The (model, feature representation) pairs were evaluated under three complementary regimes that together cover the deployment scenario. The first regime is a random $80/20$ cross-validation across 20 seeds. Hyperparameter selection within each training fold was done using a 5-fold unshuffled inner loop on the 80% training set. Because the calibration grid (with a 10×4 resolution) is four times denser than the evaluation cell grid mentioned above, the test dots tend to land close to training dots, simulating a low difficulty scenario. Furthermore, almost every held-out capture has other visits of the very same dot in the training set. A model can therefore score well partly by recognizing a location's exact feature signature rather than by generalizing across space.

The second regime is a leave-one-calibration-dot-out (LODO) cross-validation with 40 outer folds which holds out a single calibration dot at once. Each outer fold removes all 5 visits of one calibration dot at once, so that position is entirely absent from training. C is tuned on the remaining 39 dots by a 5-fold `GroupKFold` inner loop grouped by calibration dot, so no other visit of the held-out dot can leak into tuning either. Since all captures for a single dot are removed from the training data, when the prediction is correct we know that it measures spatial generalization rather than visit-level overlap.

The third regime evaluates on-device continuous tracking. We treat tracking as a continuous re-localization, running the static classifier after capturing data from all PDs. Since when tracking we first collect data and then we run the inference, the tracking latency is the sum of the capture time and the duration of the inference. We thus evaluate the tracking performance by the cell accuracy as the capture window shrinks (Figure 7) and the per-update latency, described in Section 3.5.

3.2 Headline Results

The (model, feature representation) pair that had the best results across all three evaluation regimes was:

```
(logistic regression, log+raw+sq+pn)
```

We will refer to it as the *production classifier* from now on since it is also deployed on the sensing device. The feature set has 18 dimensions and the two stages of the classifier are trained independently with class-weighted loss. Comparing Figure 4 and Figure 5 we can clearly see an improvement between the two models.

Across 20 random seeds, the production classifier reached a cell accuracy of $77.2\% \pm 7.1\%$, composed of a column-axis accuracy of $81.2\% \pm 5.9\%$ and a row-axis accuracy of $95.9\% \pm 2.2\%$. On the other hand the best results obtained by any regression-then-snap-to-grid pipeline (model, feature representation) pair was:

```
(Kernel Ridge Regression, log+raw+sq+SFDP)
```

and only reached 59.5% cell accuracy under the same protocol. We will refer to it as the *regression standard* from now on.

Table 1: Headline cell-classification results across the continuous regression-then-snap and the production two-stage classifier approaches, and two cross-validation regimes, random 80/20 across 20 seeds and 40-fold LODO.

Approach	Cell acc.
KRR + `log+raw+sq+sfdp` (random)	0.595
KRR + `log+raw+sq+sfdp` (LODO)	0.558
Two-stage logreg + `log+raw+sq+pn` (random)	**0.772**
Two-stage logreg + `log+raw+sq+pn` (LODO)	**0.668**

Table 2: Feature-representation ablation under the production two-stage logreg classifier, random 80/20 cross-validation across 20 seeds.

Feature set	Dim.	Cell	X	Y
raw	4	0.726	0.817	0.908
raw + pn	8	0.724	0.815	0.908
log + raw	10	0.727	0.817	0.909
raw + sq	8	0.762	0.803	0.958
raw + sq + pn	12	0.764	0.804	0.959
log + raw + sq + pn	**18**	**0.772**	**0.812**	**0.959**
log + raw + sq + sfdp + pn	18	0.737	0.785	0.949

Under the 40 fold LODO regime the production classifier reached a pooled cell accuracy of 66.8%, a column-axis accuracy of 71.9%, a row-axis accuracy of 95.0%. The per-dot cell accuracy averaged 67.0% with a standard deviation of 27.0% across the 40 held-out positions, reflecting position dependent spread that we revisit in Section 4. The regression standard under the same LODO protocol reached 55.8% cell accuracy. Table 1 consolidates the headline numbers.

As mentioned before we introduced multiple features to further extract more information about the finger's contribution. The 4-dimensional baseline-corrected raw features already reach a cell accuracy of $72.6\% \pm 9.6\%$, which is within 4.6% of the production classifier. The single largest contribution comes from the squared per-PD features which add $+3.6$ accuracy points ($72.6\% \rightarrow 76.2\%$). The improvement is fully attributable to the row axis, which moves from 90.8% with only the raw features to 95.9% while the column axis remains around 80% across all variants. An interesting negative result comes from the addition of SFDP which manages to reduce the cell accuracy to 73.7%. Table 2 presents the results in greater detail.

3.3 Per-Axis and Per-Cell Matrix

Figure 4 shows that the regression standard has difficulties distinguishing between neighboring columns. The best results are obtained for column 4 with a 99% accuracy and for column 1 with an 88% accuracy, while the worst performing column, column 0, was correctly identified only 42% of the time. The regression standard approach performed much better when identifying the row, managing to correctly find the bottom row 99% of the time and the top row 76% of the time. We will further discuss and explain these results in Section 4.

Figure 5 shows the best performing column is now the third

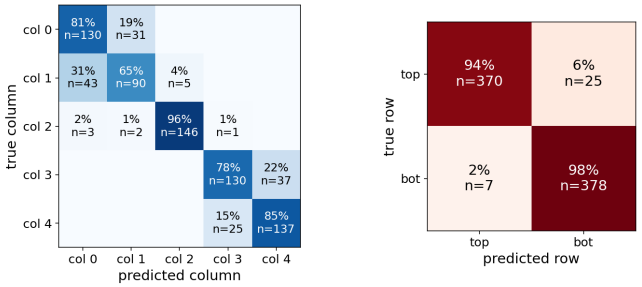

Figure 5: Per-axis confusion matrices for the production classifier.

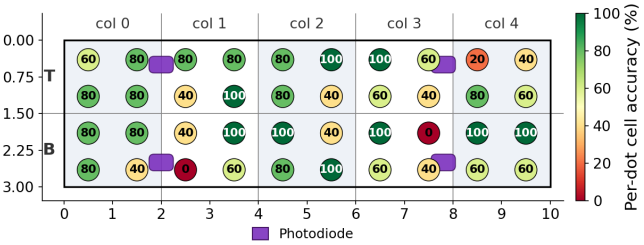

Figure 6: Accuracy of the production classifier in LODO evaluation regime. Each dot was evaluated five times while the other 39 dots were training data.

one with 96% accuracy and while overall there is a solid improvement for the column average accuracy. Not all column accuracies have increased, column 4 for example has decreased from a 99% accuracy to 85%. On the other hand the top-bottom confusion is almost non-existent.

Figure 6 breaks the LODO penalty down by held out dot. The 27% standard deviation mentioned before is structural rather than uniform. 21 of 40 dots show an accuracy of 80% or higher, 9 having perfect accuracy while 15 dots carry almost the entire LODO penalty. The per-dot spread within a single cell can be wide, for example the second and fourth cells on the bottom row show dots spanning the full 0–100% range, while in the other bottom row cells dots start from 40%. Between rows the accuracy varies, the top row having better accuracy in most columns.

3.4 Robustness Analysis

When talking about the robustness of the sensing setup we will examine how changes in factors such as the capture window, how many milliseconds of data are given as input for inference and finger hover height influence cell prediction accuracy.

If we take a look at Figure 7 we can see that for a capture window of 1000 ms the accuracy is identical to the one presented in the paragraphs above and it slowly degrades towards the 58.3% mark on a capture window 20 times smaller than the one the model was trained on. When this model is deployed for continuous tracking the tracking window is directly contributing to the latency of the prediction, since it first gathers the data then based on the capture it predicts the finger's position. As we can see even though the regression standard does not degrade as much with a smaller capture window, it still performs worse across the board. We think

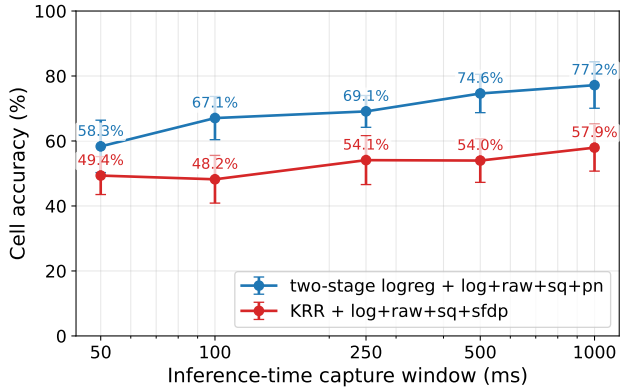

Figure 7: Cell classification accuracy for the production classifier (blue) and the regression standard (red), both trained on 1000 ms captures, as capture window shrinks.

shrinking the capture window yields a noisier per-channel median. A KRR prediction is a kernel-weighted average over training points, which averages out per-feature noise, whereas the logistic head is a weighted sum of the now noisier features and so responds more to it. The production classifier nevertheless retains higher absolute accuracy across the entire range.

When investigating how the accuracy changes with the hovering height we fixed various hovering distances from 10 to 60 mm. After that we visited all 10 evaluation cells three times and calculated how many times the model predicted the correct cell. The results are presented in Figure 8. Since during the capture of the training and evaluation data the finger was hovering around 20 mm the 76.7% accuracy for that height mirrors the headline results. For a hover height smaller than the ideal one, results stay strong at 70% accuracy. For a hovering distance from 30 to 40 mm accuracy drops from 40% to 36.7% respectively. At the 60 mm threshold the accuracy matches the naive approach at 10%. Results degrade as the hovering height increases. Figure 8 also shows that the classification accuracy for rows and columns degrades roughly at the same pace.

3.5 On-Device Deployment

The production model, deployed on the Arduino Due, is a two-stage logistic regression with an 18-dimensional feature vector. We ported all the model parameters and confirmed the firmware fits the device's memory. The parameters are a 5×18 column-head weight matrix plus a 5-element bias vector of 380 B, and a binary row-head with an 18-element weight vector plus a scalar bias (76 B), all stored as 32-bit floats. They are all stored as constants and read from the flash at inference. The per-capture sample buffer occupies $1000 \times 4 \times 2 = 8000$ bytes of SRAM at the 1-second window, and the 18-element float32 feature vector adds a further 72 bytes per inference. They fit comfortably within the memory budget. The remainder of the 15 916 B static footprint totaling to 16.2% of the 96 KB SRAM is the HUD display driver with around 4.2 KB and the Arduino/USB runtime. The complete firmware image

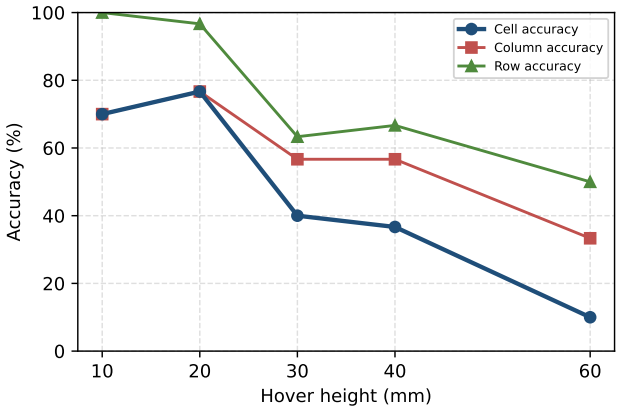

Figure 8: Cell classification accuracy for the production classifier when finger hovering height is changed.

occupies 6.9% of the 512 KB flash.

Another important measure for a successful deployment on the device is latency. Moreover, if we remember the possible applications of this technology, for example loading apps on one's phone from the time they point towards it to the moment they press on it, the lower the inference time the higher the time for the app to load. We collected data across 200 inference cycles and the median latency was 0.85 ms, with a 95th-percentile of 0.89 ms and a maximum of 0.92 ms. This means that latency is two orders of magnitude smaller than a 100 ms capture window.

4 Discussions

As we have pointed out before choosing the right feature representation makes a difference in the performance of the classifier. That being said, in each classifier family the feature representation choices can lead to huge improvements or slight fine tuning. Between family best-accuracy scores range from 63.6% to 77.2% for the production classifier. For the logistic regression classifier, all feature representation variants containing raw values (raw) and squared raw values (sq) features show results above 71% accuracy which shows that the rest of the features help more for fine tuning the results. Another classifier where the feature representations had little impact was the random forest classifier. This happens because internally the tree splits performed by the random forest do their own feature selection so adding and removing inputs generated only a 6% swing in the cell accuracy. Both k-NN and ELM performed better when the features had simpler representations, in fewer dimensions, in the case of k-NN adding too many features and preprocessing techniques stopped the algorithm from finding clusters that were related to actual close finger positions. The extreme 65.8% swing in accuracy for SVC presented in Table 3 only shows the RBF kernel's high sensitivity to input scaling, without SFDP squashing the features, the SVC performs no better than guessing the cell blindly. The largest single lever in the entire static-localization pipeline, however, is neither preprocessing nor model family. It is the architectural decision to predict column and row indices independently rather than snap a continuous (x,y) regression onto the cell grid. The two-stage split improves results by 17 percentage points over the strongest regression-then-snap pipeline, exceeding the gain available from either feature design or model family in isolation.

Table 3: Two-stage classifier-family comparison restricted to feature variants that include the raw baseline-corrected 4-vector, random 80/20 cross-validation across 20 seeds. Each family is shown at its best- and worst-performing variant within this restricted set (best row first).

Classifier	Variant	Accuracy
Logistic regression	`log+raw+sq+pn`	77.2%
	`raw+sfdp`	69.5%
SVC (RBF kernel)	`raw+sq+sfdp`	76.3%
	`raw+sq`	10.5%
Extreme learning machine	`raw+sfdp`	71.9%
	`raw+sq`	59.5%
Random forest	`log+raw+sq`	70.5%
	`raw+sfdp+pn`	64.6%
k-nearest neighbours	`raw+sfdp`	63.6%
	`log+raw+sfdp+pn`	50.6%

4.1 Geometric interpretation of results

The first observation derived from all model and feature representations that we tested is that X axis limitation is structural, not algorithmic. All two stage classifiers showed better performance on Y axis than the X axis since to distinguish between the top and bottom rows it was enough to determine whether the stronger signal came from the top or the bottom photodiodes. For the column classification there are two distinct PD positions and the model had to discriminate between 5 options only 2 cm apart. One other effect of the lack of information on the X axis is shown by Figure 6, where dots near cell boundaries have worse accuracy, since the values are not just physically close, but also close in the feature space.

From the LODO evaluation of the production model we observed that the worst performing dots were the ones closer to the PDs. Near a PD, the received power function is steepest, meaning that a small shift in position has a bigger impact on the received value. This means that a held out dot near a PD will land further in feature space than its surviving neighbors.

4.2 Limitations in the real world

One of the most impactful limitations is the ambient light. Even if capture window and hover height are at ideal parameters ambient light can destroy the performance of the model. The baseline subtraction procedure described in Section 2.3 helps with highlighting the finger's contribution in dim and light controlled environments. It relies on the assumption that the finger contributes positively to the readings. In very well lit environments subtracting the baseline is not enough since the finger will not just have a positive contribution to the readings, due to the reflected light from the screen, but it will also have a negative contribution because it also blocks sur-

rounding light. When the classifier only trained on positive-contribution captures receives these negative contribution inputs it produces unreliable predictions.

One area not explored in this paper, but which might influence real world performance is testing this model with different users. Every person has different fingers which may have different reflectivity and for which results might vary in quality. For the most consistent results the model should be trained on every person's finger. Moreover, if display configurations change the performance of the model does as well. If one decides to deploy this model on a completely different setup results will not hold.

5 Responsible Research

At every milestone of the project we considered responsible research, namely reproducibility of results, whether or not we introduced a bias in our results, integrity and the possible applications of our research. In this section we will explore these concerns and discuss the contribution of Gen AI tools in the creation of the paper.

5.1 Reproducibility

Because the reflected light signal is setup dependent, as mentioned in the limitations section, we don't claim that another VLP setup deploying the production classifier will reach the same accuracy. That being said, we aim for procedural reproducibility, meaning that on the same setup following the same procedure the results are recoverable. We made sure to mention the names of the essential parts like the screen, photodiodes and microcontroller in Section 2.1. On top of this all features, preprocessing and model choices, and evaluation methodology are presented in their respective subsections. Another important mention for reproducibility is that all seeds are generated by `numpy.default_rng(0..19)` and used for the 80/20 split.

5.2 Bias

In terms of bias it is important to mention that all data captures were performed by only a single person, having the same finger width and skin properties. This means that we cannot make cross-user and cross-skin-tone claims. In terms of results the headline figure of 77.2% cell accuracy is biased since the training dots stay close to the evaluation dots. We anticipated that from the beginning and that is why we introduced the LODO evaluation regime and presented both results. We searched all 30 feature representations against five classifier families and the complete grid is reported in Appendix Table 4. That being said, Table 3 summarizes it per family and shows that the production classifier is not an outlier, other (model, feature representation) pairs coming close to the headline results.

5.3 Applicability Concerns

Tracking the finger's position using reflected light has potential for lots of great applications, for example pre-touch intent applications like pre-fetching content of the links, or preloading applications on mobile phones. More novel applications include but are not limited to current touchscreen kiosks in hospitals or in other crowded environments to reduce the spread of germs or just to increase accessibility.

On the other hand, the technology can be used to gather information about pin codes, passwords and usage habits. Although one of the main purposes of this paper is to show that all data processing can happen on device some alternative implementations can neglect this and publish these types of data in a cloud server.

5.4 AI Usage

For coding tasks AI was used to speed up implementation of boilerplate code, for writing comments and documentation, for refactoring of some methods and files. The essential handwritten code, like the device firmware, and main model training methods, was preserved, but presented in a more maintainable and eye pleasing manner.

In terms of writing assistance AI was used to proofread, to find inconsistencies between paragraphs, to suggest rephrasing within paragraphs, to generate small drafts based on my ideas, and to validate structure and clarity concerns.

The AI's responses were used as suggestions and the end decision was never taken with only an AI's response in mind. Experiment designs regarding hover height and latency, feature representation and model choices were not suggested by the AI, they were ideas extracted from related works and personal judgment. On top of this all data collection and all experiments had to be done manually since they necessitate interaction with the physical sensing device.

6 Conclusions and Future Work

Unlike in other visible light positioning works, where the light sources are fixed and the receiver is moving, in this setup the geometry is reversed. The finger reflects light onto an array of static photodiodes and due to this positive contribution its position can be inferred. We showed that four under-screen photodiodes can localize and track a hovering finger in real time using only the microcontroller that already drives the screen, and without the deep-learning models that prior through-screen sensing relied on. The four photodiodes manage to localize a finger hovering 15 to 20 mm above a 5×2 grid with 77.2% accuracy in a random split evaluation regime and a 66.8% accuracy in a leave-one-calibration-dot-out regime, where every recording of the evaluated position is held out of training. The second one is the figure that more closely resembles how the model performs when deployed and tested in the real world.

When trying to increase the accuracy of our solution we found that using a two-stage classification architecture, where we classified the row and column separately, produced the best results. This architecture choice improved the best performing results of the regression standard by 17 percentage points, having a bigger impact than model choice and feature engineering had. We saw that introducing the squared per photodiode terms generated the strongest gains in cell accuracy, mostly concentrated on the row axis. We also observed that preprocessing techniques like sigmoid function data preprocessing only helped in scale sensitive models like kernel ridge regression and an SVC with a radial-basis-function kernel.

When transitioning from static localization to continuous tracking we observed that the accuracy degrades gracefully as the capture window shrinks, from the headline results of 77.2% at a 1000 ms capture window to 58.3% with only 50 ms of data. On top of this when deploying the tracking firmware on-device we observed sub-millisecond inference time. In terms of memory footprint the model fits comfortably within the 96 KB SRAM and 512 KB flash limit of the device, using 16.2% and 6.9% respectively. That being said, we showed that through-screen reflected light carries enough spatial information for cell level finger localization without using deep learning techniques, by creating a system that can run entirely on an Arduino Due that also manages the screen.

6.1 Future Work

One of the most impactful ways to continue our work is to increase the setup's robustness to changes in ambient light. We can achieve that the same way Okuli did [7], by turning on and off the screen at a 30 Hz frequency and continuously subtracting the values read when the screen was off from the values read when the screen was on, to accurately determine the finger's contribution. Unfortunately this approach would still not improve results when the ambient light is strong enough to saturate the PD readings. A second direction for further improvements targets the X axis' lack of geometric information. We believe that the structural bottleneck of our setup can be mitigated by adding more photodiodes or repositioning them such that their X axis symmetry is broken. Finally, the tracking we evaluated only shrinks the capture window on stationary captures, so tracking a fast moving finger, with an overlapping sliding window and light temporal smoothing, would extend our results and expose effects such as motion blur within the capture window.

References

[1] P. H. Pathak, X. Feng, P. Hu, and P. Mohapatra, "Visible light communication, networking, and sensing: A survey, potential and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2047–2077, 2015. DOI: 10.1109/COMST.2015.2476474.

[2] X. Sun et al., "RSS-based visible light positioning using nonlinear optimization," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 14 137–14 150, 2022. DOI: 10.1109/JIOT.2022.3156616.

[3] Y. Zhuang et al., "A survey of positioning systems using visible LED lights," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1963–1988, 2018. DOI: 10.1109/COMST.2018.2806558.

[4] K. Hinckley et al., "Pre-touch sensing for mobile interaction," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16, San Jose, California, USA: Association for Computing Machinery, 2016, pp. 2869–2881, ISBN: 9781450333627. DOI: 10.1145/2858036.2858095.

[5] Ultraleap, *Leap Motion Controller Optical Hand-Tracking Module Datasheet*, Accessed: Jun. 19, 2026. [Online]. Available: https://cdn.sparkfun.com/assets/9/6/c/2/5/DS-16844-Leap_Motion_Controller_Optical_Hand-Tracking_Module.pdf.

[6] T. Kamijo et al., "A touchless user interface based on a near-infrared-sensitive transparent optical imager," *Nature Electronics*, vol. 6, no. 6, pp. 451–461, Jun. 1, 2023, ISSN: 2520-1131. DOI: 10.1038/s41928-023-00970-8. [Online]. Available: https://doi.org/10.1038/s41928-023-00970-8.

[7] C. Zhang, J. Tabor, J. Zhang, and X. Zhang, "Extending mobile interaction through near-field visible light sensing," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15, Paris, France: Association for Computing Machinery, 2015, pp. 345–357, ISBN: 9781450336192. DOI: 10.1145/2789168.2790115.

[8] H. Liu, H. Ye, J. Yang, and Q. Wang, "Through-screen visible light sensing empowered by embedded deep learning," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '21, Coimbra, Portugal: Association for Computing Machinery, 2021, pp. 478–484, ISBN: 9781450390972. [Online]. Available: https://doi.org/10.1145/3485730.3493454.

[9] SparkFun Electronics, *SparkFun Transparent OLED HUD Breakout (Qwiic)*, Accessed: Jun. 19, 2026. [Online]. Available: https://www.sparkfun.com/sparkfun-transparent-oled-hud-breakout-qwiic.html.

[10] Texas Instruments, *OPT101 Monolithic Photodiode and Single-Supply Transimpedance Amplifier Datasheet*, Accessed: Jun. 19, 2026. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/545673/TI/OPT101.html.

[11] Arduino, *Arduino Due Datasheet*, Accessed: Jun. 19, 2026. [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000062-datasheet.pdf.

[12] K. Deprez, S. Bastiaens, L. Martens, W. Joseph, and D. Plets, "Passive visible light detection of humans," *Sensors*, vol. 20, no. 7, p. 1902, 2020.

[13] T. Komine and M. Nakagawa, "Fundamental analysis for visible-light communication system using LED lights," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 1, pp. 100–107, 2004. DOI: 10.1109/TCE.2004.1277847.

[14] W. Raes, N. Knudde, J. De Bruycker, T. Dhaene, and N. Stevens, "Experimental evaluation of machine learning methods for robust received signal strength-based visible light positioning," *Sensors*, vol. 20, no. 21, p. 6109, 2020.

[15] Y.-C. Wu et al., "Received-signal-strength RSS based 3D visible-light-positioning (VLP) system using kernel ridge regression machine learning algorithm with sigmoid function data preprocessing method," *IEEE Access*, vol. 8, pp. 214 269–214 281, 2020. DOI: 10.1109/ACCESS.2020.3041192.

A Appendix

Table 4: Full two-stage classifier ablation showing cell accuracy for every one of the 30 feature variants crossed with five classifier families in the random $80/20$ evaluation regime across 20 seeds. **Bold** marks each family's best variant.

Feature variant	Dimensions	LogReg	k-NN	SVC	RF	ELM
pn	4	0.392	0.435	0.428	0.449	0.281
raw	4	0.726	0.613	0.718	0.677	0.677
raw+sfdp	4	0.695	**0.636**	0.747	0.673	**0.719**
sfdp+pn	4	0.399	0.423	0.435	0.446	0.440
sq	4	0.737	0.521	0.109	0.673	0.608
sq+sfdp	4	0.640	0.563	0.733	0.674	0.569
log	6	0.419	0.419	0.433	0.497	0.392
log+sfdp	6	0.405	0.428	0.436	0.506	0.442
raw+pn	8	0.724	0.613	0.718	0.647	0.704
raw+sfdp+pn	8	0.717	0.549	0.745	0.646	0.699
raw+sq	8	0.762	0.521	0.105	0.671	0.595
raw+sq+sfdp	8	0.719	0.603	**0.763**	0.669	0.700
sq+pn	8	0.737	0.521	0.109	0.642	0.637
sq+sfdp+pn	8	0.658	0.497	0.705	0.642	0.646
log+pn	10	0.423	0.421	0.449	0.479	0.401
log+raw	10	0.727	0.613	0.718	0.668	0.712
log+raw+sfdp	10	0.708	0.526	0.703	0.676	0.672
log+sfdp+pn	10	0.410	0.429	0.437	0.477	0.423
log+sq	10	0.727	0.521	0.109	0.673	0.596
log+sq+sfdp	10	0.663	0.492	0.686	0.677	0.623
raw+sq+pn	12	0.764	0.521	0.105	0.667	0.653
raw+sq+sfdp+pn	12	0.736	0.568	0.735	0.659	0.695
log+raw+pn	14	0.724	0.613	0.718	0.653	0.697
log+raw+sfdp+pn	14	0.715	0.506	0.692	0.653	0.668
log+raw+sq	14	0.755	0.521	0.105	**0.705**	0.623
log+raw+sq+sfdp	14	0.738	0.522	0.706	0.703	0.690
log+sq+pn	14	0.740	0.521	0.109	0.655	0.632
log+sq+sfdp+pn	14	0.660	0.469	0.682	0.658	0.642
log+raw+sq+pn	18	**0.772**	0.521	0.105	0.681	0.609
log+raw+sq+sfdp+pn	18	0.737	0.526	0.728	0.686	0.695