Type–Hover–Swipe in 96 Bytes: A Motion Sensing Mechanical Keyboard

Stuart Taylor[1], **Cem Keskin**[1], **Otmar Hilliges**[2], **Shahram Izadi**[1], **John Helmes**[1]
[1]Microsoft Research, [2]ETH Zurich
{stuart|cemke|shahrami|v-johelm}@microsoft.com, otmarh@ethz.ch

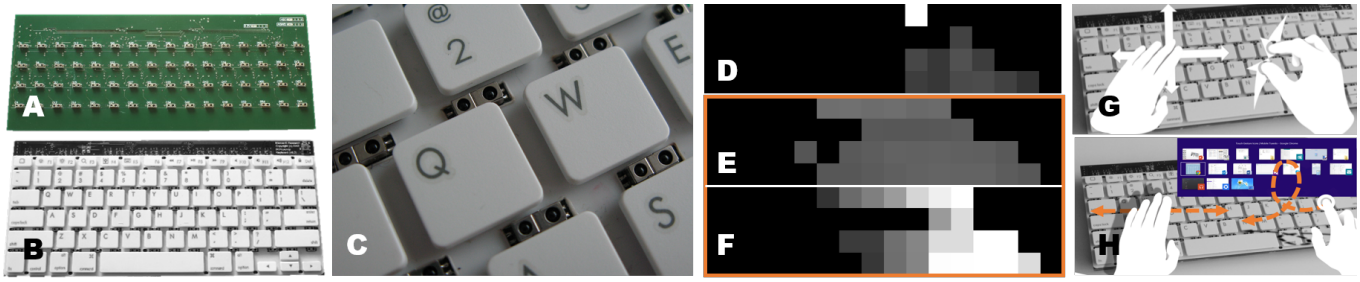

Figure 1. We present a novel mechanical keyboard combining motion gestures on and directly above the keys with regular tactile typing. A low-resolution but high-speed sensor (A) is embedded into an off-the-shelf keyboard (B). IR proximity sensors are interspersed between the keycaps (C). This results in a low-resolution raw intensity image when hands are interacting above (D). A sequence of these images are accumulated into proximity (E) and motion (F) history images. Together these form a *motion signature* (E+F) which can be used to robustly recognize a number of dynamic on-keyboard (G) and hover gestures (H) using a machine learning-based classifier.

ABSTRACT
We present a new type of augmented mechanical keyboard, sensing rich and expressive *motion gestures* performed both *on* and directly *above* the device. A low-resolution matrix of infrared (IR) proximity sensors is interspersed with the keys of a regular mechanical keyboard. This results in coarse but high frame-rate motion data. We extend a machine learning algorithm, traditionally used for static classification only, to robustly support dynamic, temporal gestures. We propose the use of *motion signatures* a technique that utilizes pairs of motion history images and a random forest classifier to robustly recognize a large set of motion gestures. Our technique achieves a mean per-frame classification accuracy of 75.6% in leave–one–subject–out and 89.9% in half-test/half-training cross-validation. We detail hardware and gesture recognition algorithm, provide accuracy results, and demonstrate a large set of gestures designed to be performed with the device. We conclude with qualitative feedback from users, discussion of limitations and areas for future work.

Author Keywords
Input devices; Keyboard; Gesture recognition;

ACM Classification Keywords
H.5.2 User Interfaces: Input devices and strategies

INTRODUCTION
Since the invention of the typewriter in the 1860s, mechanical keyboards have remained the preferred method for text entry. Their haptic nature ensures they are still central to much of today's desktop productivity and gaming scenarios, despite the recent popularity of alternative natural user interfaces (NUIs) such as touch, speech and 3D input.

Given their widespread adoption, research has explored how to *augment* keyboards with additional sensing capabilities. In particular, previous work has demonstrated multi-touch capabilities combined with soft (i.e. non-moving) [36] and mechanical keyboards [3, 10, 11, 12]. The aim of such systems is to *co-locate* gestural interaction with regular typing so that the user can quickly move between a resting 'home-position' and either typing or touch-based interactions. Previously studies have shown advantages of combining keyboard use with touch-based 2D pointing[11, 12]. We build on this related work by presenting a new type of augmented mechanical keyboard, capable of sensing *motion gestures*. These are rich, expressive interactions which are performed both *on* and *directly above* the device.

Our hardware design comprises of a low-resolution matrix of infrared (IR) proximity sensors interspersed between the keys of a mechanical keyboard. This low spatial resolution is intentional, allowing the mechanical qualities of the keyboard to be preserved, and providing the ability to support frame-rates of up to an order of magnitude more than previous touch and in-air sensing systems ($325Hz$). We demonstrate that this trade-off between temporal and spatial resolution allows for a rich set of motion gestures.

The coarse but high-speed motion data generated by our prototype requires a new approach to gesture recognition. We demonstrate a novel machine learning algorithm based on randomized decision forests (RDF) [5]. RDFs have been shown to be effective for classification of static hand gestures

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from **Permissions@acm.org**.
CHI'14, April 26–May 01 2014, Toronto, ON, Canada
Copyright © 2014 ACM 978-1-4503-2473-1/14/04$15.00.
http://dx.doi.org/10.1145/2556288.2557030

[19], hand pose [18], body pose [30] and objects [13], and generally provide high classification accuracy, are robust to noise and are relatively fast to train [5, 9]. However, one limitation of standard RDFs is that they operate on single images and therefore are not well suited for temporal data.

Given the sensor speed, the temporal domain becomes a rich source of information. We present an extension to standard RDFs which natively supports temporal gestures. We propose the use of *motion signatures* a technique that utilizes pairs of motion history images and an RDF based classifier to robustly recognize a large set of motion gestures on and above the keyboard. The technique automatically performs temporal segmentation, is invariant to linear changes in speed and runs in real-time on commodity hardware.

The structure of the rest of this paper is as follows: We review the relevant literature in the domain of augmented keyboards, touch and in-air motion sensing. After highlighting compelling motion gestures and interactions, we detail the sensing hardware (Fig. 1, A-C), followed by a more in-depth exploration of the gesture design space (Fig. 1, G+H). One of the key contributions of our work is the concept of *motion signatures*, we explain the working principle and detail how they can be used to robustly recognize dynamic gestures (Fig. 1, D-F). We present our initial results in terms of recognition accuracy. Finally, we discuss advantages and limitations and conclude with directions for future work.

RELATED WORK

There has been extensive work on the augmentation of standard input devices such as mice, styluses, touch pads and keyboards, highlighting that this is indeed a rich design space for novel interaction. Systems have added multi-touch [33], pressure [6], and display capabilities to mice [38], as well as touch [31] and proximity sensing [23] to styluses.

Augmented Touchpads

Touch pads have for some time supported multi-touch capabilities [1, 29]. Choi et al. [7, 8] extend prior work [16, 17, 24] that uses discrete proximity sensors for multi-touch displays to create a touch and hover pad. A later variant [14] uses a flexible compliant surface with proximity sensors around the edge to support both touch, pressure and hover in a single sensor. The system is intended to replace the standard touch pad on laptops with a wider sensor, occupying a laptops entire width underneath the keyboard, and support both simultaneous or sequential keyboard and touch input.

Whilst the combination of separate touch pad and keyboard is interesting, the sensors must still reside in two physical locations. Systems therefore enforce highly sequential use and require the user to reposition their hands, acquiring the touch pad much the same way as a mouse. This may increase interaction time and interrupt flow especially for quick, modal gestures that do not require the same precision as targeting.

Multi-touch & keyboards

The limitation of using a dedicated gesture device alongside the keyboard has led researchers to explore other configurations where the two are more tightly coupled. The pressure sensitive keyboard [10] allows simple gestures based on sensing the force applied to each key. Fingerworks produced a flat keyboard (without moving keys) with support for multi-touch gestures [36]. The recently announced Microsoft TouchCover 2* uses a resistive force sensing scheme to detect a small number of gestures, performed by exerting small amounts of force while gesturing directly on top of the keys but does not allow for gestures above the keyboard or gesturing without pressing down. PreSense [27] and SmartPad [28] demonstrate single finger capacitive sensing above a mobile phone keypad and number pad, using a single capacitive matrix behind the keypad.

Touch-based pointing on keyboards

Fallot-Burghardt first describe the concept of touch sensing above a full mechanical keyboard to support regular typing and single finger pointer control [12]. A study of the system shows improved time-to-completion over a standard mouse and keyboard configuration [11]. Due to inadvertent triggering of pointing whilst typing, the system required an explicit mode switch (via a mechanical button) to transition between gesturing or typing. Later variants of this system [15] were also capable of sensing multi-touch on top of the physical keys, but did not solve the problem of false triggering when hands are resting or typing on the keyboard. Block et al. [3] combined the concept of coarse, per-key capacitive sensing with projection capabilities, to display a variety of content such as dynamic keyboard legends.

Above keyboard gestures

Other researchers have placed an RGB camera above a mechanical keyboard [37] or two depth cameras above a desk [34] to detect pinch gestures in mid-air for a variety of tasks such as map navigation and CAD. More recently, commercial products such as LEAPMotion† and 3Gear Systems‡, have added in-air gesturing capability either above or in-front of a keyboard using consumer or custom depth sensors. These systems focus on in-air gestures and do not support transitions between on surface to above surface input.

Whilst capacitive systems have some inherent hover capabilities, to our knowledge we are the first mechanical keyboard to support rich motion gestures on and above the keyboard with a single integrated hardware unit and no use of external cameras. Our system does not require an explicit mode switch to transition between gesturing and typing. We achieve a wide range of motion gestures using a robust gesture recognition technique described in detail later.

INTERACTING WITH THE MOTION KEYBOARD

Our system couples coarse motion and hover gestures with a mechanical keyboard, whilst preserving its core function. Fig. 2 illustrates a number of interesting motion keyboard interactions. For example, a user pans and zooms a map using multi-touch style bi-manual gestures directly on the key caps (A). Slightly lifting the splayed left hand off the keyboard and quick subsequent taps invoke the visual task switcher to quickly jump between applications (B). In a graphics application pressing the 'C' key brings up a color wheel and a motion gesture performed with the other hand changes the selected

*`http://www.microsoft.com/surface/`

†`https://www.leapmotion.com/`

‡`http://www.threegear.com`

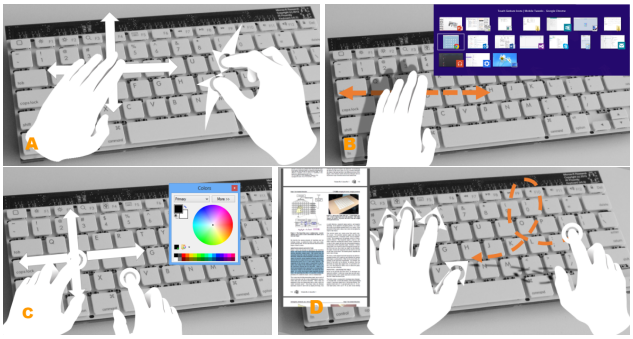

Figure 2. Usage scenarios. (A) Panning and Zooming a document. (B) Open hand hovering above keyboard invokes task switcher. (C) Pressing 'C' brings up color wheel and non-dominant hand selects color. (D) Coarse flicks scroll whole page; index finger performs a touch and hover 'loop' gesture to next/previous paragraph.

color (C). In a document, swiping above the keyboard with the whole hand advances an entire page while a single finger in-air 'loop' advances by a single paragraph (D).

In summary a key goal of enhancing a keyboard in this manner is to enable fast, easy, low-effort gestures and smooth transitions back to typing, where the user's hands always remain in the 'home position'. The motion keyboard enables this in a single, self-contained mechanical package, as opposed to having an external sensor such as a 2D or depth-sensing camera. Our solution coarsely approximates depth, but does allow for rich gestural interaction directly on the keycaps and in a narrow band above.

HARDWARE

Physically the motion keyboard prototype does not differ much from commercially available keyboards. Large parts of the sensor electronics have been integrated into the electro-mechanical 'sandwich' of an off-the-shelf keyboard, which we dismantled, modified and then reassembled with additional custom sensing electronics embedded within it.

The sensor comprises of a two-dimensional array of 64 IR proximity sensors (Avago HSDL-9100-021, Fig. 3), arranged as four rows of 16, where each row lies between two rows of keys. Sensor nodes are surface-mounted on a PCB which is inserted between the array of keycaps and silicone rubber membrane which sits underneath.

Sensor electronics

The sensor LEDs are wired in a matrix fashion, with row and column lines driven by FET switches (Fig. 4). Using this addressing scheme allows individual LEDs to be illuminated by activating just a single row and column. Alternatively, blocks of multiple LEDs can be illuminated by driving several rows

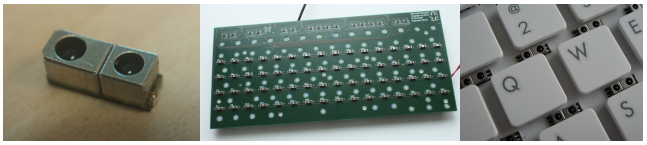

Figure 3. Motion keyboard prototype hardware components. Left: individual IR proximity sensor in metal housing. Middle: Bare PCB with board-mounted 16x4 array of IR proximity transceivers. Note holes cut into the PCB, through which the membrane key switches protrude. Right: IR transceivers interspersed with regular keycaps.

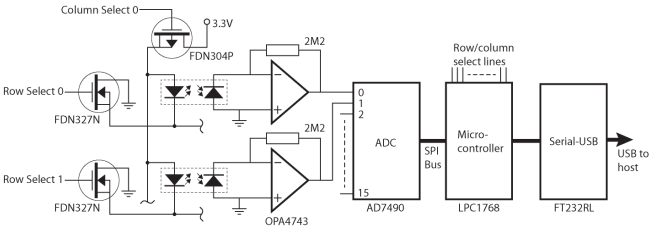

Figure 4. Sensor circuit diagram.

and/or columns simultaneously. One advantage of illuminating multiple LEDs per read is that the hover range above the keyboard can be increased programmatically.

Previously, similar sensor designs [16] have measured a capacitor's discharge time, regulated by the photo-diode, for analog-to-digital conversion. This readout scheme introduces an inherent latency as the charge level must drop to logic state zero. In contrast we use a design where the output from the photodiodes is fed to 16 OPA4743UA quad op-amps in a high-gain configuration, before being passed to four 16-channel, 12-bit analog-to-digital converters (ADCs). The conversion timing and data transfer from the ADCs is performed over an SPI bus (see Fig. 4). This more direct readout scheme allows us to run the circuitry at more than $20\times$ the frame rate of the original Thinsight system [16]. We use an NXP LPC1768 ARM Cortex M3 microcontroller for LED control, co-ordinating ADC readings and transferring data frames to the host PC via USB.

Sensor Data

Capturing one complete frame of proximity data is achieved with a raster scanning process. In sequence, the firmware activates the first row/column line driver (turning on the IR emitter for $24\mu s$), and then instructs the appropriate ADC to perform a conversion of the corresponding amplified photodiode reading. This sequence repeats for each of the sensors, in column order, collecting a complete frame of 64×12-bit ADC readings (i.e. 96 bytes per full frame). The frame is then sent over USB to the host computer for post processing and gesture recognition.

Fig. 5 illustrates typical sensor data from hands moving above the keyboard. Despite the low resolution of only 64 pixels, there is clear signal as the user is swiping or hovering. However, this resolution is not intended for accurate localization of a fingertip. Although fingertip contact is sensed reliably, there are often scenarios where only a single sensor will detect signal, making up-sampling or interpolation of an accurate touch location difficult.

Additionally, the sensor output can be read at high frame rates ($325Hz$), this can benefit in discriminating between intentional gesturing, and regular typing – a much cited issue in previous touch sensing keyboards [11]. Another big advantage of using proximity sensors is that the signal contains data about fingers and hands hovering over the key caps, illustrated through false-color rendering in Fig. 5, (C).

Sensor characteristics and calibration

Each individual sensor node returns raw intensity values that map roughly to proximity. In a first step we establish the

Figure 5. Sensor data. (A) Hand moving across the keyboard, including brief hover period (middle). (B) Sensor data is low-res but can be read at high frame rate (intensity encodes proximity). (C) False-color rendering, scale from blue (0mm) to red (400mm).

noise floor for each cell by capturing a series of empty frames which are accumulated into a background model that can later be used for foreground subtraction.

We conducted experiments to find a model that best linearizes the raw intensity values and maps them to proximity estimates. Fig. 6 (left) shows the response curve for a single sensor. Measurements where made of a 100mm square of gray cardboard mounted on a motorized stage, and human finger stabilized on the same stage. Both targets were measured at fixed intervals from the sensor, parallel to the keycaps. Fig. 6 (right) plots sensor readings against the reciprocal of squared distance, showing that, for both targets, the sensor response tracks the inverse square law reasonably well. These experimental results show that the sensor readings are meaningful in a range from 6mm-33mm (as indicated by the dotted green line). We use these curves to linearize the data and define coarse depth estimates $D = \frac{1}{\sqrt{I}}$ from raw intensity values I.

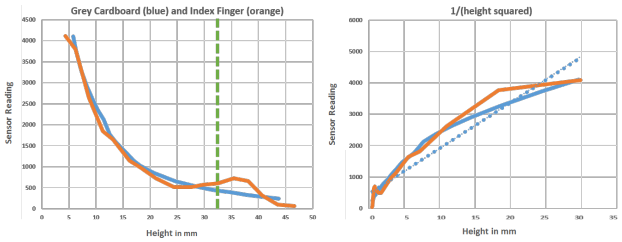

Figure 6. Left: Response curves for a single sensor where gray cardboard (blue) and human finger (orange) are positioned at varying distances from the sensor. Right: Raw intensity values approximate inverse square law (dotted line).

GESTURE RECOGNITION

Now that the reader has a feeling for the hardware configuration and the sensor data, we want to shift the focus towards sensing and gesture recognition. In this section we first introduce a number of gestures that we seek to enable, followed by an in-depth description of our recognition engine.

The gestures discussed here are not necessarily an attempt to design a final end-user experience but serve the dual purpose of exploring the sensor characteristics, illustrating the flexibility and power of the gesture recognition engine, and further highlight the interactive capabilities of our system.

On-Keyboard swipes

The most basic class of gestures we support are on-keyboard swipes, shown in Fig. 7. These directional swipes may be used to scroll and pan large documents, to trigger scrolling or to flick through a collection of images. Swipes may be performed with either hand, single or multiple fingers and anywhere on the keyboard. Our recognition engine can discriminate gestures performed by left and right hand (assuming hands are never crossed) and between large swipes across the central areas of the keyboard (see Fig. 7, left) and shorter swipes crossing the edges of the physical keyboard (see Fig. 7, right). We can separately create compound gestures such as swiping in an out across an edge of the keyboard.

Figure 7. On-Keyboard swipe gestures. Gestures can be performed by either hand with single finger or whole hand.

On-Keyboard shapes

A variant of the coarse swipes for navigational purposes, are shape or stroke based gestures which could be used to trigger discrete commands. These gestures make use of the physical keyboard as visual landmark to aide recall. For example, one gesture is performed by 'drawing' an up arrow onto the keyboard, which causes a window to maximize (Fig. 8, left). The stroke starts with "Left-Alt" moving to "Y" and ending on the "Right-Alt" key. Similarly, a user can use strokes to perform complex keyboard shortcuts and leverage muscle memory to better memorize them, for example a "Ctrl-Shift-D" stroke inverts a selection in Photoshop (Fig. 8, right).

Figure 8. On-Keyboard shapes. Shape drawing to perform discrete commands. Physical keyboard as landmark may help as mnemonic aide in gesture memorization.

Multi-touch gestures

We also recognize a number of multi-touch inspired gestures. For example, a static pinch gesture where the index finger touches the thumb may invoke an OS wide search functionality. Furthermore, dynamic variants of these gestures can be recognized as well where a user may pinch to zoom in and out of a map or photo.

Micro-hover gestures

These are a class of gestures where both hands remain in home position, ready to type or are already typing when the user quickly lifts one or several fingers to issue commands. For example, quickly lifting the open left hand of the keyboard and lowering it back to resting may invoke a marking menu on screen (Fig. 9, left). Coarse swipes can then be used to navigate the menu and selection could happen by quickly lifting and lowering the outstretched index finger of the right hand (Fig. 9, right).

Our system also recognizes static variants of these gestures such as a single finger held for a short duration hovering above the keyboard, or a victory sign also held hovering above the keyboard. These could be used to switch between two modes in an application.

Dynamic hover gestures

We exploit the proximity sensing capability to add further commands to the gesture set by discriminating stroke-based gestures that are performed entirely on the keyboard from those that contain an in-air portion. One example of this class is a horizontal stroke starting roughly on "LKJ" keys after which the finger lifts off the keyboard performs a tight loop and ends the stroke by touching roughly on "GFD" keys (Fig. 10, left). This gesture may perform an undo while the opposite direction may perform a redo. Another hover gesture could forgo the in-air loop and simply include a hover segment in an otherwise straight stroke (Fig. 10, right), for example invoking a task switcher with a preview function such as Window's Aero Peek.

Figure 10. Dynamic hover gestures. Part of the gesture trajectory is performed in narrow band above the keyboard.

This gives a flavor for the types of gestures we wish to implement. There are two aspects that are important to highlight.

First, whilst our sensor is coarse, the gesture set described here is already reasonably *large*. To perform all the motion and static gestures described earlier in this section, in all the different directions across the keyboard would result in needing to discriminate between a total of 28 gestures, from only 96 bytes of raw proximity data.

Second, many of the gestures share similarities in their trajectories. Our approach is capable of distinguishing gestures that follow the exact same path but with one performed entirely on the keyboard and the other containing a hover section.

These two aspects suggest that heuristic-based recognition approaches would struggle to robustly discriminate between gestures (this is an intuition actually born out of our own experiences during the development of this system). We therefore have explored a novel machine learning based technique for gesture recognition.

METHOD

Our approach leverages randomized decision forests (RDFs) for frame-by-frame gesture classification. RDFs have been shown to be effective for a number of high-level computer vision problems such as object recognition [13], body part [30] and hand shape classification [18] and generally provide high classification accuracy at low computational cost.

Figure 9. Micro-hover gestures. Hands remain in home position while typing or resting. Quick lifts off fingers triggers different commands.

However, RDFs are a per-frame technique that is they do not inherently lend themselves to the classification of temporal activities such as human gestures. One of our contributions in this context is a simple but powerful way to represent temporal (and associated proximity) data so that it can be used in one-shot, direct motion classification using RDFs.

The basic idea is that of training a classifier to recognize a *motion signature* rather than a single image of an object. This motion signature encodes the accumulation of motion information over a number of frames (a temporal signature) into a single image. A second image encodes a running average of the proximity information over the same duration.

At training time we annotate a series of these motion signatures with ground truth gesture labels (data capture and labeling is described later), and train a RDF with three trees. At test time, we classify each pixel in the motion signature image into one of the gestures. Note that this approach does not introduce latency, each accumulated image contains a motion signature that allows the RDF to recognize the gesture even before it is completed. The technique can also cope with different gesture execution speeds and variations in motion trajectory (within limits of the training data).

Our approach has some similarities to the method of [25]. However, this previous approach requires the extraction of a large number of features and relies on already available per-pixel body part classification from depth images. We extend the capability to directly classify temporal actions in a way that operates on essentially unprocessed data and does not require computationally expensive feature extraction.

Building motion signatures

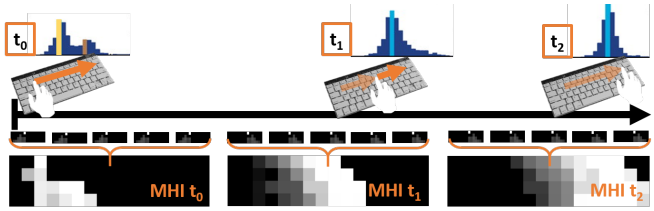

Figure 11. Generating a motion history images. Each MHI contains 160 frames of history. All frames t_1 to t_2 are labeled as a specific gesture in the training data. At test time each pixel in the MHI is classified. All MHIs within $t_1 - t_2$ range will be labeled as the corresponding class.

To represent important temporal information necessary to robustly recognize gestures from a single image we build upon and extend the concept of motion history images (MHI)[4]. MHIs are formed by accumulating binary foreground masks, weighted according to their age in a fixed size temporal window. More weight is assigned to recent frames, producing visually recognizable motion trajectories, shown in Fig. 11.

Note that MHIs can be used to recognize both dynamic and static gestures, without motion the MHI is identical to a single foreground mask. However, the binary foreground mask alone does not contain depth information and is thus not suitable to represent the type of hover gestures we seek to recognize. To retain the proximity information in the signal, we accumulate a second image representing motion from weighted intensity images instead of binarized foreground masks.

During training and classification time, we use *both* the binary-MHI (bMHI) and intensity-MHI (iMHI). The bMHI do not contain hover information, and the iMHIs are ambiguous to whether changes correspond to motion over time or changes in depth. Hence, neither bMHI nor iMHI alone are sufficient to robustly detect all types of gestures. However, by using the pair of images together as *motion signature*, we can distinguish between static, dynamic, and hover gestures.

At each time-frame t, the sensor data consists of low resolution 16x4 IR intensity image I_t (12-bit intensity values). We threshold I_t to discard sensor noise to form I_t^s, and binarize it producing the foreground mask I_t^m. These masks are summed $\sum_{i=0}^{k-1} w_{k-1-i} I_{t-i}^m$, where $w_i = \frac{2i}{k(k-1)}$, and k is the number of frames used in the history and i is the current frame index. To form the iMHI, we first normalize the data and produce an approximated depth image I_t^d by applying the reciprocal of the inverse-square law. The iMHI is then the weighted sum of the last k depth images as before.

Training data

In order to train a classifier to robustly recognize a large gesture set with significant variation in terms of execution speed and motion trajectory, we need a database which reflects as much of this natural variation as possible.

We asked 11 subjects recruited from the lab to perform many examples of the 28 gestures outlined previously. Instances of each gesture were semi-automatically delimited and annotated with a class label according to the current gesture. To avoid biases introduced by learning and fatigue effects, gestures were performed in a random order, and counterbalanced appropriately. To capture a wide variation of gesture styles, subjects received minimal instruction on how to perform gestures. Only clear outliers were removed from the database.

We recorded raw frames at $325Hz$ and captured a total of 28 gestures $\times 10$ repetitions $\times 11$ subjects resulting in 3080 sequences. We additionally recorded large amounts of data containing many different types of miscellaneous non-gestural interactions, including typing but also hands moving to and away from the keyboard. This data adds roughly 30% to the entire database and is collectively labeled as 'Typing'. This catch all class for *non-gestures* greatly increases overall robustness as it allows the classifier to learn tighter decision boundaries around the actual gesture classes.

A key contribution is our motion signature representation. Instead of using the entire sequence as an example gesture, we use clusters of (iMHI, bMHI) pairs. At training time we compute these pairs for every tenth frame in all sequences, since consecutive images are very similar due to high frame-rate. The sequences are split into a gesture onset period t_0 to t_1 and a gesture completion period t_1 to t_2 (see Fig. 11). Assuming that the (iMHI, bMHI) pairs from t_1 onward carry a unique signature of the gesture to reliably classify it at test time. We have experimented with different sequence split ratios and have found $t_1 = 0.2 \times t_2$ to be a good compromise between recognition latency and accuracy. MHI pairs extracted from the onset periods are included in the non-gesture class to prevent early false triggering of the recognizer.

Classification

We use a standard RDF classifier similar to previous body [30] and hand pose [19] estimation methods. These approaches use ensembles of decision trees to classify depth images. Decision trees consist of internal split nodes, used to analyze the data, and leaf nodes, used to infer the posterior probability of the class label. At test time, each split node sends the incoming input to one of its children according to the test result. For the final decision, the posterior probabilities estimated by all the trees in the ensemble are averaged.

Tests typically are of the form: $f_n(F_n) < T_n$, where f_n is a function on features F_n and T_n is a threshold, for the split node n. The training of a decision tree involves determining the tests and collecting statistics from a training set in a supervised manner. In the case of RDFs, f_n operates on a subset of the features selected during training. This is done by randomly selecting multiple function candidates and choosing the one that best splits the data. For more details see [9].

Here, we follow the same methodology, but instead of depth features (correlating to the shape of the object) we use MHI pixel intensity differences, which correlate to the dynamic signature of the gesture for bMHI, and to the proximity of the hand for iMHI. Furthermore, during training time the technique selects features from both the bMHI and iMHI, as both can contain information crucial for gesture discrimination. Given an MHI $I(\boldsymbol{x})$, where $\boldsymbol{x}$ denotes location, we define a feature $F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x})$ as follows:

$$F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x}) = I(\boldsymbol{x} + \boldsymbol{u}) - I(\boldsymbol{x} + \boldsymbol{v}) \quad (1)$$

The offsets $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors relative to the pixel in question. At training time, each split node is assigned a pair of offsets and a threshold that best separates the labeled training data. The leaf nodes are assigned the histograms of class labels of the pixels that end up in those nodes.

At run-time, every pixel is evaluated independently with each tree in the forest. The histograms at the leaf nodes are then normalized and averaged to obtain the posterior likelihood of each pixel. These posteriors are pooled across the image to form a final likelihood for the entire frame. The sequence of posteriors is then smoothed with a running mean filter, and the class label of the mode is assigned to the current frame.

Discussion of Method

Our gesture set includes the static, dynamic and hover gestures illustrated earlier. We picked an RDF classifier over other methods, mainly because they are accurate and robust to noise and can classify each gesture type without modifying the model. Furthermore, RDFs are fast enough to enable classification in real-time, can skip high level feature extraction and work directly on images, and local feature tests can

be very simple pixel operations. RDFs are also highly parallelizable (every tree and every pixel is independent).

Whilst in principle it is feasible to use other classifiers in combination with *motion signatures*, we briefly want to highlight potential issues with other classification approaches. Hidden Markov Models (HMMs) [26] or Support Vector Machines (SVMs) [2] are not easily applicable to classification of such motion signals. HMMs typically model feature sequences rather than image sequences. It is an open research question which features would work for our low-res IR images. Designing HMMs that can model both dynamic and static signals is also difficult and uncommon. More importantly, continuous recognition is not straight-forward and usually requires mechanisms for gesture spotting [22]. SVMs on the other hand, lend themselves primarily to the classification of static gestures. A common approach is to extract a single feature vector from a fixed number of images to train the SVM. This again brings up the non-trivial feature selection problem, and other pre-processing steps such as noise reduction may prove necessary. Note that using an MHI as a 64 dimensional feature vector would lead to gestures being conditioned on their locations, which is undesirable. RDFs do not suffer from this issue by using relative features only.

SYSTEM EVALUATION

We have introduced a novel scheme to directly and per-frame recognize dynamic or motion gestures using RDF classifiers. Our approach is centered around the idea of *motion signatures* capturing the essence of both static and motion gestures. In this section we detail experiments we have conducted to evaluate the performance of our classification scheme.

Test Data and Forest Parameters

The forest size (number of trees) and the depth of the trees are the two main parameters affecting classifier performance. Increasing the forest size enhances the accuracy at a linear computational cost. To ensure that recognition is fast enough for real-time interaction at $325Hz$, we set the number of trees to 3. Table 1 summarizes performance at different tree depths.

To evaluate the accuracy of our gesture recognition technique we collected an additional data set of 28 gestures in total collected from 11 users. 6 of these gestures are essentially the same motion, performed by left or right hand on different locations of the keyboard (e.g. swipe-right with left or right hand). We merged these gestures such that there is no distinction between left and right hands at the RDF level. The hand type is determined at a higher level based on where the gesture is performed. After these modifications we have 21 gesture classes and one non-gesture class. We report results using both leave–one–subject–out cross–validation and by using half of the gesture samples in the set for training and the remaining half for validation.

Table 1 summarizes overall classification accuracy, averaged over all recognized gestures. Increasing tree depth improves classification accuracy but also increases training time and memory footprint. Over-training starts at depth 19, so we set the tree depths to 18.

	15	16	17	18	19	20
Tr PPCR	0.69	0.74	0.79	0.83	0.86	0.89
Tr SCR	0.80	0.82	0.85	0.89	0.92	0.95
Ts PPCR	0.57	0.58	0.59	0.6	0.60	0.59
Ts SCR	0.74	0.73	0.75	0.76	0.75	0.75

Table 1. Classification accuracy for half training / half test (Tr) and leave–one–subject–out (Ts) cross validation. PPCR stands for per-pixel classification rate and SCR for state classification rate. Columns report accuracy for different tree depths. Over-fitting starts at depth 19.

The accuracy of the leave–one–subject–out technique directly depends on the inter-personal variation in the dataset. Users have their own preferences for natural gestures, and the variation is quite high. As the number of subjects increases, so does the variation covered in the training set, and the model generalizes better to previously unseen subjects. At depth 18 the accuracy of leave–one–subject–out technique converges in our case and over-training starts when depth is increased more. This accuracy can be viewed as a lower bound. Half training–half test scheme is provided to give a better estimate of the power of the proposed method. Overall, recognition performance is very good even with very shallow trees for this technique (cf. Table 1).

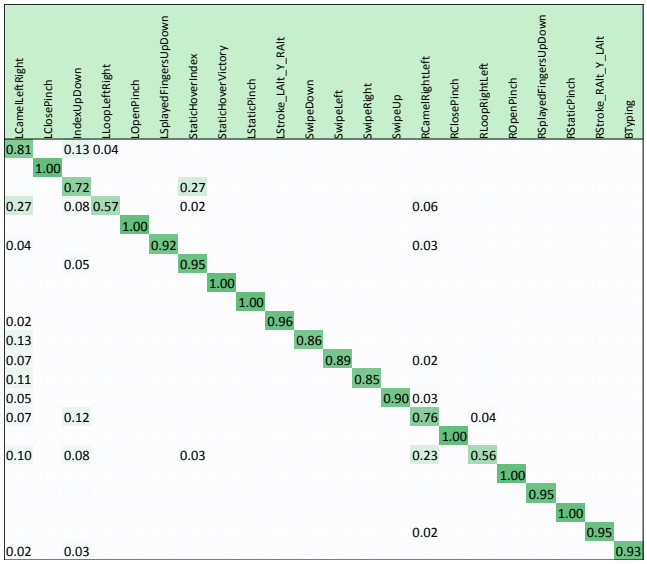

Figure 12. Confusion matrix for all 22 recognized gestures, test data recorded from 11 participants. Results from our forest-based classifier; mean success rate 88.9%.

Fig. 12 summarizes classification accuracy as a confusion matrix for the entire gesture set that an RDF has been trained on. Our technique achieves generally very good classification accuracies with a mean per-class accuracy (mean of the confusion matrix's diagonal) of 88.9%. Furthermore, many gestures achieve close to 100% accuracy per frame. Please keep in mind that these are raw, per frame classification rates. Simple temporal low-pass filtering will further improve these and for most gestures real-world accuracy should be higher. It is worth noting very successful, per-frame classification approaches such as the Xbox Kinect body-tracker, which is based on [30], 'only' reports 60% per-frame mean accuracy yet provides good enough performance when combined with temporal tracking for a mass market product.

The results show excellent classification performance for dynamic and static gestures on the keyboard, and some confusion between the dynamic hover gestures 'Wave' and 'Loop'. The primary reason for the reduced accuracy for the hover gestures is the loss of signal when the hand is out of range of the sensor. The remainder of the motion signature is very similar for both gestures, making it hard to distinguish between the signals.

Also important to highlight is the 'non-gesture' classification rates of 93%, as indicated previously this is the state in which users were typing, resting their hands, and not performing a gesture. This accuracy level is compelling and adds more substance to the machine learning based approach outlined.

PRELIMINARY USAGE OBSERVATIONS

So far we have concentrated on evaluating the gesture recognition accuracy and specifically the impact of *motion signatures*, which we deem one of our main contributions. We also ran some very informal 30 min sessions with 8 users (6 male, 2 female) recruited from our lab, just to gather initial feedback and observations regarding the prototype.

Uniformly all participants instantaneously 'stress-tested' the system to see if it still worked for real text input. All participants expressed real surprise at how little the gestures impacted the keyboard functionality. Once they had learned some of the gestures they quickly tried to explore and understand the whole corpus. The left-right swipes Fig. 7 and the hover gestures that do not require much lateral motion, were well received (Index Up/Down, SplayedHand Up/Down (Fig. 9), Index Loop (Fig. 10, left)).

While other gestures received some criticism for being cumbersome to perform (Swipe Up/Down) or difficult to remember (Hover Stroke) (Fig. 10, right). The former might be due to the keyboard's form-factor which has only limited depth and hence larger hands have very little room to travel in the up/down direction. Finally, one idea that was seen as something with a lot of potential was that of combining the benefits of the keyboard (visual landmarks, haptic feedback) with that of stroke based character recognition or 'shape writing' [20] (e.g., Maximize/Minimize (Fig. 8)) to augment or replace keyboard short cuts – similarly too prior work on touch keyboards [21]. Compound gestures comprising of key presses and swipes such as the color picker example Fig. 2 (C) were also received favorably. Some users even generated several new ideas based on these examples. For example, controlling volume, screen brightness and other continuous OS settings in such a manner.

Of course these observations are only preliminary. Our gesture set was in part designed to evaluate the range of variation our recognizer can cope with. Nonetheless, the current gestures illustrate the utility of enriching mechanical keyboards with gesture sensing capabilities, especially those that go beyond simple 2D pointing. A key benefit seems to be the fact that the system can robustly detect very quick and subtle gestures such as the rapid 'SwipeLeft', 'SwipeRight' gestures and the 'IndexFingerLoop' gesture. Furthermore, users commented that the absence of an explicit mode switch is very compelling. This can be seen as partial confirmation of our initial design goal of low-effort gestures, allowing user's hands to always remain in the 'home position' and of smooth transitions between typing and gesturing.

A final important observation was that of user's initial concerns and reservations with the concept of a sensor embedded in a keyboard, which is perhaps understandable given the importance of the keyboard for their day-to-day work life. However, all of the participants expressed their surprise in how robust the system is in delineating resting and typing motion from intentional gesturing. Hence, many participants indicated that they would use such a system in earnest if, and only if, it provided close to zero false gesture activation. We feel this backed up by our initial accuracy rates, and highlights the importance of a sophisticated and robust gesture recognition engines such as the one outlined here.

DISCUSSION

The initial quantitative and qualitative results of the motion keyboard in use are compelling. Classification results based on motion gestures performed by a variety of users on the keyboard illustrates that even our initial prototype can be used for a diverse set of gestures, even with the low resolution and noise characteristics of our sensor. To our knowledge this is the first time that a touch and hover based keyboard has been demonstrated, especially with this level of robustness for gesture recognition. One important aspect to note is that given our recognition engine is machine learning based, new gestures can be added simply by providing new examples, and retraining our system.

There are also clearly many areas of improvement, which we discuss in this section, outlining limitations and future work. As with all vision systems, ambient light is an issue. We have experimented with IR cut-off filters to alleviate common ambient IR sources (e.g. natural lighting, and room and desk lights), but strong direct light sources are still an issue.

More resolution

In this paper we have intentionally designed a keyboard with low spatial resolution sensing. First, we have wanted to use this device for coarse and lightweight (i.e. low precision but also low effort) gestures that are easy to perform. This takes an opposite approach to systems such as [11], which have looked at replacing the mouse. We on the other hand wish to complement mouse interactions with motion gestures mapped to other UI actions. There have also been mechanical and electronic constraints and cost and power considerations which have also pushed us to choose our current sensing resolution. Clearly, there are ways to improve the spatial resolution in the future. One interesting possibility is to use a denser [23] arrangement of IR proximity sensors. Reverse-biased IR LEDs could also give more flexibility removing the need for an emitter/receiver pair, and instead integrating both sensing or illumination into a single unit (which could be driven using interesting sampling strategies as described below). Higher resolution sensing could allow for fine grained gestures to be coupled with coarser gestures using different readout schemes. For example, the sensor could operate by default in a "coarse" mode of operation allowing regular mo-

tion gestures to be sensed, but once detected, could switch to finer scale sensing, allowing fingertip interactions.

How unique are motion signatures?
One interesting finding from experiments is that even with the coarseness of the current sensor configuration we are able to use RDFs to distinguish between diverse *static* gestures robustly (as well as dynamic). There is clearly signal present even at this coarse resolution, which is being learned by our RDF. So clearly our input has some unique signature even for a single frame of data (rather than multiple frames in a motion signature). The question then becomes whether this per-frame signature is unique enough to identify coarsely the overall hand pose, potentially super-resolving the shape of the hand, from the low resolution signal. Inspired by recent work on image retrieval from tiny images [32, 35], we think it may be possible to apply a database look-up scheme to retrieve high-res images of the user's hand, whereby the 96-byte raw sensor data is used as hashing index. To this end we have recorded sequences of 2D images from a top down camera pointing at our sensor, and captured interactions with the sensor using a single finger sliding across the surface.

At test-time we estimate the hand configuration (and in this case the fingertip location) by performing a nearest neighbor lookup into the database (Fig. 13). Initial experiments are promising and seem to indicate some uniqueness per frame even for subtle hand pose differences.

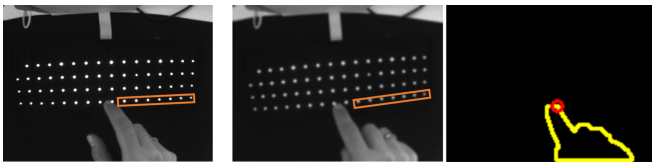

Figure 13. Fingertip localization. Left: Ground truth data. Middle: best candidate from kNN search. Right: Hand contour is traced and fingertip extracted and tracked over time.

Intelligent sampling
Relating to the issue of teasing out more data from the signal we have, is the ability to use more sophisticated sampling mechanisms. Currently, we use a simple scan-line based sampling scheme where each emitter is turned on for a small fraction of time ($24\mu s$) sequentially. However, there might be other ways of sampling. For example, rather than turning a single emitter on in sequence, we could turn a neighborhood of 3×3 around the current sensor on (i.e. 8 other emitters, with the one in the middle sensing). Given the speed of the sensor, we could do this type of sampling every other frame. This could become a separate iMHI and bMHI, which could improve the "uniqueness" of motion signatures, potentially improving our kNN approach outlined earlier.

Per-pixel calibration
Whilst we have started to characterize the response of the proximity sensors within our keyboard, we have yet to exploit this knowledge fully during calibration. Fig. 4 plots the response curve of a single pixel, and as noted, the readings become non-linear the further we move away from the sensor (essentially following the inverse square law). One can accommodate for some of these non-linearities by correcting all pixels (based on the simple inverse square law approximation), as currently implemented. However, we have empirically noted that each sensor has a different response curve, so instead it might be possible to correct each sensor independently. This is akin to radiometric calibration of cameras.Here operating the sensors occasionally in "shadow mode" where no active illumination is used for a single frame, could also allow subtraction of ambient light to help reduce external IR interference.

Interestingly despite our current naive calibration process, our classifier is robust to these differences in per pixel responses. Clearly with radiometric aligned responses from the sensors, our classification could potentially be improved further. However, the RDF is also encoding and learning this mapping in an implicit way. Another avenue of future work therefore could be to have the RDF more explicitly learn the individual response curves of each pixel.

Other hardware/software configurations
Another interesting area of research is to think about both our sensor and gesture recognition in new contexts. Whilst we have demonstrated the advantages of our sensor in a mechanical keyboard, in the future it might be interesting to consider new form-factors such as a stand-alone touch and hover pad. One of the limitations of our current device is extended depth. One interesting possibility is to combine our bottom-up sensing approach with top-down depth cameras to capture interactions in this extended space, using both sensor modalities depending on the proximity to the keyboard. We also feel that our motion signature classification algorithm is general and can be applied to other sensor modalities beyond the motion keyboard, such as depth cameras. Obviously, another big area of future experience is in investigating more specific application scenarios for the motion keyboard, especially ones such as gaming or CAD, where the proximity could be directly mapped to 3D interaction.

CONCLUSIONS
We have presented a new input device, a motion sensing mechanical keyboard. Our sensor allows a variety of gestures on and directly above the keyboard. The device allows very lightweight gestures to be performed without moving your hands away from the keyboard. Our main contributions can be summarized as follows: 1) a new motion sensing keyboard prototype which for the first time demonstrates both touch and hover gestures; 2) a new gesture recognition engine for robustly identifying both static and temporal gestures using a single *motion signature* and RDF framework; 3) a large example gesture set realized using our novel hardware prototype and classification algorithm.

ACKNOWLEDGMENTS
We thank Christoph Rhemann, Alex Butler and Christopher Zach for insightful discussions and Emily Whiting for providing the video voiceover.

REFERENCES
1. Apple Inc. Magic TrackPad, 2010.
2. Bahlmann, C., Haasdonk, B., and Burkhardt, H. Online handwriting recognition with support vector machines -

a kernel approach. In *Proc. Frontiers in Handwriting Recognition, 2002.* (2002), 49–54.
3. Block, F., Gellersen, H., and Villar, N. Touch-display keyboards: transforming keyboards into interactive surfaces. In *Proceedings of ACM CHI* (Apr. 2010), 1145–1154.
4. Bobick, A., and Davis, J. The recognition of human movement using temporal templates. *IEEE PAMI 23*, 3 (Mar. 2001), 257–267.
5. Breiman, L. Random Forests. *Machine Learning 45*, 1 (Oct. 2001), 5–32.
6. Cechanowicz, J., Irani, P., and Subramanian, S. Augmenting the mouse with pressure sensitive input. In *Proceedings of ACM SIGCHI* (Apr. 2007), 1385–1394.
7. Choi, S., Gu, J., Han, J., and Lee, G. Area gestures for a laptop computer enabled by a hover-tracking touchpad. In *Proceedings APCHI*, ACM Press (New York, New York, USA, Aug. 2012), 119–124.
8. Choi, S., Han, J., Kim, S., Heo, S., and Lee, G. ThickPad: a hover-tracking touchpad for a laptop. In *Adjunct Proceedings of ACM UIST* (Oct. 2011), 15–16.
9. Criminisi, A., and Shotton, J., Eds. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer London, London, 2013.
10. Dietz, P. H., Eidelson, B., Westhues, J., and Bathiche, S. A practical pressure sensitive computer keyboard. In *Proceedings of ACM UIST* (Oct. 2009), 55–58.
11. Fallot-Burghardt, W., Fjeld, M., Speirs, C., Ziegenspeck, S., Krueger, H., and Läubli, T. Touch&Type. In *Proc. NordiCHI '06* (Oct. 2006), 465–468.
12. Fallot-Burghardt, W., Speirs, C., Ziegenspeck, C., Krueger, H., and Läubli, T. Touch&TypeTM: a Novel Input Method for Portable Computers. In *Proc. INTERACT* (2003), 954–957.
13. Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. Hough forests for object detection, tracking, and action recognition. *IEEE PAMI 33*, 11 (Nov. 2011), 2188–202.
14. Gu, J., Heo, S., Han, J., Kim, S., and Lee, G. LongPad: A TouchPad Using the Entire Area below the Keyboard on a Laptop Computer. In *ACM CHI'13*, ACM Press (Paris, France, 2013).
15. Habib, I., Berggren, N., Rehn, E., Josefsson, G., Kunz, A., and Fjeld, M. DGTS: Integrated Typing and Pointing. In *Proc. INTERACT*, vol. 5727 of *Lecture Notes in Computer Science*, Springer Verlag (Berlin, Heidelberg, 2009), 232–235.
16. Hodges, S., Izadi, S., Butler, A., Rrustemi, A., and Buxton, B. ThinSight: versatile multi-touch sensing for thin form-factor displays. In *Proceedings of ACM UIST* (Oct. 2007), 259–268.
17. Hofer, R., Naeff, D., and Kunz, A. FLATIR: FTIR multi-touch detection on a discrete distributed sensor array. In *Proceedings of TEI '09*, ACM Press (New York, New York, USA, Feb. 2009), 317.
18. Keskin, C., Kiraç, F., Kara, Y. E., and Akarun, L. Hand Pose Estimation and Hand Shape Classification Using Multi-layered Randomized Decision Forests. In *ECCV 2012*, Lecture Notes in Computer Science, Springer Verlag (Berlin, Heidelberg, Oct. 2012), 852–863.
19. Keskin, C., Kiraç, F., Kara, Y. E., and Akarun, L. Randomized decision forests for static and dynamic hand shape classification. In *2012 IEEE CVPR Workshops*, IEEE (June 2012), 31–36.
20. Kristensson, P.-O., and Zhai, S. SHARK 2. In *Proceedings of ACM UIST* (Oct. 2004), 43.
21. Kristensson, P. O., and Zhai, S. Command strokes with and without preview. In *Proc. ACM CHI* (Apr. 2007), 1137.
22. Lee, H.-K., and Kim, J. H. An HMM-based threshold model approach for gesture recognition. *IEEE PAMI 21*, 10 (1999), 961–973.
23. Liu, S., and Guimbretière, F. FlexAura: a flexible near-surface range sensor. In *Proceedings of ACM UIST* (Oct. 2012), 327–330.
24. Moeller, J., and Kerne, A. ZeroTouch: an optical multi-touch and free-air interaction architecture. In *Proceedings of ACM CHI* (May 2012), 2165.
25. Nowozin, S., and Shotton, J. Action Points: A Representation for Low-latency Online Human Action Recognition. Tech. rep., Microsoft Research Cambridge, 2012.
26. Rabiner, L., and Juang, B. An Introduction to Hidden Markov Models. In *IEEE Acoustic Speech Signal Processing Magazine* (1986), 3–4.
27. Rekimoto, J., Ishizawa, T., Schwesig, C., and Oba, H. PreSense: interaction techniques for finger sensing input devices. In *Proceedings of ACM UIST* (Nov. 2003), 203–212.
28. Rekimoto, J., Oba, H., and Ishizawa, T. SmartPad: a finger-sensing keypad for mobile interaction. In *CHI '03 extended abstracts*, ACM Press (New York, New York, USA, Apr. 2003), 850–851.
29. Rosenberg, I., and Perlin, K. The UnMousePad: an interpolating multi-touch force-sensing input pad. *ACM Transactions on Graphics 28*, 3 (July 2009), 1.
30. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, IEEE (2011), 1297–1304.
31. Song, H., Benko, H., Guimbretiere, F., Izadi, S., Cao, X., and Hinckley, K. Grips and gestures on a multi-touch pen. In *Proceedings of ACM CHI* (May 2011).
32. Torralba, A., Fergus, R., and Freeman, W. T. 80 million tiny images: a large data set for nonparametric object and scene recognition. *IEEE PAMI 30*, 11 (Nov. 2008), 1958–70.
33. Villar, N., Cao, X., Chen, B., Izadi, S., Rosenfeld, D., Benko, H., Helmes, J., Westhues, J., Hodges, S., Ofek, E., and Butler, A. Mouse 2.0: multi-touch meets the mouse. In *Proceedings of ACM UIST* (Oct. 2009).
34. Wang, R., Paris, S., and Popović, J. 6D hands: markerless hand-tracking for computer aided design. In *Proceedings of ACM UIST* (Oct. 2011), 549–558.
35. Wang, R. Y., and Popović, J. Real-time hand-tracking with a color glove. In *ACM Transactions on Graphics*, vol. 28 (July 2009).
36. Westerman, W., Elias, J. G., and Hedge, A. Multi-Touch: A New Tactile 2-D Gesture Interface for Human-Computer Interaction. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 45*, 6 (Oct. 2001), 632–636.

37. Wilson, A. D. Robust computer vision-based detection of pinching for one and two-handed gesture input. In *Proceedings of ACM UIST* (Oct. 2006), 255–258.
38. Yang, X.-D., Mak, E., McCallum, D., Irani, P., Cao, X., and Izadi, S. LensMouse: augmenting the mouse with an interactive touch display. In *Proceedings of ACM SIGCHI* (Apr. 2010), 2431–2440.