TypingRing: A Wearable Ring Platform for Text Input

Shahriar Nirjon, Jeremy Gummeson, Dan Gelb, Kyu-Han Kim
Hewlett-Packard Labs, CA, USA
{nirjon, jeremy.gummeson, dan.gelb, kyu-han.kim}@hp.com

ABSTRACT

This paper presents *TypingRing*, a wearable ring platform that enables text input into computers of different forms, such as PCs, smartphones, tablets, or even wearables with tiny screens. The basic idea of TypingRing is to have a user wear a ring on his middle finger and let him type on a surface – such as a table, a wall, or his lap. The user types as if a standard QWERTY keyboard is lying underneath his hand but is invisible to him. By using the embedded sensors TypingRing determines what key is pressed by the user. Further, the platform provides visual feedback to the user and communicates with the computing device wirelessly. This paper describes the hardware and software prototype of TypingRing and provides an in-depth evaluation of the platform. Our evaluation shows that TypingRing is capable of detecting and sending key events in real-time with an average accuracy of 98.67%. In a field study, we let seven users type a paragraph with the ring, and we find that TypingRing yields a reasonable typing speed (e.g., $33 - 50$ keys per minute) and their typing speed improves over time.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithm, Design, Experimentation

Keywords

Wearable, Typing, Ring

1. INTRODUCTION

As computing systems evolve, so do their input methods. With advancements in computing technology, different forms of text input methods have been proposed and used in practice. These forms include the standard QWERTY keyboards for PCs, alphanumeric keypads and small keypads in earlier mobile phones, and on-screen soft keyboards in modern smartphones and tablets. Each of these input methodologies for typing in text has been invented out of the need for a change as the form factor and the mobility requirements of these devices have changed. We are now at the forefront of technology where wearable computers, such as smart watches and smart bands, have entered the consumer market. These devices have even smaller screen sizes and none of the existing typing methods are viable for these devices. A quick fix to this problem has so far been in the form of speech-to-text or shared keypads. However, the core problem has still remained unsolved, i.e. *there is no typing accessory that is portable and usable with computers of all form factors and mobility requirements.*

To meet this need we have created TypingRing, which is a wearable keyboard in the form factor of a ring. A user wears the ring on his middle finger and types in text with his three fingers (the index, the middle and the traditional ring finger) on a surface, such as – a table, his lap, or a wall. The user types and moves his hand as if there is an invisible standard keyboard underneath his hand. By moving the hand horizontally and vertically, TypingRing identifies one region to another on the imaginary keyboard. Further, by pressing one of his three fingers, the user types in the key. By using the embedded sensors surrounding the ring, TypingRing determines what key is pressed by the user completely inside the ring and then sends the key event to a remote computer over the Bluetooth LE network. TypingRing implements the standard BLE keyboard protocol so that it can be used with commercially available computing devices, such as – PCs, smartphones, tablets, or even wearables with tiny screens that support an external BLE keyboard. A piece of software running on the computing device intercepts the key events and provides a visual feedback to the user by highlighting a key or a portion of a custom on-screen keyboard, as the user moves his hand on the surface and types in keys.

Several salient features when combined together make TypingRing unique of its kind. First, TypingRing being a wearable device, is mobile and portable. The ring comes handy in scenarios where a quick and on-the-go text input is needed or scenarios when an alternative input method is not convenient, e.g. devices with tiny screens. Second, TypingRing is fast and highly accurate in detecting keys, and it performs all its computations inside the ring – without requiring any computational support from a more capable de-

vice. Third, TypingRing is multi-platform. Because of its adoption of standard BLE keyboard protocol, TypingRing is usable with any computing device that supports an external BLE keyboard. Fourth, typing with TypingRing is intuitive and it is easy to learn. TypingRing breaks down the task of typing on a standard keyboard into two intuitive tasks, i.e. moving a hand on a surface and then pressing a finger, which require little or no practice to get started with. Fifth, TypingRing is flexible and extensible. It is not tied to English alphabet or any specific keyboard layout. By changing the mapping between a position and a key, TypingRing is usable with keyboards of different layouts and dimensions.

TypingRing brings both engineering and computational challenges in front of us. The hardware architecture of TypingRing is designed to obtain the relative movements of the finger, and horizontal and vertical motions of the hand on a surface, so that thus-obtained data can be used to infer the position of the hand and typing gestures from just a single finger. To realize this, we embed a tiny microcontroller, an accelerometer, multiple line sensors, an optical displacement sensor, and a BLE chip on the perimeter of a circular ring platform. These sensors are read by a software running inside the ring, which detects and classifies typing gestures using an offline trained Hidden Markov Model (HMM) classifier. As an additional feature in TypingRing, we have implemented a simple Naïve Bayesian classifier to infer 3D gestures, such as - pitch, roll, and yaw, and map them to commonly used keys on a keyboard to offer shortcut keys to the user.

We have created a prototype of TypingRing using off-the-shelf sensors and an open source miniature hardware platform called TinyDuino [11]. In order to, tune various parameters of the system and train the typing and gesture classifiers, we perform an empirical study involving 18 users who uses the ring to type in letters, words, lines, and gestures while we store all the raw sensor readings. Based on this empirical data, we measure execution time, energy consumption, and the accuracy of typing and gesture classifiers. Our empirical evaluation shows that TypingRing is capable of detecting and generating key events in real-time with an average accuracy of 98.67%. Finally, we perform a field study, in which, we let seven users type a paragraph with TypingRing and we find that TypingRing yields a typing speed of $0.55 - 0.81$ keys per second, and their typing speed improves over time.

The contributions of this paper are the following –

- We introduce TypingRing, which is a wearable, portable accessory device that allows a user to input text into mobile and non-mobile computers of different forms.
- We describe a Hidden Markov Model (HMM) based typing gesture recognizer that uses acceleration, optical displacement and proximity sensors to infer the typing finger in real-time and with an average accuracy of 98.67%.
- We perform a field study by letting seven end users type a paragraph with TypingRing and we find that TypingRing yields a typing speed of $33 - 50$ keys per minute, and their typing speed improves over time.

2. USING TYPING RING

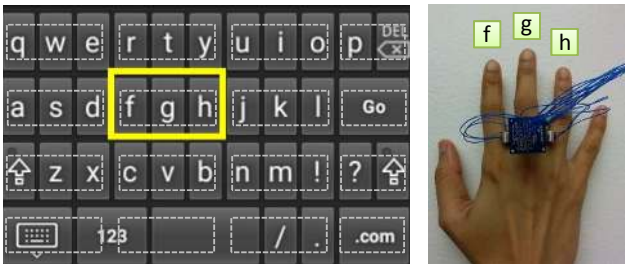
This section describes the working principle of the TypingRing along with some potential use cases.

2.1 Working Principle

TypingRing is worn on the middle finger of a user. As the user rests his ring on a horizontal surface, three consecutive keys on the on-screen keyboard of the computing device are highlighted. By using embedded sensors surrounding the perimeter of the ring, the TypingRing detects typing gestures made by the user's three fingers – middle, index, and traditional ring fingers. To highlight a different set of keys, the user simply drags the ring up, down, left or right on the surface. As long as the user is seeking a key or typing it in, visual feedback is provided to him on the screen of the computing device.

TypingRing assumes that the standard keyboard layout is divided into multiple *zones*. A zone is defined as a sequence of consecutive 3 keys on the same row on a keyboard layout. With this definition of a zone, the task of locating a key becomes a two-step problem: first, to identify the zone of the intended key, and second, to identify the key within that zone.

By moving the ring horizontally and vertically on a surface, a user moves from one zone to another. The user is given visual feedback by either highlighting the 3 keys on the zone on a soft-keyboard or just showing 3 keys on the screen when the computing device has a limited screen space. Once a zone is selected, each of the three fingers is associated with one of the 3 keys in that zone. To type in a specific key, the user makes a typing gesture using the corresponding finger. The ring detects the finger and sends the key associated with it to the computing device over the wireless channel.



(a) 3-Letter Zones (b) Typing Ring

Figure 1: Working principle of TypingRing.

Example. In Figure 1(a), we show sixteen zones on an Android keyboard layout marked with dotted rectangles and the selected zone `fgh` with a solid rectangle. Only one zone is allowed to be active at a time. A large key (e.g. the space bar) may be a part of multiple zones and the last zone on a

row may not have all three keys. To type in a letter, e.g. `W`, the user at first moves his ring up and left to select the `QWE` zone, and then makes a typing gesture with his middle finger to input the key.

2.2 Usage Scenarios

TypingRing may be used with a variety of computing devices that support standard external wireless keyboards. The list includes desktops, laptops, tablets, smartphones, and smart watches. However, below we list several compelling usage scenarios, where TypingRing is a more convenient choice than other input methods (e.g., on-screen keyboard).

- **Devices with Tiny Screens.** Some computing devices, such as smart watches and smart wristbands, have very small sized screens where a full scale touch enabled keyboard is not an option. TypingRing can be used with these devices as it physically separates the actual typing action from the visual feedback, and hence, the keyboard layout can be scaled down enough to just to show a tiny keyboard with highlighted keys, or a single row of keys, or just the 3 keys on a selected zone.
- **Saving Screen Space on a Mobile Display.** Typical on-screen soft keyboards on a mobile device block out more than 40% of the display. This is annoying to the user as the blocked out area may contain information that the user needs to see while typing in his inputs. With TypingRing, the size of the blocked area is reduced up to 10 times, and hence, the freed space can be utilized by the application to improve user experience.
- **Quick and On-the-Go Typing.** In some situations, e.g. self checking-in at an airport kiosk, making transactions at ATMs, or inputting pass codes into locks, we want to input text quickly and on-the-go. Health conscious people who want to avoid touching the keypads on these public devices might want to use their personal mobile device for input. The TypingRing being a wearable device, is more convenient in these scenarios than pulling out a mobile phone from the pocket and interacting with the display.

3. SYSTEM OVERVIEW

This section overviews the system architecture of TypingRing. We defer the algorithms and implementation details to subsequent sections.

3.1 The Hardware Architecture

TypingRing hardware platform is a redesign of conventional rings that has embedded sensors surrounding its perimeter and a tiny micro-controller unit having wireless data communication capability. The platform enables continuous sensing of user's hand and finger movements and processes the sensing data on-board to generate the key events. Figure 2 is a schematic of the TypingRing– showing the components and their placements on the ring.

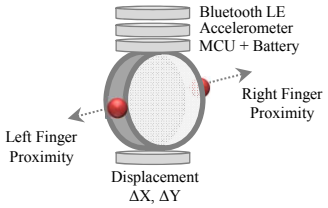


Figure 2: Hardware architecture of TypingRing.

- **Microcontroller.** The top of the ring, where usually a jewel is placed, consists of three boards stacked on top of each other. One of these is a micro-controller unit (MCU) that operates all the components, processes sensing data to determine key events, and transmits the key information. The MCU is powered by a thin film battery.
- **Accelerometer.** A 3-axis accelerometer is placed on top the ring, and it primarily detects the movement of the middle finger and helps detect other fingers. This sensor is kept always on to detect the presence of motion and turn on and off other sensors as needed.
- **Proximity Sensors.** Two proximity sensors are placed on the sides of the ring facing the two fingers which are next to the middle finger. Their placement allows the ring to measure the proximity between the middle finger and the two fingers next to it. This is used to detect which finger is used for typing.
- **Displacement Sensor.** An optical displacement sensor, like the ones used in an optical mouse, is placed underneath the ring to detect the X and Y displacements of the ring. This is used to detect when the user changes his typing zone.
- **Bluetooth LE.** A Bluetooth LE chip is connected on top of the ring, which is used by the MCU to send key events wirelessly to the computing device using the standard BLE keyboard protocol.

3.2 The Firmware Architecture

The TypingRing determines the key events completely inside the ring without relying on the computing device for any computational help. This makes it a self-sufficient input accessory just like a regular Bluetooth LE keyboard. An alternative to this would be to transmit raw or partially processed sensing data and let the device determine the key events. This, however, would require higher bandwidth communication, increase the data transmission cost, and make the ring a non-standard input accessory. The firmware running inside the TypingRing is responsible for controlling the sensors, determining the key strokes, and generating and sending key events to the computing device.

Figure 3 shows the architecture of the firmware. Figure 3 shows the architecture of the firmware. As shown in the figure, the firmware is composed of three layers: sensing, recognition, and communication.

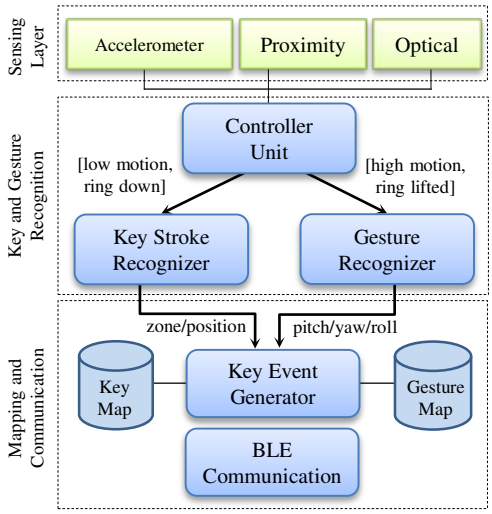


Figure 3: Firmware architecture of TypingRing.

Sensing Layer. The top layer of the firmware contains sensor sampling modules, which are the accelerometer, the left and right proximity sensors, and the optical displacement sensor modules. These modules read and hold sensor values in bounded circular FIFO queues.

Key and Gesture Recognition Layer. The `Controller Unit` is responsible for turning on and off a particular type of sensor. Since sampling the accelerometer is relatively inexpensive, only the accelerometer is continuously monitored. If it detects a significant motion, it turns on the optical sensor to check if the ring is lifted up or is on a surface, and triggers either the key or the gesture detection module.

The `Key-Stroke Recognizer` kicks off when the user's hand movement is low to moderate and the ring is on a surface. It takes into account all three types of sensors to determine the zone or the finger using the algorithm in Section 4. The `Gesture Recognizer`, on the other hand, starts when the ring is lifted up and the user's hand movement is relatively higher. Using the algorithm described in Section 4.4, TypingRing detects simple 3D gestures (e.g. pitch, roll, and yaw) to enable short-cuts for commonly used keys.

Mapping and Communication Layer. The `Key Event Generator` translates the detected zone, finger or the 3D gesture into appropriate key events according to a predefined mapping.

- A reported typing gesture by the `Key-Stroke Recognizer` is mapped into a real key event corresponding to the finger in the currently selected zone.
- A reported change of zone by the `Key-Stroke Recognizer` is mapped to a 'fake' key event, such as an unused `ALT` + a specific key, and is sent to the computing device to enable visual feedback to the user. Such a fake key event is ignored by the OS of the computing device, but is useful to the TypingRing system, as this is used to determine which zone to highlight (Section 5.3 describes this in detail).
- A reported 3D gesture by the `Gesture Recognizer` is mapped to a commonly used key. For example, pitch is mapped to space bar, double pitch to enter, yaw to delete, and roll to shift.

4. KEY STROKE RECOGNITION

This section describes the typing finger recognition algorithm for key strokes along with a simple gesture recognition algorithm for shortcut keys.

4.1 The State Machine

As described earlier, typing with the ring involves two phases – seeking for a zone followed by making a typing gesture. To enable this, the ring maintains a state machine which is shown in Figure 4. The transitions between states happen at some specific events (shown in brackets) which are determined by processing the sensor streams. Below we describe the states, the events and transitions.

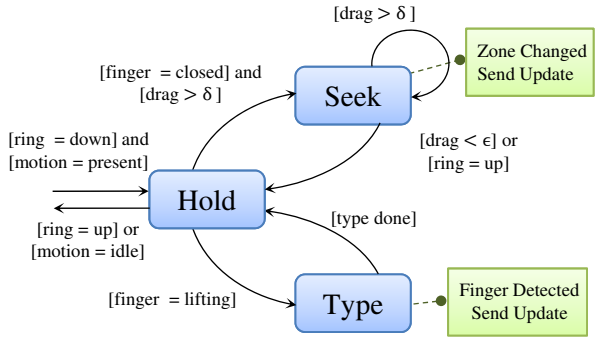


Figure 4: The state machine for key stroke detection.

Once the `Key-Stroke Recognizer` is activated, the ring stays in one of its three states: `hold`, `seek`, or `type`.

- **Hold.** This is both the start and the final state of the state machine. This state is entered when the ring is on a surface (`ring = down`) and motion is detected (`motion = present`). From then on, this acts as the intermediate state between the seeking and the typing phase. The ring leaves this state when it is lifted up (`ring = up`) or is idle (`motion = idle`).
- **Seek.** The `seek` state is entered when the user starts dragging the ring on the surface (`drag` > δ) while keeping his fingers close to each other (`finger = closed`), i.e. the user is not making any typing gestures. Since a user may have to cross multiple zones until he reaches the desired one, the seek state repeats itself. Every time there is a change in zone, the information is sent to the key event generator. The ring leaves this state when it is lifted up (`ring = up`) or no significant movement is detected (`drag` < ϵ).

- **Type.** The `type` state is entered when the user lifts up one or both of his fingers (`finger = lifting`) indicating an intention to type. This state runs the typing gesture detection algorithm (Section 4.3) to determine a valid typing gesture and the finger used, and then goes back to the hold state. In case of a valid gesture, it sends the detected finger to the key event generator.

4.2 Detecting Sensor Events

All the state transitions in the state machine depend on the values of four types of event variables: ring, motion, drag, and finger.

- **Drag Events.** The `drag` event relates to the dragging of the ring on a surface. These events are triggered whenever the displacement of the ring over a period crosses a certain threshold. A large drag results in a change of zone whereas the ring leaves the seek state when there is no significant drag. The value of a drag is calculated from the optical displacement sensor's readings. The optical sensor periodically sends updates on X- and Y-displacements of the ring as a vector, $\Delta d = (\Delta x, \Delta y)$. These values are too noisy and are hardware accelerated. To compute a drag, Δd is first passed through a moving average filter and then integrated continuously. Transition from one zone to another happens when the integrated value exceeds an empirically obtained threshold δ_d. Equations 1 and 2 show the filtering and drag computation steps.

$$\Delta d = \alpha\ \Delta d_k + (1-\alpha)\ \Delta d_{k-1} \tag{1}$$

$$\text{drag} = \left| \int \Delta d\, \mathrm{d}t \right| \tag{2}$$

- **Finger Events.** The `finger` events happen whenever the user lifts up one of his fingers or presses down both of his fingers. These events are determined by two identical detectors, one for each finger. Each of these detectors accumulate proximity values, $\{p_k\}$ over a period of τ_p and compute the range. The range is compared to two empirically obtained thresholds δ_p and ϵ_p to detect finger lifting and pressing events.

$$\text{range} = \max_{t-\tau_p \le k \le t} \{p_k\} - \min_{t-\tau_p \le k \le t} \{p_k\} \tag{3}$$

- **Motion Events.** The presence or absence of `motion` is determined from the 3-axis accelerometer sensor readings (a_x, a_y, a_z). The ring keeps track of running variances of the magnitude of acceleration $\sqrt{a_x^2 + a_y^2 + a_z^2}$ over a period τ_a, and based on a threshold, ϵ_a, it determines the presence or absence of motion.
- **Ring Events.** The `ring` events, i.e. whether or not the bottom of the ring is touching the surface, are read directly from the optical sensor. The optical sensor has a built-in lift detector, which is programmed to detect up to 5 mm lifting.

4.3 Detecting the Typing Finger

A naive algorithm to detect the typing finger is to use thresholds to detect finger events similar to Section 4.2 and thereby identify the finger. However, we empirically found that such an algorithm does not work in practical cases and results in false positives (multiple detections of one or more fingers), false negatives (no detection), and does not provide any confidence on the inference. By employing conservative thresholds false positives may be eliminated, but this induces delays in typing gesture detection and frustrates the user as unlike zone seeking there is no visual feedback on how much additional effort is needed to exert to type in the key.

In order to enable a robust and accurate finger detection, we leverage an observation that – each finger's movement has an effect on the accelerometer readings obtained from the middle finger – either directly (for the middle finger) or indirectly (for the other two). With this observation, we design an algorithm that takes both the proximity sensors and the accelerometer into account. Hence, the problem of typing finger detection is stated as – *given periodic proximity and accelerometer sensor readings over an interval, what is the probability of making a typing gesture with one of the three fingers?*. If the highest probability for a finger is greater than a threshold, the finger is reported by the algorithm.

Because the algorithm runs inside the ring, the solution has to meet some additional requirements. The algorithm needs to be simple enough for implementation inside the ring's microcontroller, and also has to be fast, accurate, and robust. To achieve this, we use a N-state Hidden Markov Model (HMM). This satisfies our goals as HMMs are known to be robust classifiers for time series signals [25] and they provide a confidence value of the solution (i.e. a probability for each class), and a trained HMM is compact enough to store inside the ring and is fast enough to obtain the probabilities in real time. The steps of the algorithm are as follows:

- **Quantization**. The left and right proximity sensor values, p_i^L and p_i^R, and the accelerometer reading $\mathbf{a_i}$, are first quantized using Q' and Q'' functions, respectively.

$$Q'(p_i) = \underset{1 \le k \le L}{\operatorname{argmin}} \left| q'_k - p_i \right| \tag{4}$$

$$Q''(\mathbf{a_i}) = \underset{1 \le k \le L}{\operatorname{argmin}} \left| q''_k - V(||\mathbf{a_i}||) \right| \tag{5}$$

  where, $V(||\mathbf{a_i}||)$ is the variance in acceleration over a second, L is the number of quantization levels, and q'_k and q''_k are the quantization levels for proximity and accelerometer sensors, respectively. To limit the number of possible states in the HMM, we use $L = 3$ level quantization, and the levels correspond to the quartiles of the empirical data.

- **Encoding.** At every sampling interval, the quantized values from the 3 sensor streams produce a 3-tuple, $(\rho_i^l, \rho_i^r, \alpha_i)$, $1 \le \rho_i^l, \rho_i^r, \alpha_i \le L$. Considering all possible combi-

nations and 3-level quantization, there can be at most $L^3 = 27$ different tuples. Each of these tuples is considered as an observed event (called an *emmision*) in the HMM, resulting in a set of possible emissions, $\{Y_i\}$, $1 \leq i \leq 27$, at each state of HMM.

- **HMM Classification.** Assuming we have a precomputed HMM corresponding to sensor data obtained by typing with a specific finger, this step computes the likelihood of a given sequence of emissions, $y = (y_1, ..., y_T)$ being generated by the model. The length of the sequence T depends on the sampling frequency f_s and the duration of a typing event W. In TypingRing, we use a sliding window of $W = 1$ second (with 50% overlap) to capture a typing event. This bounds the value of T to $\lceil W/f_s \rceil$. Using Viterbi [14] algorithm, we compute the maximum likelihood of y given the precomputed HMM – having a state space S, initial probabilites of each state $\pi_i, 1 < lei \leq |S|$, and transition probabilities $u_{i,j}$ of transitioning from state i to j – as follows:

$$v(1,i) = p(y_1|i).\,\pi_i \tag{6}$$

$$v(t,i) = p(y_t|i).\max_{s \in S}(u_{s,i}.\,v(t-1,s)) \tag{7}$$

where, $v(t,i)$ is the probability that the most probable state sequence that could emit y ends in state i. Now, to obtain the maximum likelihood of our HMM to generate y, we apply the following:

$$v_{ML} = \max_{1 \leq i \leq |S|} v(T,i) \tag{8}$$

This step is performed once for each of the 3 HMMs corresponding to typing with 3 fingers and the model that has the maximum v_{ML} is classified as the most likely finger. The complexity of this algorithm is $O(T * |S|^2)$.

In order to train the HMMs, we use our empirical data to estimate the parameters of these HMMs offline. At first, all labeled training examples – containing the proximity and accelerometer sensor readings for typing with one of the three fingers – are quantized and encoded to form a time series of emissions. We truncate each instance of typing data to 1 second by removing the leading and trailing no-motion values. This makes the length of each encoded training example exactly $1/f_s$ elements long, which is used in classification. Using the encoded training examples for each finger, we use the standard Baum-Welch [25, 29] algorithm to find the unknown parameters of the HMM. These models are stored inside the ring and the ring only runs the classification step in real-time.

4.4 Detecting Gesture Shortcuts

Gesture recognition from motion data is a well-studied problem and there are many efficient and highly accurate algorithms that detect a wide variety of gestures from accelerometer readings obtained from mobile and wearable devices. We have adopted one of the simplest of them in TypingRing that uses an offline trained Naïve Bayesian classifier to detect the gestures from 3-axis accelerometer readings from the middle finger. TypingRing recognizes three simple 3D gestures - pitch, roll, and yaw, and maps them to six commonly used non-alphabetic keys on a keyboard. In general, the mapping could be anything, but in our implementation, we have done it as shown in the Table 1. To support more than three keys, we consider repetitions of the same gesture within a small period of time (2 seconds) as two different keys.

Gesture	Repetition	Key
Pitch	1	Space Bar
	2	Enter
Roll	1	Shift
	2	Caps Lock
Yaw	1	Delete (letter)
	2	Delete (word)

Table 1: Gesture shortcuts in TypingRing to enable faster entry of common keys.

We use an offline-trained Gaussian Naïve Bayesian classifier that uses the variance in each of the three axes of acceleration as a feature. The steps that run inside the ring for real-time gesture classifications are the following-

- **Windowing.** The ring periodically samples the accelerometer and accumulates 1 second worth of 3-axis accelerometer readings, $a^i = (a^i_x, a^i_y, a^i_z)$. This window of 1 second is shifted by 0.5 second to form the window for the next iteration.
- **Feature Computation.** We compute a 3-element continuous valued feature vector, $f = (f_1, f_2, f_3)$, where the elements are the variances of $\{a^i_x\}$, $\{a^i_y\}$, and $\{a^i_z\}$, respectively.
- **Classification.** To determine the most likely class, v_{NB} for the feature vector f among the classes, $C = \{pitch, roll, yaw\}$, we use Equation 9:

$$v_{NB} = \underset{c \in C}{\operatorname{argmax}} \prod_{i=1}^{3} \frac{1}{\sqrt{2\pi}\sigma^i_c} \exp\left\{ -\left(\frac{f_i - \mu^i_c}{2\sigma^i_c} \right)^2 \right\} \tag{9}$$

where, μ^i_c and σ^i_c are the mean and standard deviation of the feature f_i for the class $c \in C$ which are empirically determined in our system.

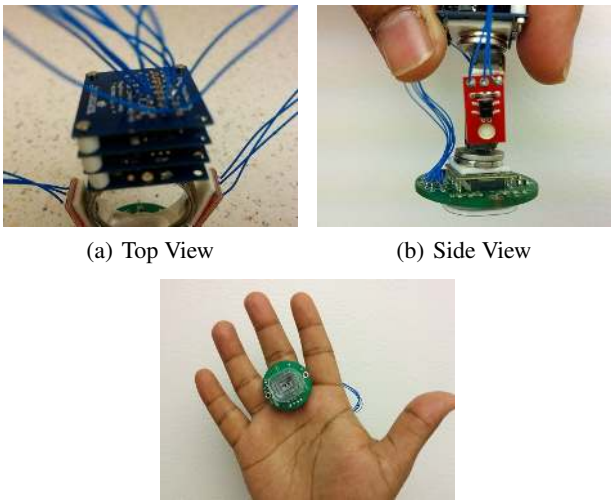
We use this model as our empirical observation reveals that, for each gesture, the features – i.e. the variance in acceleration along each axis – closely follow normal distributions. The algorithm is also highly efficient as these variances are calculated as part of motion detection, hence no extra computation is required for feature extraction. Given a fixed set of classes the classification step is also essentially a constant operation and runs very fast.

5. SYSTEM IMPLEMENTATION

This section describes the implementation details of our TypingRing prototype. We describe the hardware, communication between the ring and the computing device, and the software that enables visual feedback.

5.1 Hardware Implementation

The hardware is comprised of a total of seven sensor boards surrounding all four sides of a circular titanium ring. Figure 5 shows the top view, side view, and the bottom view of the TypingRing prototype.



(a) Top View (b) Side View

(c) Bottom View

Figure 5: The top view, side view, and bottom view of the TypingRing.

The top of the ring consists of a stack of four TinyDuino [11] boards. TinyDuino is an open source platform that features the full capability of the popular Arduino platform but miniaturizes the board to make its size smaller than a quarter. Each board has dimensions of 20 mm $\times$ 20 mm, and the stack of four boards has a height < 15 mm. Of the four boards, the bottom one contains an Atmel ATmega328P MCU (8MHz, 32KB Flash, 2KB RAM, 1KB EEPROM) and is powered by a coin cell battery. The board on top of it is a Bosch BMA250 3-axis accelerometer shield, which connects to the MCU using the I2C bus. The third board contains a low energy Bluegiga BLE112 [1] Bluetooth module, which is used to send the key events over the air. The top is an extension board of the MCU, containing ten digital and four analog IO pins.

The two sides of the ring, facing the index finger and the traditional ring finger, have two identical QRE 1113 IR line sensor boards attached to them. Each of these boards contains an IR emitting LED and an IR sensitive photo transistor and has an optimal sensing distance of 3 mm. At the bottom of the ring, there is an ADNS 9800 optical motion sensor board. This sensor is used by the ring to measure the X and Y displacements when the user drags the ring on a surface. The motion sensor communicates to the MCU over the standard SPI interface.

5.2 Ring to Device Communication

The Bluegiga BLE112 Bluetooth low energy system on chip (SoC) is used for communications. A Tinyduino microcontroller communicates with the SoC using the BGLib API to configure the hosted radio stack. After the user presses a button, the SoC begins sending undirected, connectable advertisement beacon packets that contains the name "Typing Ring"; additionally, a field indicates the ring should have the appearance of a keyboard-based human interface device (HID).

After scanning for devices, a remote device will see "Typing Ring" appear in a list of available devices and has the option of connecting to it. During the connection process, Android's Bluetooth Low Energy stack scans the remote device's global attribute table (GATT) and determines that the device is implementing the HID over GATT profile [2]. Once this discovery process completes, the remote Android device can receive keystroke events generated by the ring.

In order to send a keystroke to the remote device, the ring encodes key presses as HID reports that each consist of 8 bytes. The first byte indicates a modifier value (i.e. Shift, Alt, Control), the second byte is reserved, and the remaining 6 bytes contain slots that indicate individual characters that are in the *key down* state. Communications between the ring and the remote device are achieved by sending a key down event, where a single slot in an HID report is populated by a character, and optionally the modifier field. After transmitting the key down event, a null report where all bytes are set to 0x00, indicate that the key(s) have been released from their down state.

Two types of HID reports are sent to the remote device. The first type indicates values that indicate the visual feedback a user receives should be updated. As the user moves their hand around on the surface, the position on the virtual keyboard changes in terms of the row/column position. These movements are encoded as non-visible keyboard values that make use of the identifier slot: ALT+1-4 indicate the column of the currently highlighted zone, and ALT+6-9 indicate the row of the currently highlighted zone. In total, 4 HID reports are used to indicate an update to the current row/column position – 2 key down reports and 2 key release reports.

Additionally, when a virtual key is pressed, the user uses one of their middle 3 fingers on the ring worn hand. The index finger is encoded as ALT+Q, the middle finger as ALT+W and the ring finger as ALT+R – this is used as additional feedback to highlight a particular key in the currently highlighted zone. The second type of HID report that is sent consists of the characters themselves. Each HID report contains a single key value in byte position 3, while the modifier is either set to 0x00 – no modification, or 0x02 – indicating capital letters with the left shift modifier.

5.3 Enabling Visual Feedback

We have created an Android keyboard application to run on the user's device and provide visual feedback for the ring. It functions similarly to a traditional Android software keyboard but has additional functionality to provide ring related feedback. It can be used as a traditional on-screen software keyboard when the ring is not connected. When the ring is in use with the app the ring transmits three types of messages. All the messages are sent as Bluetooth keyboard HID events. The first type of message is used to indicate the current zone. This is used to draw a box around the currently active zone to the user has visual feedback on what keys finger typing gestures will activate. The second message type is to indicate a finger press event and which finger was pressed. This is used to draw a circle around the pressed key to give visual feedback that a press was registered. These messages are based as ALT-key modifiers of keys that have no meaning in Android. The ALT-key messages are intercepted by the keyboard app and used to update the visual feedback as shown in Figure 6. The final message type is the actual key character event generated by a press event. This is sent as a standard key event so that the ring could be used for typing with any device that supports Bluetooth keyboards or if the keyboard app was not running. Figure 6 shows the appearance of the visual feedback for a normal sized Android keyboard when a key has been recently pressed. The circle around the pressed key serves as visual feedback that the key press event was detected.

Figure 6: Visual Feedback for Key Press

Our technique can be easily adapted for devices with very limited screen area, such as wearables including watches or eyeglass displays. In such devices displaying a full keyboard is undesirable since it can consume a significant portion of the display, hiding valuable visual information from the user. In these situations we can display only a very limited keyboard representation showing only the active region. This is illustrated in Figure 7. As the user moves their hand and changes the active zone the micro-keyboard is updated. The figure shows a region in a keyboard with the typical QWERTY layout, but other layouts including a simple alphabetically ordered keyboard can also be used if desired.

Figure 7: Minimal Keyboard for Small Screen Devices

6. EVALUATION

In this section, we describe three types of experiments. First, we measure the execution time and energy consumption of various components of TypingRing. Second, we perform an empirical study to select various parameters and to evaluate the performance of the key-stroke detection algorithm. Third, we perform a user study by letting 7 users type with the ring and summarize the findings.

6.1 System Measurement

The TypingRing is a self-contained system where all the computations – from sensing to key stroke detection and then sending the key events to the computing device – happen inside the firmware of the ring. Understanding the execution time and the energy consumption of its component is important to gauge its responsiveness and lifetime.

6.1.1 Execution Time

We measure the execution time of all major components of the system software. Since the timer provided by the Arduino platform is not precise, we use a more reliable approach to measure the execution time using a Saleae Logic16 high-sensitivity logic probe [8]. Prior to measuring the execution time, we instrument the code by enclosing it inside two instructions – one that writes a logic `HIGH` to an unused digital pin of Arduino and another that writes a logic `LOW` to high. The digital pin is monitored by the logic probe which samples the pin at 100 MHz – giving us a 10 ns resolution in time measurement. We cross-check measured execution times using Arduino's `micros()` API which has a μs level resolution and have found that the measurements are close to each other when rounded to the nearest ms.

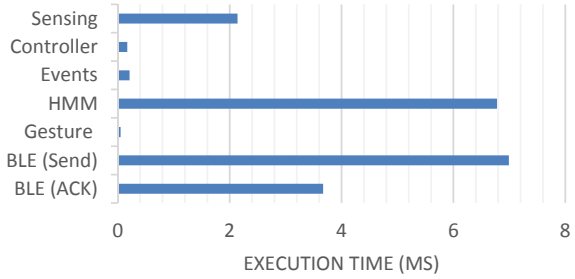


Figure 8: Execution time of different components in TypingRing.

Figure 8 shows the measured execution time of seven major components of the TypingRing system software which includes both computation and communication modules. Among

all the computational modules, the HMM classification task takes the highest amount of execution time of 6.75 ms, while combining all other tasks the total computation time still remains < 10 ms. The BLE (Send) of 6.99 ms denotes the duration between time of issuing a send command and the instant when the key is actually transmitted by the Bluetooth LE chip. The BLE (ACK) of 3.67 ms denotes the time it takes to get an acknowledgment back from the computing device, after a key has been sent. Overall, the system has an end-to-end execution time of about 20 ms from making a typing gesture to getting back an acknowledgment. Considering the 100 ms sensor sampling interval of TypingRing, this indicates that the system is capable of detecting and sending a key in real-time.

6.1.2 *Energy Profile*

We create an energy profile of our prototype and estimate its lifetime. We identify the states of the system and the components (e.g. processor, sensors, and BLE) that consume power in each state. Using the estimated power of each component and the duration of each state from our empirical data, we obtain an energy profile that is shown in Figure9.

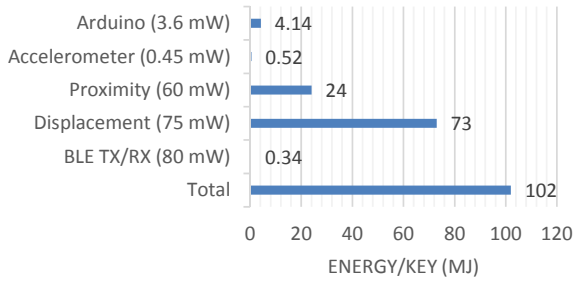


Figure 9: Energy profile of TypingRing.

We show the energy consumption per key in this figure. The labels on the left show the hardware components and their power consumption when they are active. From our empirical data, we get the fraction of time each of these components are active when typing a key and by multiplying the power to an average key stroke length, we obtain the energy values. For example, BLE TX/RX power in general is about 80 mW. However, when we consider idle time and data exchanges separately, the average power to maintain connection and send one key becomes less than 0.3 mW. Similarly, not all sensors are active at all states - e.g. proximity sensors are not used during seeking, optical sensors are not using during typing, and accelerometer is used in hold and gesture states. With this energy profile and a coin cell battery of $125mAh$, TypingRing's life-time is about $13,650 - 15,500$ key events. Assuming approximately 1s per key stroke, the lifetime of TypingRing is about $3.8 - 4.3$ hours.

6.2 Empirical Evaluation

In this section, we evaluate various aspects of the key stroke recognition algorithm. First, we determine the parameters of the sensor event detectors that have been introduced in section 4.2. Second, we evaluate the accuracy of the HMM based finger detector and compare its performance with two other baseline classifiers. Third, we evaluate the accuracy of the gesture recognition algorithm.

6.2.1 *Empirical Dataset*

We have created an empirical dataset that we obtained by letting 18 users (4 females, and 14 males) use the ring. The users were shown randomly generated characters and randomly selected text from a standard phrase set [23] on a mobile device. They typed the text – at first, using the TypingRing, then on the touch-based soft keyboard of a mobile device, and then by clicking on a Windows 7 on-screen keyboard. Each user typed in about 50 random characters, $5-15$ phrases, and about 30 gestures. The users were given visual feedback on the screen by showing the input text, the highlighted zone, and the typed text. In order to enable visual feedback, we used a bootstrap classifier that is trained on a singe user prior to the data collection. We programed the ring to sample and store accelerometer, proximity, and optical sensor readings at 100 ms interval. The collected data were analyzed offline in a desktop computer.

6.2.2 *Drag Events*

The drag event depends on two parameters – the smoothing factor (α) and the drag threshold (δ_d) which we determine in this experiment.

The smoothing factor is used to reduce the variance in optical sensor reading prior to integration. However, the required amount of smoothing depends on the surface material on which the sensor is being dragged. Hence, we conduct an experiment to understand the relationship between the quality of a surface and the amount of variance in displacement sensor reading on that surface. We leverage this relationship in TypingRing to tune in the smoothing factor (α) based on the surface on which the user is typing.

The ADNS 9800 optical sensor comes with a register that contains the amount of high-quality optical features it used to compute the displacement. This gives us an indication of the quality of the surface underneath the sensor. In our experiment, we collect displacement sensor readings from various types of surfaces, such as – wooden table, wall, plastic, white board, various types of fabrics, and paper. As read by the ADNS 9800, these materials have a surface quality value ranging from 15 to 75. On each surface, we drag the ring horizontally, vertically, and diagonally for about 3 feet, at a speed of approximately 4 inches per second, and then measure the variance in displacement.

From Figure 10 we see that there is a linear relationship between the surface quality and the variance in sensor readings. We leverage this information to obtain the smoothing factor α for a particular surface. We do the mapping by first computing α for a white paper surface so that the zone tran-

sitions are smooth. For other surfaces, we scale the value of α according to its relative surface quality with respect to white paper. This technique makes the zone transition consistent across different types of surface materials.

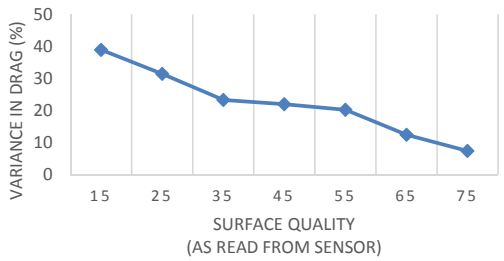


Figure 10: Impact of surface quality on variance in data.

The drag threshold (δ_d) represents the total linear displacement reached when a zone transition happens. A smaller δ_d results in a quicker zone transition and vice versa. We perform an experiment to quantify this inverse relationship between the drag threshold and the zone transition rate. Figure 11 shows that as the total displacement (shown in units of optical sensor reading) increases from 60 to 270, the ring updates the zone at the rate of 200 to 50 times per minute.

In TypingRing, we adopt an adaptive thresholding scheme for δ_d depending on the state of typing. For an initial zone transition, we use a larger threshold to make certain that the user really wants to move and it is not just casual hand motion. For successive zone transitions, a smaller threshold is used so that the zone transitions are smoother.

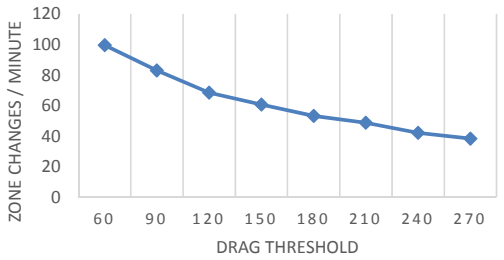


Figure 11: Adaptive thresholding for smooth zone transitions.

6.2.3 *Finger Events*

The finger lift and press events depend on the range of proximity sensor values. Two thresholds δ_p and ϵ_p are used to detect these two types of events. Both of these thresholds are chosen so that the false positives, false negatives, and the overall error in event detection are minimal. As an illustration of this, we describe how δ_p is determined.

Each typing gesture made by either the left or the right finger in our dataset contains exactly one finger lift event and one finger press event. For a given threshold, based on the type of finger and the number of times the time vs. proximity curve crosses the threshold, we determine whether it contributes to false positives or false negatives, or if it is accurate.

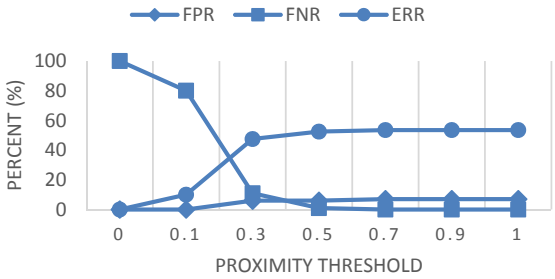


Figure 12: Finger threshold to reduce false positive/negatives and overall error.

Figure 12 shows the false positive rates (FPR), false negative rates (FNR), and the error in detection (ERR) of the finger lift events, as the threshold δ_p is varied. A smaller δ_p increases the false positives and larger ones tend to increase both the false negative rate and the overall error. We choose $\delta_p = 0.25$ to ensure negligible false positives as this is more problematic than the other two.

6.2.4 *Motion Events*

The presence or absence of significant motion is detected from the variance in acceleration. To obtain a threshold to distinguish motion and non-motion, we divide our empirical data into two subsets – one containing examples that are collected when a user is typing or seeking a zone, and the other one containing examples when the user's hand is resting and idle. Figure 13 shows the box plot of variance in acceleration for these two cases. We see that the two classes are clearly separable with a small threshold between 20 – 50.

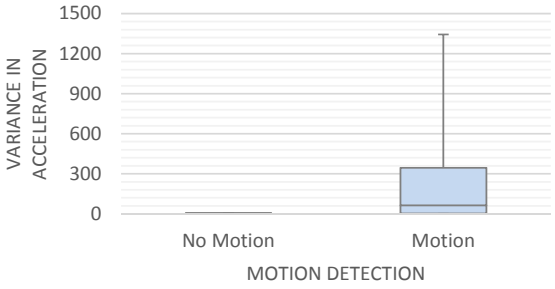


Figure 13: Variance in acceleration for motion detection.

6.2.5 *Typing Finger Detection*

The goal of this experiment is to evaluate the accuracy of the HMM based classifier that determines which of the three fingers a user used to make a typing gesture. The HMM is trained and tested on the empirical dataset that contains a to-

tal of over 2500 letters (including phrases). Each letter in the alphabet is typed in at least 75 times, and each of the three fingers has been used at least 650 times. Letters in a phrase are isolated by noting the duration in zones, and sensor values for each letter are stored separately. The training phase of HMM is run for 1000 iterations and the initial values of transition and emission matrices are randomized. Because there is randomness involved, the experiment is repeated 10 times to increase the confidence. On each run, 70% of the examples are used for training and the rest are used for testing. The accuracy is measured by calculating the percentage of correct predictions among all test cases. The accuracy of the algorithm is compared with the accuracy of two other baseline classifiers: a decision tree classifier and a Naïve Bayesian classifier. Both of these use the same set of features – which are the quantized proximity and accelerometer readings.

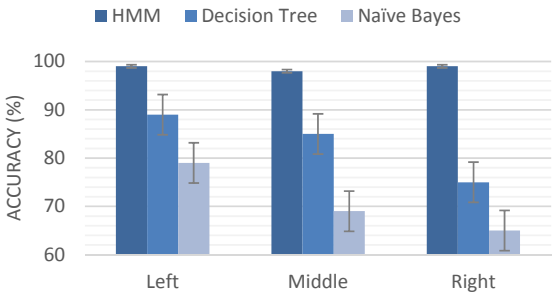


Figure 14: Accuracy of typing finger detection.

Figure 14 compares the accuracy of the three algorithms in detecting the typing finger. Of the three, HMM performs the best with an average accuracy of 98.67%, which is 15%–27% higher compared to the other two. The main reason for HMM to perform better than the other two is that the HMM considers the sequence of states and each state is formed by taking all combinations of the three sensors into account. The other two classifiers are fundamentally extreme in this issue. The Naïve Bayesian classifier assumes independence of all three sensor types – which is not true in our problem. The decision tree classifier, on the other hand, assumes that all variables interact with each other – which is also not quite true in TypingRing, e.g. the left and the right proximity sensors are mostly independent of each other. The decision tree also suffers from over fitting the training data, which results in better training accuracy, but because of their poor generalization ability, the cross validation accuracy remains lower than the HMM.

One design decision in HMM is to select the number of hidden states. For our problem, we empirically determine this by varying the number of states from 2 to 5, and then comparing the accuracy. Figure 15 compares the classification error, false positive rate, and the false negative rate of four HMMs. The number after the HMM denotes the number of states. We observe that a 3 state HMM is the best

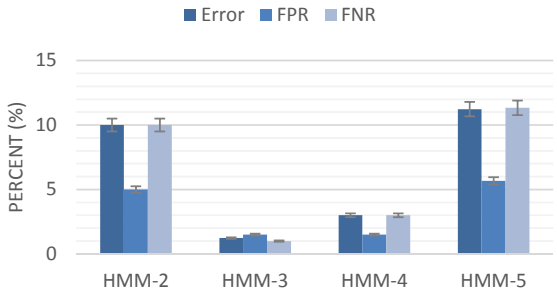


Figure 15: Performance of different sizes of HMMs.

among the four having the lowest error rate of 1.3% with less than the 1.5% false positive and false negative rates. Compared to other models, this is 2.4−9.13 times lower. HMM-3 is better than HMM-2 as it is more expressive and capable of encoding more information. HMM-4 and HMM-5 however is not better than HMM-3, because of their tendency to overfit the training data and then failing to recognize unknown examples from the test set.

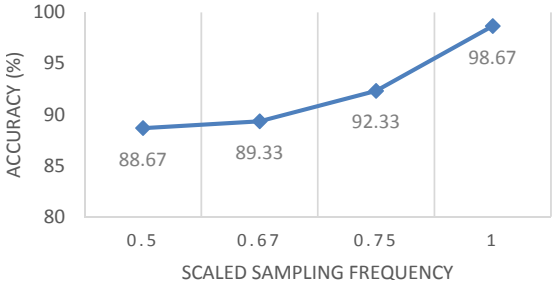


Figure 16: Effect of sampling frequency.

While detecting the typing finger, we use a sampling period of 100 ms for all three types of sensors. This results in a very high accuracy in finger detection of over 98%. A higher frequency does not have enough room for further improvement, but we wanted to determine the lowest sampling frequency for which our algorithm still performs reasonably well. To do this test, we down-sample the sensor streams to lower the sampling frequency to $\frac{3}{4}$, $\frac{2}{3}$, and $\frac{1}{2}$ of the original frequency and then evaluate the accuracy of HMM using these modified samples. Figure 16 plots the results. We see that we could potentially double the sampling interval, but in that case we have to sacrifice about 10% accuracy. Further investigation reveals that most of the misclassifications at the lower frequencies come from the right finger (the traditional ring finger when worn on the right hand) detection errors. This is due to the fact that humans have limitations in lifting up the traditional ring finger higher than other fingers. This results in higher false positives and false negatives unless the right proximity sensor is sampled at a higher rate. We leave this as a future work to enhance the efficiency of the ring by suitably choosing the sampling intervals for each

sensor individually.

6.2.6 *Gesture Detection*

We perform an experiment to determine the accuracy of the gesture recognizer. Our participants perform about $25-35$ gestures of each type, i.e. pitch, roll, and yaw, which results in a sample size of over 500 gesture instances. This dataset is used to train the 3-class Gaussian Naïve Bayesian gesture recognizer in TypingRing. We use randomly chosen 70% examples for model training and use the rest for testing. The experiment is repeated 10 times.

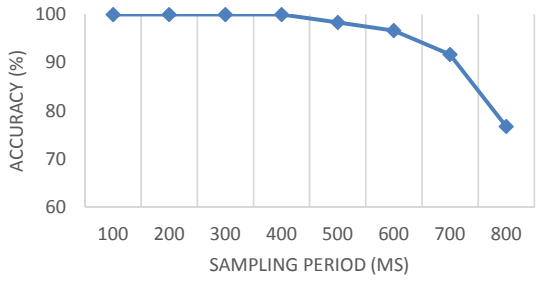


Figure 17: Accuracy of gesture recognizer.

Figure 17 shows the accuracy of gesture recognizer for various sampling intervals of the accelerometer. During data collection, we set the sampling interval to 100 ms. During our analysis, we down-sample the accelerometer readings to vary the sampling frequency and compute the accuracy of the classifier for different sampling intervals in the range $100-800$ ms.

We observe that up until 400 ms sampling interval, the classifier's accuracy remains 100%, and it starts to make mistakes afterward. This indicates that when the ring is lifted by the user to make a gesture we may use a larger sampling interval to save some CPU cycles. However, this does not help much in practice as gestures in TypingRing are for shortcut keys with long intervals and last for a short duration. In applications where short-cut keys may be used for a longer duration, e.g. pressing only the 'next', 'cancel' or 'okay' button at an interactive kiosk, this principle could be applied.

6.3 User Study

We performed a user study involving 7 users (2 females and 5 males). The differences between this experiment and our empirical data collection experiments were that during the user study the system was trained on empirical data and computations happened inside the ring. We gave each user a paragraph printed on a piece of paper to type using the TypingRing. Users were asked to type the entire paragraph correctly – i.e. they were allowed to use the delete key in case of an error. Each user participated in the study in two sessions. In order to compare the performance of TypingRing, we used two baseline solutions. The first one is an on-screen soft keyboard of an Android smartphone, and the second one is a Windows 7 on-screen keyboard where a user types by clicking a mouse.

6.3.1 *Typing Speed*

Figure 18 plots typing speed in terms of number of typed keys per second for each user, for each of the three key entry methods. We order the users according to their typing speed with TypingRing. We observe that, the typing speed on a soft keyboard (shown as Soft KB) is the highest with an average speed of 1.63 keys/sec. This is about 2.4 times higher than TypingRing. This is somewhat expected as our users are highly experienced with touch-screen keyboards on mobile devices. A major reason of this gap in speed is the amount of hand movements in each technique. In case of a soft keyboard, fingers can be directly targeted to a key, but in case of TypingRing, a user needs to seek a zone and then press the key. TypingRing's speed, however, is comparable to that of a mouse based on-screen keyboard. For some users, e.g. U1, U5, U6, and U7, TypingRing's speed is as close as $0.92-1.0$ times and on average the ratio is 0.88.

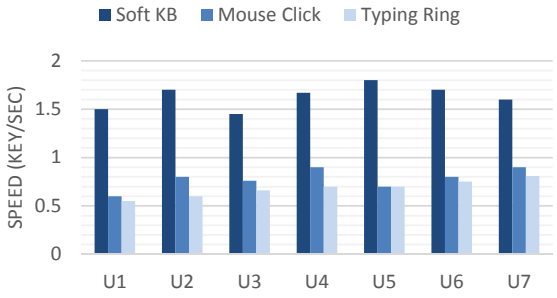


Figure 18: Typing speed of our participants.

6.3.2 *Learning Effect*

As TypingRing is a new input technique, it requires some practice to get used to it. This is evident when we compare a user's typing time in two separate sessions, which is shown in Figure 19. For the first 4 users, we see a dramatic reduction in typing time by $4.5-8.6$ times. Further investigation reveals that, for the most part, the improvement comes from the reduction in seek time. This is intuitive since zone seeking is more time consuming than making typing gestures and it also requires some practice to coordinate between ring movement and visual feedback. From our experience with end users we noticed that those who took time to practice before telling us that they are ready for the session were the ones who achieve the best results with TypingRing.

6.3.3 *User Survey*

At the end of the sessions, we ask our participants some questionnaire to understand their satisfaction on various aspects of the system. We ask them how they feel about the size, weight, portability, usability, and visual feedback. We

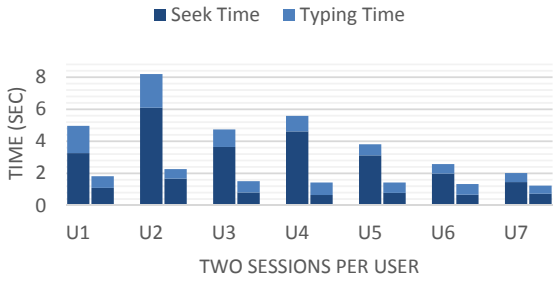


Figure 19: Learning effect in TypingRing.

also ask them if they notice any difference in their own performance between two sessions, and their feeling about the ring to becoming a replacement for other text input methods on mobile computers. From their responses, some unanimous things are noticed, e.g. everybody seems to love the visual feedback, felt that the system is easy to use, and with practice they are doing better. For other issues, such as about the size, weight, and portability, we have got mixed responses. Several of them think that it is fine for typing, but they might not want to wear it all day long. Some of them complain about the size as it is larger than a typical ring. Others who have used a smart ring before, are somewhat neutral on this issue. Most of them would like to see this replace the on screen keyboard so that they get more screen space which is a limitation of current mobile devices. Overall they gave the ring a rating of 7.6. We note their feedback and plan to improve the usability of the ring in our next iteration which we keep as a future work.

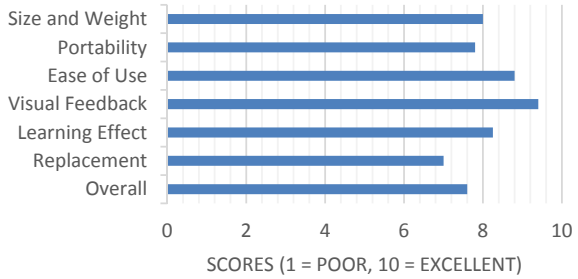


Figure 20: Summary of user ratings.

7. RELATED WORK

Ring-based wearable devices. Several ring-based wearable devices have been proposed for variety of purposes. [16] proposes an energy harvesting wearable ring platform for gesture input on surfaces. Ring [7] is proposed for text recognition, gesture detection, and payment. Fin [3] is another gesture detecting ring that a user wears on his thumb and then makes gestures by touching other fingers with his thumb. NFC Ring [5] uses passive near field communication to provide easy locking/unlocking of a door or a smartphone. ThumbTrack [10] is an ergonometric ring that performs the functions of a mouse. Similarly, a ring proposed in [21] uses a mouse optical sensor and an RGB camera to detect finger movements as well as surface types / textures. SmartyRing [9] pairs with a smartphone and shows useful information, such as notifications, triggers camera, tracks phone, and shows time. In contrast, TypingRing is designed specific for a text-input with all types of computer devices.

Wearable input devices. Other than the ring-form factor used in our work, there have been various wearable formfactors used for new input methods. Keyglove [4] is a wearable glove that places 37 touch sensors on the palm of a glove. Using the combination of these touch sensors, it is possible to input text on to a computer. [15] uses sensors worn on each finger as a PDA input device, while our system uses only one finger. Virtual keyboard [6] is another wearable keyboard with high detection accuracy achieved by using artificial neural networks. It requires a user to wear the device on both hands, while TypingRing requires only one finger.

Gesture recognition with various sensors. Gesture recognition as an input method has been extensively investigated. uWave in [21] uses dynamic time warping (DTW) techniques to classify movements of a hand in free space as 1 of 8 different gestures. Work in [26] is similar to uWave, but it uses a HMM approach to classify 5 different gestures. Authors in [12] use an accelerometer to detect characters and words drawn in the air. Recently, authors in [27] used a RF signal and NFC receiver to detect a gesture in air and translate it to characters. While these gesture recognition work is exciting, TypingRing is different in that it focuses on both key events and hand gestures.

Keyboards for mobile devices and surfaces. Various types of keyboards have been proposed for mobile devices and surface environments. Recently the use of acoustic sensors for keyboard has drawn significant attention. Ubik [28] is a portable text-entry method that allows a user to make keystrokes on conventional surfaces, e.g., wood desktop, by extracting location-dependent multi-path fading features from the audio signals. This however relies upon the dual-microphone interface on a smartphone and not suitable for other devices or surfaces. Stane [24] is a synthesized surface that makes particular sounds and facilitates tactile interactions based on the sounds. [17] uses a microphone on a stethoscope attached to different surfaces to interpret gestures based on sound alone. In [20], identifiers are carved into a surface similar to a barcode. A microphone listens to a fingernail move through the notches to determine the identifier. [22, 18] use microphones inside the surface to distinguish between different types of finger strikes (e.g., Knuckle vs. pad of finger). [13, 19] combine acoustic sensors and sounds from the body (e.g., bone movement) to detect movement/gesture of finger or location of taps. TypingRing uses off-the-shelf motion and proximity sensors without requiring any external computing entity or synthesized surfaces.

8. CONCLUSION

This paper describes the design, implementation, and evaluation of a ring-based wearable platform, called TypingRing that enables text inputs into computers of different forms and mobile requirements. TypingRing employs embedded sensors to detect the position of a hand and typing gestures made by a finger, and infers the key using a HMM classifier. Evaluations on empirical data and field tests show that the ring detects and reports keys in real-time, achieves over 98% accuracy, yields a typing speed of up to 0.81 keys per second, the typing speed improves as a user uses it more. While TypingRing is capable of inputting text into any computer that supports a BLE keyboard, it's typing speed is still not comparable to that of a regular keyboard or on-screen soft keyboards. However, in some scenarios where quick text input is needed or other methods are inconvenient, TypingRing presents a viable solution.

9. REFERENCES

[1] Bluegiga technologies. `bluegiga.com/`.

[2] Bluetooth Developer Portal. `developer.bluetooth.org/`.

[3] Fin. `www.finrobotics.com`.

[4] Keyglove. `keyglove.net`.

[5] NFC Ring. `nfcring.com`.

[6] Project Virtual Keyboard. `www.senseboard.com/`.

[7] Ring: Shortcut Everything. `www.kickstarter.com/projects/1761670738/ring-shortcut-everything`.

[8] Saleae Logic Probe. `saleae.com/logic16`.

[9] Smarty Ring. `smartyring.com`.

[10] ThumbTrack. `mindstreaminc.com`.

[11] Tiny Duino. `tiny-circuits.com/`.

[12] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter. Using mobile phones to write in air. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 15–28. ACM, 2011.

[13] T. Deyle, S. Palinko, E. S. Poole, and T. Starner. Hambone: A bio-acoustic gesture interface. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 3–10. IEEE, 2007.

[14] G. D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[15] M. Fukumoto and Y. Tonomura. "body coupled fingerring": wireless wearable keyboard. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 147–154. ACM, 1997.

[16] J. Gummeson, B. Priyantha, and J. Liu. An energy harvesting wearable ring platform for gestureinput on surfaces. In *MobiSys*, pages 162–175. ACM, 2014.

[17] C. Harrison and S. E. Hudson. Scratch input: creating large, inexpensive, unpowered and mobile finger input surfaces. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 205–208. ACM, 2008.

[18] C. Harrison, J. Schwarz, and S. E. Hudson. Tapsense: enhancing finger interaction on touch surfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 627–636. ACM, 2011.

[19] C. Harrison, D. Tan, and D. Morris. Skinput: appropriating the body as an input surface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 453–462. ACM, 2010.

[20] C. Harrison, R. Xiao, and S. Hudson. Acoustic barcodes: passive, durable and inexpensive notched identification tags. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 563–568. ACM, 2012.

[21] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[22] P. Lopes, R. Jota, and J. A. Jorge. Augmenting touch interaction through acoustic sensing. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 53–56. ACM, 2011.

[23] I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 754–755. ACM, 2003.

[24] R. Murray-Smith, J. Williamson, S. Hughes, and T. Quaade. Stane: synthesized surfaces for tactile input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1299–1302. ACM, 2008.

[25] S. M. Ross. *Introduction to probability models*. Academic press, 2014.

[26] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14. ACM, 2008.

[27] J. Wang, D. Vasisht, and D. Katabi. Rf-idraw: virtual touch screen in the air using rf signals. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 235–246. ACM, 2014.

[28] J. Wang, K. Zhao, X. Zhang, and C. Peng. Ubiquitous keyboard for small mobile devices: harnessing multipath fading for fine-grained keystroke localization. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 14–27. ACM, 2014.

[29] L. R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4):10–13, 2003.