GesturePod: Programmable Gesture Recognition for Augmenting Assistive Devices

Shishir G. Patil Don Kurian Dennis Chirag Pabbaraju Rajanikant Deshmukh
Harsha Vardhan Simhadri Manik Varma Prateek Jain
Microsoft Research India
{t-shpati, t-dodenn, t-chpabb, v-rades, harshasi, manik, prajain}@microsoft.com

ABSTRACT

UPDATED—May 31, 2018. We have designed a GesturePod that converts any white cane into an interactive cane that can enable easy access to smartphones and other home devices for people who are visually impaired (VI) and the elderly. Users can control devices by performing programmable gestures on the cane. For example, a user can answer an incoming call with a double swipe.

Our GesturePod is a lightweight, non-intrusive device built from inexpensive Inertial Measurement Unit (IMU) sensors and a battery-powered Cortex M0+ microcontroller. A key technical contribution of this work is the development of a robust real-time gesture recognition system based on Machine Learning (ML) models that can be deployed on tiny microcontrollers.

We also developed an Android App which pairs with GesturePod and makes it easier and faster to complete frequent phone tasks like answering a call or reading notifications. A study with 12 VI participants demonstrated that with just 10 minutes of training, users were able to perform gestures with >90% accuracy and complete some of the tasks 9 times faster.

Author Keywords

Gesture recognition; Machine learning; Accessibility; Microcontrollers.

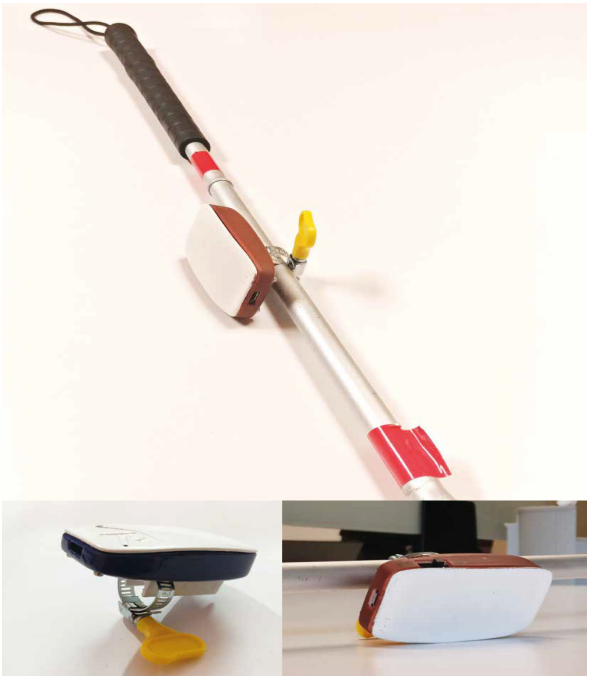

Figure 1. GesturePod on a white cane, pod can be attached to any cane and can be detached at any time.

INTRODUCTION

Smartphones and other home devices such as Amazon Echo are now an integral part of work and personal life. Such devices have a potential to positively impact lives of people with visual impairment, the elderly, and others who might have trouble interacting with common computing devices. A glimpse of some of the exciting possibilities is demonstrated by a few recent apps such as Seeing.AI [20] and Eye-D [24].

While smartphones apps and other devices can open up new scenarios for visually impaired (VI) persons, in many cases a more fundamental problem is that of accessing these devices in the first place. This issue is more pronounced in certain mobile settings or constrained environments where it might be difficult for VI people to access/locate the phone (see Figure 2). Moreover, due to ambient noises and for privacy reasons, using voice commands may not be preferred. Finally, our informal interviews and existing studies [12] suggest that UI for mobile apps and other devices can be cumbersome to use for a VI person.

In this paper, our goal is to help the visually impaired and the elderly access smartphones (or other connected home devices such as TV, microwave) without the need for buying new devices or disrupting their normal routine. VI persons, especially in mobile settings, are recommended to carry a white-cane (http://whitecaneday.org). Therefore, we focus our efforts to transform the cane itself into an accessible input device so that it can interact with connected devices - i.e., an interactive cane. An interactive cane alleviates the issue of introducing completely new hardware (such as wristband) or the need to teach an entirely new technology. Moreover, since the cost of the device can be critical, especially in developing nations [22] where disability is highly correlated with poverty, the guiding principle behind our design was enabling ease of use

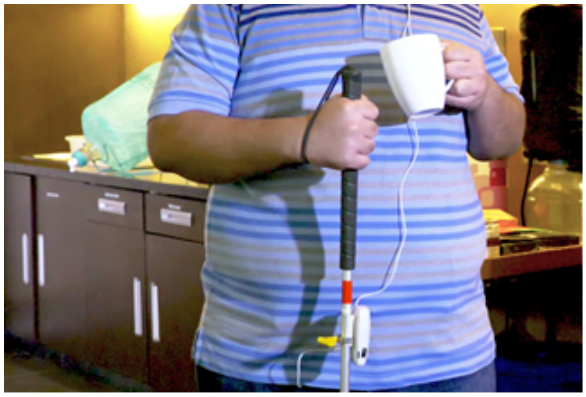

Figure 2. A Visually Impaired (VI) person in constrained environment, with both hands occupied.

and applicability to a large number of scenarios without incurring significant overhead in terms of training, device cost, and deviation from regular usage.

An approach to designing such an interface would be to augment the cane with buttons that can control the phone [2]. However, buttons are not easy to access in general, especially if the user tends to hold the cane in different grips and different orientations. Moreover, one cannot add more than a couple of buttons without the clutter that could counteract the functionality. Our interviews with VI users suggests the same, as most VI users indicated a preference for solutions other than buttons (Table 4, APPENDIX I)

Therefore, we develop a gesture-based solution for the interactive cane. We propose to control the phone (and other home devices) using certain programmable gestures on the cane. Natural gestures can significantly reduce the training overhead and can provide an intuitive experience to users. For example, a double tap of cane can pick up an incoming phone call. Moreover, gestures can allow for much more nuanced tasks- for example, a fast twist of cane vs slow twist of cane can enable faster scrolling of certain phone screen vs slower scrolling. Finally, gestures can be combined with buttons to give a more wholesome user experience.

To this end, we hand-designed the GesturePod that can be clamped onto any cane and weighs less than 50 grams, thus not impacting the usability of the cane. Our pod is powered by a rechargeable lithium-ion battery, contains an ARM M0+ class microcontroller which is attached to an inexpensive off the shelf accelerometer and a gyroscope along with a Bluetooth Low Energy (BLE) module for communication with the phone. Our gestures trigger certain specific signature in accelerometer/gyroscope readings, that can be used to detect the gesture.

Recognizing gestures of the cane is a challenging task and requires carefully designed ML algorithms to account for real-world uncertainties. For example, a double tap of the cane on carpet might product completely different signal than a double tap on a hardwood floor. Similarly, variations of the gestures exist across different users as well. Hence, simple rule-based solutions might not suffice for robust detection of gestures. However, ML algorithms and the requisite featurization of data tend to be computationally expensive and hence are difficult to deploy on tiny microcontrollers housed by GesturePod. See Section System Design for more details on how we overcame these challenges, while respecting the unique resource constraints posed by an embedded device like GesturePod.

Our current solution supports five gestures of cane: double tap, double swipe, twirl, left twist, right twist; (see Figure: 4) for an illustration of the gestures. We selected these gestures based on user feedback about ease of gestures, their distinguishability from regular cane usage, and ease of their detection using our ML algorithms.

We conducted an exploratory user study with about 12 VIPs to understand the pros and cons of our cane. Our user study suggests the following: a) our gestures are natural and easy to learn, b) the GesturePod is accurate in detecting gestures across users and across settings (it detects gestures with more than 92% accuracy), c) certain critical tasks can be performed faster by using GesturePod. For example, in our study, GesturePod decreased the average time required to enquire the current location on a smartphone by a factor of 9 and sped up the task of answering an incoming call by a factor of 3.

Roadmap: In Problem Formulation section, we formulate the problem, discuss key challenges and provide an overview of our solution. In the section on System Design, we present design of GesturePod, the workflow used to develop the predictive model for gesture recognition and discuss how we deployed the ML algorithm on GesturePod. Finally, in the Experimental study section, we describe our exploratory user study and our empirical results.

RELATED WORK

The white cane is one of the most important accessibility device for people with visual impairment and naturally has been subject of several recent studies. [26] presented a comprehensive qualitative study of the various navigation difficulties faced by VI and outlined various approaches for accessibility researchers. [10] also provided design guidelines for smart-cane designers and highlighted critical issues with electronics-based canes such as battery life, reaction time, floor-level etc.

Most existing electronic cane-based solutions enhance the cane with obstacle detection capabilities. A good example here is the SmartCane [25] which uses an ultrasound device to detect nearby obstacles. Similarly, EYECane [9] uses a camera to detect obstacles while [21] uses a LIDAR and accelerometer to detect obstacles. In another line of works, [3, 5, 15] designed canes which use RFID tags to determine obstacles and can also read out certain other information. Our solution is complementary to such obstacle detection solutions and can be potentially combined with such solutions to provide easier and safer access to devices. [2] represents one of the most related work to GesturePod as it proposes a button based solution for the cane. In particular, the proposed cane houses a “home" button as well as four arrow buttons to control the phone. As mentioned in the previous section, our studies suggest that users typically don’t prefer button based solutions

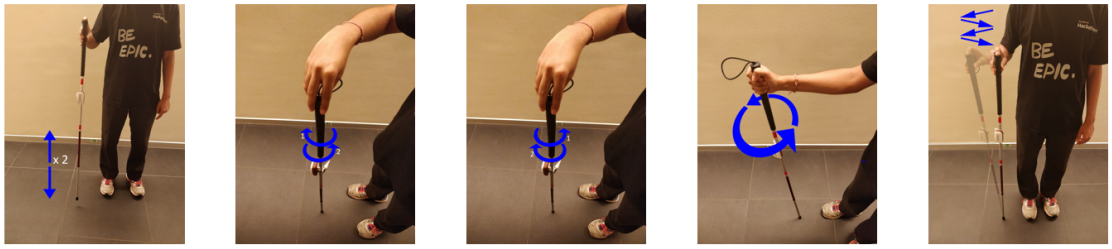

Figure 3. Illustration of various gestures: a) double tap: hit the floor twice with the cane, b) right twist: twisting the cane to right, c) left twist, d) twirl: make a circle in air with cane's grip, d) double swipe: tilting the cane to the right twice.

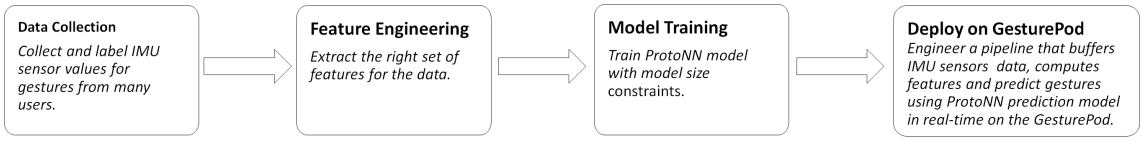

Figure 4. Flow chart for training and deploying cane-gesture recognition model.

as they are unnatural to use and might require changing how one uses a cane. Moreover, buttons can potentially be used as a complementary approach to our technique, where our gestures can be used to perform certain critical and low-latency budget tasks like picking up phone call, while buttons can be used for other tasks like navigating over all apps or controlling a relatively obscure app.

Gesture and activity recognition is an extensively studied problem and Inertial Measurement Unit (IMU) sensors like accelerometer and gyroscope have been successfully used for detecting gestures in a variety of applications [1, 4, 7, 11, 16, 19]. For example, [18] used the accelerometer readings from a watch, transmitted them to an Android tablet and ran a Naïve Bayes classifier on the data for recognizing hand gestures. [17] employed a k-Nearest Neighbor (kNN) on electromyography and accelerometer signals received via Bluetooth on a Nokia phone to remotely control mobile devices. [8] designed a Magic Ring, which was to be worn on a finger and had an accelerometer module, and would detect static predefined finger gestures like Finger Up, Finger Down, etc. using hand-tuned rules.

While gesture recognition has been studied extensively, most of the existing solutions either: a) rely on hand-tuned rules for a small number of gestures/activities like running, sleeping [8], b) apply simple ML algorithms that can handle simple gestures in restricted settings [17], c) use a powerful device to detect gestures/activities via computationally expensive ML algorithms [16].

The above-mentioned solutions do not apply to our problem setting as we need to enable reasonably complicated gestures that do not misfire during regular cane usage. Moreover, the solution should be robust enough to handle a variety of users, grips, and different environments like floor type etc. Finally, due to weight, battery and latency constraints, we are required to predict gestures on a low-powered microcontroller without any external compute help. To the best of our knowledge, our method is the first gesture detection system that can detect complicated gestures accurately and robustly despite access to tiny computing devices.

Recently, there have been a few ML algorithms proposed which can produce tiny but effective ML models. Examples include SNC [14], BNC [27], NN-prune [23], BONSAI [13], ProtoNN [6] etc. In this work, we use ProtoNN algorithm which compresses standard k-Nearest Neighbor (kNN) algorithm to decrease both the model size as well as prediction time. Similar to kNN, ProtoNN can also learn non-linear and complicated decision boundaries. However, ProtoNN still requires "featurized" data and does not perform well on raw sensor values. Hence, designing a "featurizer" for raw sensor values is equally critical to the success of the entire system.

PROBLEM FORMULATION

The technical goal is to program a microcontroller to use IMU sensors' data to detect cane-gestures accurately and in real-time. Our Gesture-Pod contains an IMU sensor that reads out values every 5 milliseconds, i.e., data is sampled at 200HZ. Each data sample consists of six values - acceleration and angular velocity along X, Y, and Z axis. That is, we get data of the form: $d_1, d_2, \ldots, d_t, \ldots$, where each data point $d_t \in R^6$ contains the six numbers corresponding to acceleration and angular velocity along each axis.

For predicting gestures, we use a sliding window-based approach, where we fix a window size ω and stride-length ψ and predict gestures in each window see Figure 8. That is, we make a prediction for gesture in data points D_i, and then stride by ψ time-steps:

$$D_i = [d_{(i-1)\cdot\psi+1}, d_{(i-1)\cdot\psi+2}, \ldots, d_{i\cdot\psi+\omega}] \quad (1)$$

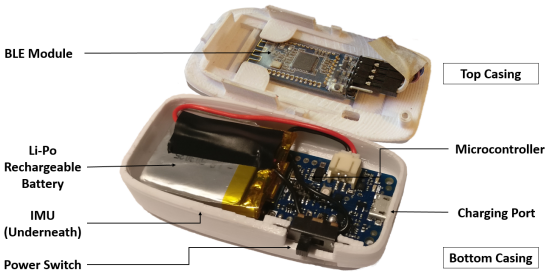

Figure 5. Internals of the GesturePod. It houses Arduino MKR1000, IMU sensors, Li-Po battery, and a BLE module. See Figure 6 for a picture of all the individual components.

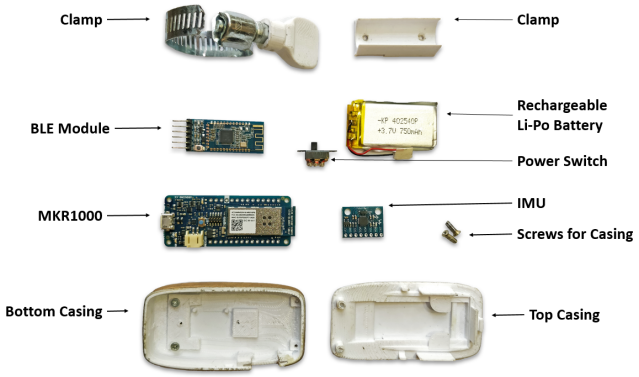

Figure 6. Components that make up the Gesture-pod.

i.e., data point D_i is $6 \cdot \omega$ -dimensional. The goal is to find a function f that maps D_i into a C -dimensional vector, i.e.,

$$f\colon R^{6\cdot\omega} \to \{0,1\}^C \tag{2}$$

where C is the number of gestures, all-zero vector denotes no gesture, i.e., regular usage. In this paper, we will focus on the following 5 gestures:

1. Double tap
2. Right twist
3. Left twist
4. Twirl
5. Double swipe

(see Figure 4) for a description of how to perform the gestures. These gestures are complicated and require >500ms to complete, so we must set the window size to around 2 seconds. Moreover, to ensure that the gestures are captured accurately without overflowing the tiny data buffer on IMU, we stride windows by 100ms. This implies that a single thread on the microcontroller must poll the IMU data, compute the features, apply the ML model to make a prediction and communicate any intended gesture within the 100ms time budget set by the IMU buffer.

Naturally, the task is quite challenging due to these hard constraints on time and memory budget along with the richness in gestures. To the best of our knowledge gesture recognition problem has not been solved under such severe compute constraints for as complicated gestures as ours.

Next section introduces our system design and method to solve the above-mentioned gesture recognition problem.

SYSTEM DESIGN

We have developed a plug-and-play GesturePod that can be clipped onto any white cane. The pod is used for both collecting the labeled data necessary for training the ML model and for deploying the trained model to infer gestures on the cane. In this section, we describe (a) the electronics, (b) the methodology used for collecting data and training the ML model, and (c) the design and control flow of the inference pipeline.

Electronic Subsystem

The pod contains an inexpensive Inertial Measurement Unit (IMU), an ARM Cortex-M0+ microcontroller based Arduino MKR1000 board, a Bluetooth Low-Energy (BLE) module, a rechargeable battery and an on/off switch. This data is fed to the microcontroller which computes the features on a sliding window basis and predicts the gesture in real-time. The predicted gesture is then communicated to the smartphone connected to the BLE module. All the components are housed in a 76mm X 38mm X 25mm GesturePod as shown in Figure 5, with a clamp that can be attached on to most canes. The specific make of IMU used in our pod can buffer 1024 bytes of data. At a rate of 43 bytes for each instance, sampled every 5ms, the buffer can retain 23 instances - in other words, 115ms of data before the buffer queue overflows. To prevent data loss due to overflows, the microcontroller reads and clear the IMU buffer at least once every 100ms.

Data Collection for model training

The data required for training the predictive model was collected from sighted volunteers who were shown examples of the five gestures. To collect data, the pod was attached to the cane at a natural location for VI people. The data consists of raw sensor readings from the IMU sampled at 200Hz, i.e., one set of readings every 5 milliseconds. This is streamed to a PC through serial communication via the microcontroller.

Data Labeling For training our ML models, we require labeled data, i.e., we need to annotate all the intervals in which the user performed a gesture with the correct gesture. There are two possible ways to label sensor readings with gestures in the data collection phase:

1. Toggle Switch: A toggle switch controlled by the cane user. The user switches it on right before performing a gesture on a cane in the other hand, and switches it off immediately after.

2. Observer: A second observer watching the cane user marks the beginning and the end of gestures.

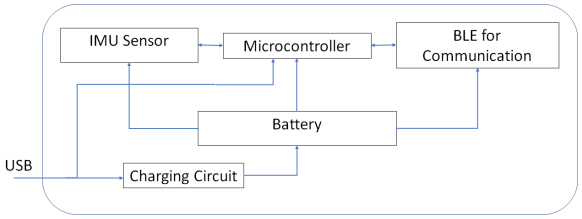

Figure 7. Functional block diagram of the plug and play device.

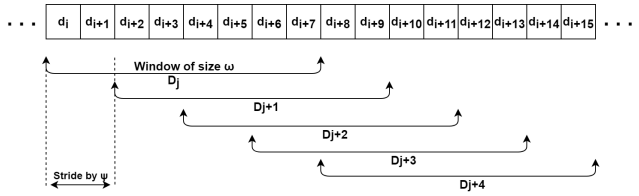

Figure 8. Sliding window with width 400 and stride length 20

However, we found both these to be inaccurate as it is hard to start and stop precisely at gesture boundaries. Hence, to further refine our approximate annotations from one of the above-mentioned techniques, we manually replayed the data on screen and refined the annotations so that (a) each gesture label spans an interval of 400 data samples, i.e., 2 seconds, and (b) the signature corresponding to the gesture is centered within the 2 second window.

The data for the gestures were collected from 7 sighted volunteers. Each user contributed at least 5 instances of at least 2 gestures to the dataset. The data includes variations in (1) flooring, (2) grip - technique of holding the cane, (3) orientation of the cane, and (4) handedness. We collected 102 double taps, 55 right twists, 54 left twists, 353 twirls and 83 double swipes.

For negative data, i.e., regular cane usage without any gestures, we asked the users to walk with the cane in their hand without trying to perform any of the five gestures. We collected about 8 minutes of such data and clipped it to generate two-second segments which are negative examples, i.e., example data points with no-gestures.

Although the positive examples described above were sufficient to generate a model that could recall gestures accurately, the negative examples were insufficient to prevent false positives. That is, due to a large amount of variation in regular cane usage, the model could misclassify regular cane usage as one of the five intended gestures, and hence needed further work to decrease the number of false positives. We will describe how we addressed this problem in Section Model Boosting.

ML Model Training Pipeline:

Data Collection for model training

The data required for training the predictive model was collected from volunteers who were shown examples of the five gestures. To collect data, the pod is attached to the cane at a location where one would expect a VI person to attach the pod to their cane. The data consists of continuous raw sensor readings from the IMU streamed to a PC through serial communication via the microcontoller. This data has no annotations about the intervals in which the user has performed a gesture. To make this data useful for supervised learning, it is necessary to identify the windows in which the gesture was performed and label the window with the right gesture. There are a couple of ways to add this annotation in the data collection phase:

1. A toggle switch controlled by the cane user. The user switches it on right before performing a gesture on a cane in the other hand, and switches it off immediately after.

2. Performing gestures on white-cane used by Visually Impaired(VI) persons to trigger actions on the phone.

The labeled data set can be accessed here.

Features

Typical ML algorithms require featurization of the data to perform accurate predictions. While Deep Learning based methods can learn features from the raw-data on its own, it is difficult to embed and execute such methods on tiny compute that our microcontroller is able to afford. In fact, even storing the raw data for a 2 second window itself is somewhat expensive as it would mean storing 400×6 integers which would occupy about 10KB of RAM.

Hence, in this work, we convert our raw data into a 124-dimensional feature vector that can be potentially computed with streaming data; for simplicity, we buffer the entire data for now. Our features consist of 120×6 $(= 120)$ features from the counts of acceleration and gyroscope values across 3 axes quantized into 20 equally spaced bins. Let $\psi(D_i)$ be the feature vector for $i-th$ data point D_i (see Eq 1) i.e.,

$$\psi(D_i) \in R^{124} \tag{3}$$

where $\psi(D_i)[1:20]$ is the count of $X-Axis$ acceleration values in 20 equispaced bins between -16384 to 16384. Similarly, $\psi(D_i)[21:40]$ represents the count of $Y-Axis$ acceleration values and so on.

However, the above-mentioned bin features lose all the phase information. For example, a left twist and a right twist differs only in the phase of gyroscope readings, resulting in similar values for bin features. To alleviate this concern, we introduce 4 additional features: the index and lengths of longest positive and negative sequence of values from the y-axis of the gyroscope (i.e. y-angular momentum). These features retain the phase information helping us discern gestures based on the angular movement of the cane. For identifying the longest positive sequence, we require each instance to be above a threshold of 0.62 (on a scale of 0 to 1). This filters out any noise that may affect the feature. Similarly, for a sequence to be considered negative it needs to be below a set threshold of 0.32 (on a scale of 0 to 1).

Model Generation

We feed our featurized and annotated training data into the ProtoNN algorithm [6] to train a model that accurately predicts one of the C gestures ($C = 5$ in our case). That is, we form the training data:

$$Z = \{(\psi(D_1), y_1), \ldots, (\psi(D_N), y_N)\}$$

where $y_i \in \{0,1\}^C$ is the training label vector. We randomly selected an equal number of points for each gesture and for no-gesture class. In all, we selected 1,25,513 training examples and also selected 31,378 examples for validation.

We recovered a 6KB predictive model by using the multi-class formulation of the ProtoNN algorithm [6] to train a model on the featurized data using the following hyperparameters: 10 projected dimensions, 4 prototypes per class using *per-class* -means prototype initialization, and 200 iterations. That is, we learn 20 prototypes $b_1, \ldots, b_{20}$, their corresponding label vector $z_1, z_2, \ldots, z_{20}$ where each label vector $z_i \in R^6$ and a sparse projection matrix $W \in R^{10\times 124}$.
The final prediction function is given by:

$$f(D_i) = \sum_{i=1} z_i \cdot exp(-\gamma||b_i - \Omega\psi(D_i)||_2^2) \qquad (4)$$

where $\gamma \geq 0$ is a hyperparameter and $||\alpha||_2^2 = \sum_j \alpha_j^2$.

Model Boosting

While the trained model recognizes gestures accurately, it suffers heavily from false positives - the model predicts a gesture even when no gesture is performed. This is not surprising as our training data might not cover a lot of tricky scenarios. For example, a single-tap of the cane is always a part of double-tap, and doesn't occur in our initial negative examples. Hence, naturally, the algorithm detects a double-tap whenever a single-tap type of gesture is performed leading to a false alarm. Similarly, half a twist of the cane or a single swipe of the cane also leads to a false alarm.

To solve this, we had to augment the negative example set in the training data with three sources of false positives: (1) Data obtained from participants walking, and performing activities they usually would - climbing a stair, strolling through the corridor etc.(2) *Partial* gestures, i.e., partial time-signatures of actual gestures, e.g. a single tap which is a part of a double tap and (3) any other stray activities where the model misfires a gesture such as throwing the cane in the air, holding it at a particular angle, etc.

After collecting data from the first scenario (day-to-day activities), we clipped only those windows that caused false positives and injected them as negative examples to keep the size of training data manageable. For scenarios (2) and (3), the data collected was already in the form of 2 second windows. By retraining the model by augmenting the dataset with these negative examples, significantly reduced false positives.See Table 5 for a confusion matrix of the learned model on 20% data sampled as validation set.

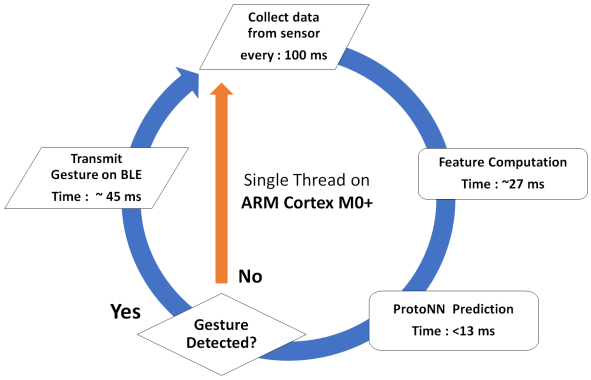

Figure 9. Machine Learning Inference flow-chart. The pipeline starts with buffering data from IMU sensors, forming data point D_i from sliding windows (Eq 1), computing their features using Eq 3 and then applying ProtoNN prediction function Eq 4 . If it predicts a gesture for 6 continuous time windows, then we transmit the detected gesture to the phone via BLE.

Prediction Pipeline

The ML inference pipeline on the cane consists of (a) data collection from IMU, (b) feature computation, (c) ProtoNN inference algorithm along with the model generated by the ProtoNN training algorithm and (d) BLE communication for relaying the gestures detected to the phone. Careful engineering was necessary to orchestrate all stages of the pipeline on a single-threaded microcontroller. Figure 9 represents the time allocated to each component of the pipeline.

At a high level, we first compute features from the data collected from IMU sensors, apply ProtoNN prediction function (Eq 4) and if a gesture is detected then communicate the detected gesture to the phone via BLE. The phone then takes appropriate action based on the gesture.

The microcontroller used in the pod does not have a native floating-point support and hence it takes about 5ns for one fixed point arithmetic vis-à-vis 50ns for a floating-point arithmetic. Hence to accommodate this, the normalized data is scaled from [zero to one] floating points to [one to hundred] integers. The microcontroller maintains a rolling window of width 400 instances. On every 20$^{\text{th}}$ new instance, we compute the feature vector for the previous 400 instances.

For prediction, we use Eq 4 to output a prediction vector $f(D_i)$ for the $i-th$ data point. We take a continuous poll of the 6 latest votes cast and output the highest polled gesture as the latest gesture performed. This voting mechanism reduces false positives. That is, we compute $\hat{y_l} = f(D_i) + f(D_{i+1}) + \ldots + f(D_{i+5})$ and output the gesture with the highest value; if none of the gesture has enough mass then we declare no-gesture.

Based on the detected gesture, our Android app initiates certain tasks on the phone. Our app programs a state-machine that is illustrated in Figure 10. A gesture performed on the cane results in corresponding transitions in the app's state machine. For example, our app has a "home" state. In "home" state, a double-tap leads to phone reading out the current time and returning to the "home" state. A twirl gives you the current

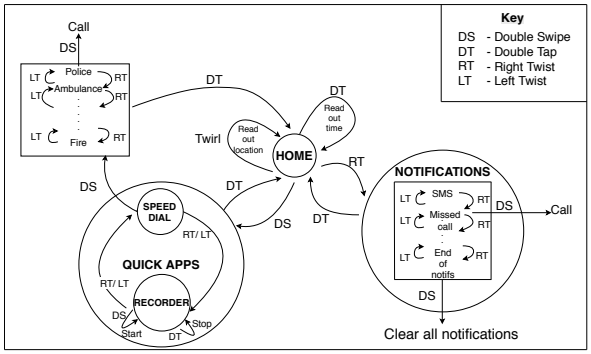

Figure 10. State machine for the App on smartphone. See Figure 4 for illustration of our DT, DS, RT, LT, Twirl gestures.

	2X Tap	Right Twist	Left Twist	Twirl	2X Swipe	No Gesture	*Recall*
2X Tap	95	0	0	0	0	0	1
R. Twist	0	86	8	0	0	1	0.91
L. Twist	0	9	81	0	0	5	0.85
Twirl	0	0	0	82	11	2	0.86
2X Swipe	0	0	0	1	93	1	0.98
Precision	1	0.91	0.91	0.99	0.89	-	

Table 1. Confusion matrix for detection gestures performed across all visually impaired users. Easy gestures like double tap and double swipe are detected with nearly 100% accuracy. Naturally, detection of right twist and left twist is relatively inaccurate as the users sometimes perform both the twists in the same action.

location and twists read out latest notifications. A double-swipe enters the app into our "quick apps" menu that can be further navigated using twists and double-swipe. A detailed description of the state machine can be found in Appendix II.

We designed our app state-machine so that certain key tasks can be performed quickly. For example, we enabled a quick call-back for a missed call or quickly reading out the current time. We designed the app state-machine based on feedback from informal interviews with 15 VI volunteers.

EXPERIMENTAL STUDY

We conducted user studies to evaluate the in-situ accuracy of the GesturePod and the utility of the interactive cane for visually impaired users. We investigated whether the gestures on the cane were natural and accurate enough to improve smartphone user experience. Specifically, we performed three rounds of experiments to test the following hypotheses:

Hypothesis 1: The ML model can accurately recognize the set of 5 gestures across participants. Further, the gestures themselves are simple natural and are easy to learn.

Hypothesis 2: The interactive cane enables faster task completion of common smartphone-related tasks.

Hypothesis 3: Interactive cane enables VIPs who use feature phone to easily adapt basic smartphone functionality and benefit from additional functions such as location.

Subjects

The user study spanned 12 visually impaired (VI) and 6 sighted users. 12 voluntary VI persons were recruited from 3 NGOs. Diversity of users was encouraged in terms of educational qualification, gender, age and handedness.

We conducted three rounds of experiments. VI participants with smartphones participated in Round 1 and Round 2 while VI participants with feature phones participated in Round 1 and Round 3. Sighted users also participated in Round 1.

EXPERIMENTAL DESIGN

Round 1

This round tests prediction accuracy and robustness of the machine learning model in the GesturePod. In addition to measuring the accuracy of our gesture recognition system, we also want to test whether the gestures are natural enough to be learned in a quick training session.

Participant Requirements: White-cane trained VI people as well as sighted volunteers participated in this round of study.

Methodology and experiment design: The experiment began with a description of the project. Users were then briefed about the current set of gestures and how to perform them. Users were then given a training smartphone app to help learn the gestures. The app requests the user to perform a randomly chosen gesture and notifies the user whether the gesture was recognized by the ML model. We noticed most users became familiar with the gestures after 10 minutes of training with the app.

After this brief training session, each VI person as well as sighted user performed 5 sets of 5 gestures in a randomized order: Double Tap, Right Twist, Left Twist, Twirl, Double Swipe. To ensure objectivity and remove any bias, we built an Android app that speaks out the gestures in randomized order and awaits a gesture in the next 10 seconds.

Evaluation Metrics: We analyze the accuracy of the gesture recognition model qualitatively using standard accuracy metrics. Our Android app records the number of times the intended gesture is successfully performed. The number of times the intended gesture wasn't fired, as well as the number of times a false gesture was detected, is also recorded. We report accuracy numbers using standard confusion matrix.$C(i,j) : (i,j) - th$ entry of confusion matrix C denotes he number of instances where the i-th gesture was recognized as j-th gesture. Recall and precision of the i-th gesture are given by:

$$\text{Recall(i)} = \frac{C(i,j)}{\sum_j C(i,j)} \quad \text{Precision(i)} = \frac{C(i,j)}{\sum_j C(j,i)}$$

We also collected feedback from questionnaire to understand how users perceive various gestures and the overall solution.

Results: Table 1 presents the confusion matrix for recognition of gestures performed by VI participants. Evidently, double-tap and double swipe gestures are recognized accurately with

	2X Tap	Right Twist	Left Twist	Twirl	2X Swipe	Recall
2X Tap	30	0	0	0	0	1
R. Twist	0	27	3	0	0	0.9
L. Twist	0	3	27	0	0	0.9
Twirl	0	0	0	2	28	0.9667
2X Swipe	0	0	0	2	28	0.9334
Precision	1	0.9	0.9	0.9355	0.9655	

Table 2. Confusion matrix for detecting gestures performed across sighted users. Naturally, the accuracy of detection is better for sighted users as they can better understand how to perform the twists but we observed that with more training VI users also started to improve on that front.

a small error in prediction. The model tends to confuse right and left twist gestures with each other on occasions, but still achieves more than 85% accuracy. That is, with just 10 minutes of user training, our method is able to achieve overall accuracy of 92%±3% (with 95% confidence). Moreover, this also indicates good generalization ability of the model as it is able to accurately detect gestures for participants whose data was not used for training the ML model. Naturally, performance of ML model on sighted volunteers' gestures was more accurate as they could see exactly how the gesture should be performed; this also indicates that with more training, our performance for VI people might also improve. See Table 5, APPENDIX I for the confusion matrix of the ML model on validation data sampled from the same distribution as the training data.

Round 2

The second experiment tests whether the interactive cane results in faster completion of frequent smartphone-related tasks as compared to prior usage patterns of the participants. We selected the tasks based on input from prior interviews with visually impaired smartphone regarding tasks that are critical to them. To facilitate access to certain smartphone tasks, we built a custom Android app that has an internal state machine. See the Prediction Pipeline section for an overview of our app state-machine.

We compare time for task completion in both constrained and unconstrained environments. Constrained environments are those in which the participant has one or both their hands occupied, e.g., walking with a bag, having a cup of coffee in one hand. In unconstrained environments the participants have at least one hand free to use their phone, e.g., standing and conversing with friends.

Participant Requirements: This round involves participants that: a) are visually impaired, b) have received mobility training with cane, and (3) have used an Android smartphone for at least 3 months.

Methodology and experiment design: Participants are trained for a period of 15 minutes. The state machine that maps gestures to tasks on the smartphone as described in Appendix II is explained to the user.

Question	Responses averaged across 12 participants
How would you rate the ease to perform the gestures?	4.25 ± 0.43
Do the gestures seem natural and intuitive?	4.17 ± 0.48
Was the GesturePod able to detect your gestures?	4.29 ± 0.40
How would you rate the ease of remembering the gesture?	4.58 ± 0.38
How would you rate the amount of effort required to get used to the way gestures must be performed?	3.87 ± 0.65
How would you rate the design of the state-machine on the app?	4.04 ± 0.50
How would you rate the design of the GesturePod?	4.63 ± 0.37
How would you rate your overall product experience?	4.41 ± 0.45

Table 3. Average participant ratings with 95% confidence interval on different aspects of GesturePod. 1 represents the lowest rating and 5 is the highest rating.

The participant is then asked to perform 4 repetitions of 5 activities in a randomized order:

1. Receive a phone call - answer a phone call from a test phone.
2. Call back the last caller from a missed call notification.
3. Start and stop audio recordings on the phone.
4. Read out the current geographic location.
5. Check for notifications and read out the time.

Each repetition is performed in a different setting:

1. Using their Android Phone (with no bindings to the cane) in unconstrained environments.
2. Using their Android Phone (with no bindings to the cane) in constrained environments.
3. Using the interactive cane to interact with their phone in unconstrained environments.
4. Using the interactive cane to interact with their phone in constrained environments.

Evaluation Metrics: For each user, we video-record the study; see supplementary material for a partial video of a recorded session. Later, we replay the video and note the following after the user study:

1. If each of the intended activities is successfully completed
2. The time taken to perform each activity.

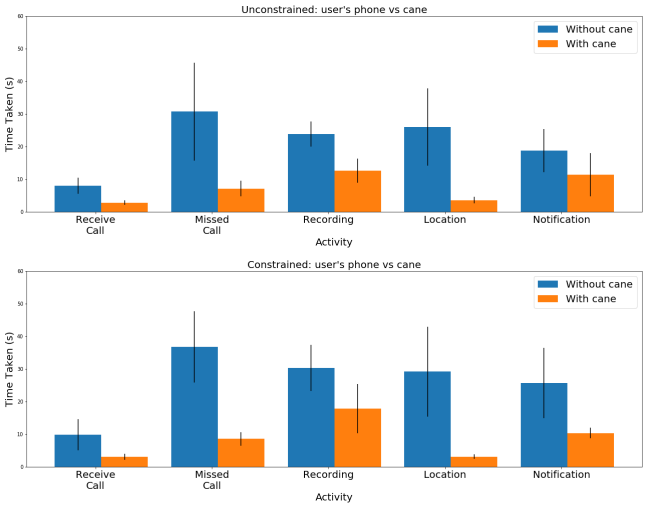

Figure 11. Comparison of task completion times between the interactive cane usage and direct smartphone usage in unconstrained (top) and constrained (bottom) settings.

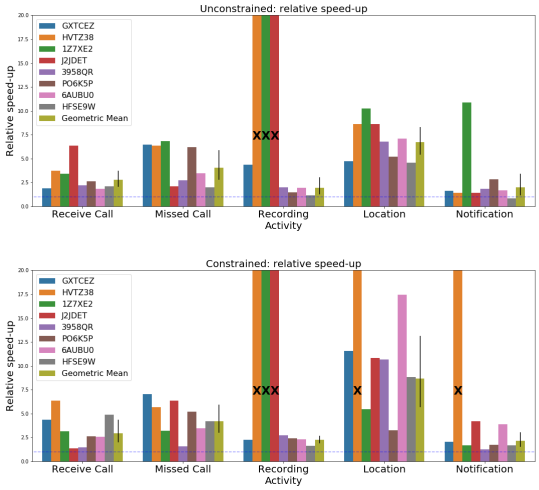

Figure 12. Relative speedup of task completion time with interactive cane usage over direct smartphone usage in unconstrained (top) and constrained (bottom) settings. Figure also shows average speedup across all users. x indicates the user could not perform the activity using *only* smartphone.

Results: Figure 11 indicates that using an interactive cane to access smartphone results in a significant improvement to all five task completion times in both constrained and unconstrained settings. Figure 12 illustrates the relative speedup in performing various activities using the interactive cane. Figure 12 presents speed-up for each of our 8 participants, as well as the average speedups and the corresponding 95% confidence intervals. The improvement in task completion times ranges between 1.75x and 9x depending on the task and setting, and is slightly higher in the constrained setting. Interestingly, some of the users could not complete some of the activities using their smartphone but were able to finish the activities quickly using GesturePod.

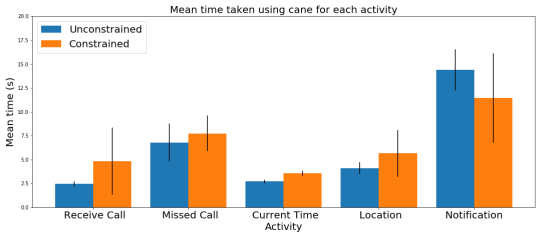

Figure 13. Time to taken to complete each activity using GesturePod paired with smartphone in an unconstrained and constrained environment by users with featurephone as their primary phone.

Round 3

This round of experiments studies the third hypothesis - that the interactive cane makes it easier for VI feature phone users to adopt basic smartphone functionality like reading out location.

Participant Requirements: This round involves participants who: (a) are visually impaired, (b) have received mobility training with cane, (c) use a feature phone as their primary phone, and (d) are not used to smartphones. Three persons participated in this round.

Methodology: Participants are briefed about the utility of a smartphone and then told about the state machine for about 15 minutes; recall that our state machine maps gestures to certain activities on the smartphone (Figure 10). We explain the concept of notifications to participants who were not aware and provide the participant with the interactive cane with a GesturePod attached to it.

Just as in the previous round, we then asked each participant to perform 4 repetitions of 5 tasks (same as mentioned in the previous section, except task number 3 - “start and stop audio recordings on the phone" is replaced with “Note the current time") in a randomized order. Similarly, we measure the time required to perform the task using feature phone as well as using our interactive cane in combination with the Android app, in both constrained and unconstrained setting.

Evaluation Metrics: We measure all the three metrics mentioned in the previous section. At the end of the user studies, the participant is provided with a questionnaire to get a qualitative response of the interactive cane.

Results: We found that within the 15-minute training period, participants were able to use the smartphone for basic functionality like accepting or rejecting calls through their smartphones. Moreover, they were also able to use functionalities such as knowing their geographic location and reading out notifications in a reasonable amount of time. Figure 13 presents the time take for each task in constrained and unconstrained settings.

All three participants could read the current time in under 3 seconds in an unconstrained environment and in under 4 seconds in a constrained environment. Further, we observed that participants could receive calls on an average (GM) 2.37

times faster in unconstrained environments. Similarly, the participants could dial back from a missed call notification on an average (GM) 1.86 times faster in constrained environments. Naturally, verifying statistical significance of these studies is difficult due to only three participants.

Other Observations: None of the participants had location feature on their feature phone. They mentioned that they relied on passers-by for getting to know their current location. Somewhat surprisingly, two of the three participants did not know how to read time from their phone and had to rely on others. They felt that interactive cane helped them easily access the time and their location.

One of the three participants knew how to read notifications on their phone. Amongst the remaining two, one participant had not heard of notifications before (we explained to them) and the other knew what notifications were but did not know how to access it on their feature phone. For the first person who knew how to read notifications, our GesturePod made it faster to read and dismiss notifications.

Overall Subjective Feedback

We also collected subjective feedback about ease of using GesturePod and performing various gestures. The feedback was collected by informing the participants that we are not the owners of the prototype. Table 3 presents average of the ratings provided by the participants for various questions; we also provide the ratings' 95% confidence intervals so as to ensure that the conclusions are statistical significant. Participants' response to first five questions in Table 3 suggests that participants were quickly able to learn various gestures and felt that they were natural and intuitive. Similarly, responses to last three questions of Table 3 indicate that the participants found GesturePod as well as connected apps to be helpful.

CONCLUSION

In summary, we developed an **interactive cane** based on our GesturePod which is a plug-play device that can be attached to any cane. GesturePod deploys real-time ML algorithm to detect gestures robustly and effectively despite variations in users, participants etc. Our user studies indicate that with simple and natural gestures, we can design a novel User Interface that can aid the VI. Going forward, we are working on detecting more nuanced gestures that can enable access to many more phone apps.

REFERENCES

1. Daniel Ashbrook and Thad Starner. 2010. MAGIC: A Motion Gesture Design Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2159–2168. DOI: http://dx.doi.org/10.1145/1753326.1753653
2. Jared M. Batterman, Vincent F. Martin, Derek Yeung, and Bruce N. Walker. 2018. Connected cane: Tactile button input for controlling gestures of iOS voiceover embedded in a white cane. *Assistive Technology* 30, 2 (2018), 91–99. DOI:http://dx.doi.org/10.1080/10400435.2016.1265024 PMID: 28140766.
3. J. Faria, S. Lopes, H. Fernandes, P. Martins, and J. Barroso. 2010. Electronic white cane for blind people navigation assistance. In *2010 World Automation Congress*. 1–7.
4. Davide Figo, Pedro C. Diniz, Diogo R. Ferreira, and João M. Cardoso. 2010. Preprocessing Techniques for Context Recognition from Accelerometer Data. *Personal Ubiquitous Comput.* 14, 7 (Oct. 2010), 645–662. DOI: http://dx.doi.org/10.1007/s00779-010-0293-9
5. A. J. Fukasawa and K. Magatani. 2012. A navigation system for the visually impaired an intelligent white cane. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 4760–4763. DOI:http://dx.doi.org/10.1109/EMBC.2012.6347031
6. Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. 2017. ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1331–1340. http://proceedings.mlr.press/v70/gupta17a.html
7. Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 145–154. DOI: http://dx.doi.org/10.1145/1240624.1240646
8. Lei Jing, Yinghui Zhou, Zixue Cheng, and Tongjun Huang. 2012. Magic Ring: A Finger-Worn Device for Multiple Appliances Control Using Static Finger Gestures. *Sensors* 12, 5 (2012), 5775–5790. DOI: http://dx.doi.org/10.3390/s120505775
9. Jin Sun Ju, Eunjeong Ko, and Eun Yi Kim. 2009. EYECane: Navigating with Camera Embedded White Cane for Visually Impaired Person. In *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility (Assets '09)*. ACM, New York, NY, USA, 237–238. DOI: http://dx.doi.org/10.1145/1639642.1639693
10. Sung Yeon Kim and Kwangsu Cho. 2007. Usability and design guidelines of smart canes for users with visual impairments. *International Journal of Design* 7, 1 (2007), 99–110. DOI: http://dx.doi.org/10.1111/j.1444-0938.2007.00199.x
11. Sven Kratz, Michael Rohs, and Georg Essl. 2013. Combining Acceleration and Gyroscope Data for Motion Gesture Recognition Using Classifiers with Dimensionality Constraints. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces (IUI '13)*. ACM, New York, NY, USA, 173–178. DOI: http://dx.doi.org/10.1145/2449396.2449419

12. Ravi Kuber, Amanda Hastings, Matthew Tretter, and Dónal Fitzpatrick. 2012. Determining the accessibility of mobile screen readers for blind users. (2012).

13. Ashish Kumar, Saurabh Goyal, and Manik Varma. 2017. Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1935–1944. http://proceedings.mlr.press/v70/kumar17a.html

14. Matt Kusner, Stephen Tyree, Kilian Weinberger, and Kunal Agrawal. 2014. Stochastic Neighbor Compression. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Eric P. Xing and Tony Jebara (Eds.), Vol. 32. PMLR, Bejing, China, 622–630. http://proceedings.mlr.press/v32/kusner14.html

15. Je Seok Lee, Heeryung Choi, and Joonhwan Lee. 2015. TalkingCane: Designing Interactive White Cane for Visually Impaired People's Bus Usage. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct (MobileHCI '15)*. ACM, New York, NY, USA, 668–673. DOI:http://dx.doi.org/10.1145/2786567.2793686

16. J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya, and V. Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. In *2009 IEEE International Conference on Pervasive Computing and Communications*. 1–9. DOI: http://dx.doi.org/10.1109/PERCOM.2009.4912759

17. Zhiyuan Lu, Xiang Chen, Zhangyan Zhao, and Kongqiao Wang. 2011. A Prototype of Gesture-based Interface. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*. ACM, New York, NY, USA, 33–36. DOI:http://dx.doi.org/10.1145/2037373.2037380

18. David Mace, Wei Gao, and Ayse Coskun. 2013. Accelerometer-based Hand Gesture Recognition Using Feature Weighted Naive Bayesian Classifiers and Dynamic Time Warping. In *Proceedings of the Companion Publication of the 2013 International Conference on Intelligent User Interfaces Companion (IUI '13 Companion)*. ACM, New York, NY, USA, 83–84. DOI:http://dx.doi.org/10.1145/2451176.2451211

19. David A. Mellis, Ben Zhang, Audrey Leung, and Björn Hartmann. 2017. Machine Learning for Makers: Interactive Sensor Data Classification Based on Augmented Code Examples. In *Proceedings of the 2017 Conference on Designing Interactive Systems (DIS '17)*. ACM, New York, NY, USA, 1213–1225. DOI: http://dx.doi.org/10.1145/3064663.3064735

20. Microsoft. 2017. Seeing AI. (2017). https://www.microsoft.com/en-us/seeing-ai/.

21. Tomàs Pallejà, Marcel Tresanchez, Mercè Teixidá, and Jordi Palacin. 2010. Bioinspired Electronic White Cane Implementation Based on a LIDAR, a Tri-Axial Accelerometer and a Tactile Belt. *Sensors* 10, 12 (2010), 11322–11339. DOI: http://dx.doi.org/10.3390/s101211322

22. Khanna Rohit, Raman Usha, and Rao Gullapalli N. 2007. Blindness and poverty in India: the way forward. *Clinical and Experimental Optometry* 90, 6 (2007), 406–414. DOI: http://dx.doi.org/10.1111/j.1444-0938.2007.00199.x

23. Huizi Mao Song Han and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR' 16)*.

24. GingerMind Technologies. 2017. Eye-d. (2017). https://www.eye-d.in/.

25. Dheeraj Mehra Anurag Gupta Vasu Dev Sharma Saumya Jain Chinmay Agarwal Ankush Garg Sandeep Singh Gujral M. Balakrishnan Kolin Paul P.V.M. Rao Vaibhav Singh, Rohan Paul and Dipendra Manocha. 2010. Smart cane for the visually impaired: Design and controlled field testing of an affordable obstacle detection system. In *12th International Conference on Mobility and Transport for Elderly and Disabled Persons (TRANSED '10)*.

26. Michele A. Williams, Caroline Galbraith, Shaun K. Kane, and Amy Hurst. 2014. "Just Let the Cane Hit It": How the Blind and Sighted See Navigation Differently. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility (ASSETS '14)*. ACM, New York, NY, USA, 217–224. DOI: http://dx.doi.org/10.1145/2661334.2661380

27. Kai Zhong, Ruiqi Guo, Sanjiv Kumar, Bowei Yan, David Simcha, and Inderjit Dhillon. 2017. Fast Classification with Binary Prototypes. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Aarti Singh and Jerry Zhu (Eds.), Vol. 54. PMLR, Fort Lauderdale, FL, USA, 1255–1263. http://proceedings.mlr.press/v54/zhong17a.html

APPENDIX I

Participants from user study were asked about their preference among the 3 choices:
Would you prefer to use the cane:

1. Using only buttons on the cane
2. Through gestures on the cane
3. Using both buttons and gestures on the cane.

Their responses were tabulated in Table 4

Options	Number of Participants who chose a particular option
using only Buttons on the cane	1
through gestures on the cane	5
both	6

Table 4. User study participant preference towards GesturePod.

Table 5 presents confusion matrix of the learned model, on 20% data sampled as validation set.

	No Gesture	2X Tap	Right Twist	Left Twist	Twirl	2X Swipe	*Recall*
No Gesture	30007	1	4	0	1	0	1.000
2X Twist	6	209	0	0	0	0	0.972
R. Twist	2	0	107	1	1	0	0.964
L. Twist	3	0	0	127	0	0	0.977
Twirl	0	0	0	0	746	0	1
2X Swipe	14	0	0	0	3	146	0.896
Precision	0.999	0.995	0.964	0.992	0.993	1.000	

Table 5. Confusion matrix for test data.

APPENDIX II

The interactive cane app works as a state machine, which is basically a system of gesture-action mappings. The app maintains its current state of execution at every point of time, and thereafter performs the corresponding action with respect to that state. The app launches in the home state, which is analogous to the standard home screen of a smartphone. Performing a double tap in the home state makes the app read out the current time, whilst remaining in the home state. A double tap in any other state reverts the state of the app back to the home state. Basically, a double tap is equivalent to the home button which any smartphone has.

The app speaks out the current location by getting the GPS coordinates when a twirl is performed in the home state. Thereafter, it maintains its state to the home state.

Performing a right twist in the home state makes the app recite the existing notifications in the smartphone. The app speaks out “You have x notifications", and doing a left twist at this stage is redundant and will simply repeat this line. Thereafter, scrolling through the notifications is achieved by the left twist - right twist pair - right twist to go one notification ahead, and left twist to scroll one notification back. At any point in time while scrolling through the notifications, performing a double tap resets the state of the app to the home state. While scrolling if the current notification is that of a missed call, performing a double swipe results in dialing the number to call the person. After having scrolled through all the notifications, there is an option available to flush all the notifications. This is achieved by performing a double swipe at the end of all the notifications.

Provided that the app has been launched, and either is alive or is running in the background, an incoming call on the phone can be accepted by performing a double swipe, whereas performing a double tap results in rejecting the call.

There is a “Quick Apps" menu as part of the app. This menu can be accessed by performing a double swipe in the home state. Two useful functionalities have been incorporated as part of this Quick Apps menu, namely, Speed Dial and Audio Recorder. Scrolling in the Quick Apps menu happens by means of left and right twists. Performing a double swipe at either of Speed Dial or Audio Recorder results in selecting it. Double swiping at Speed Dial presents a list of contacts stored beforehand as part of speed dial. Scrolling through these contacts is again possible with the help of the left twist-right twist pair. Double swiping now at any contact results in calling that number, and thereafter the call can be ended by performing a double tap.

Performing a double swipe at the audio recorder makes it start recording. Thereafter, a double tap stops the recording, and the recorded audio file gets stored on phone.

At every point in time, there will be a voice-over with respect to the option being selected. Also, a lengthy voice-over (eg. a lengthy location with unnecessary details like pin code, state, county etc.) can be interrupted and stopped by performing a double tap, and the app will cut the voice over and return to the home state.

Performing any gesture at a state other than the state-gesture pairings described above results in the app remaining in the same state. Also, not performing any gesture at a state other than the home state for a period of more than 7 seconds results in the app reverting to its home state.

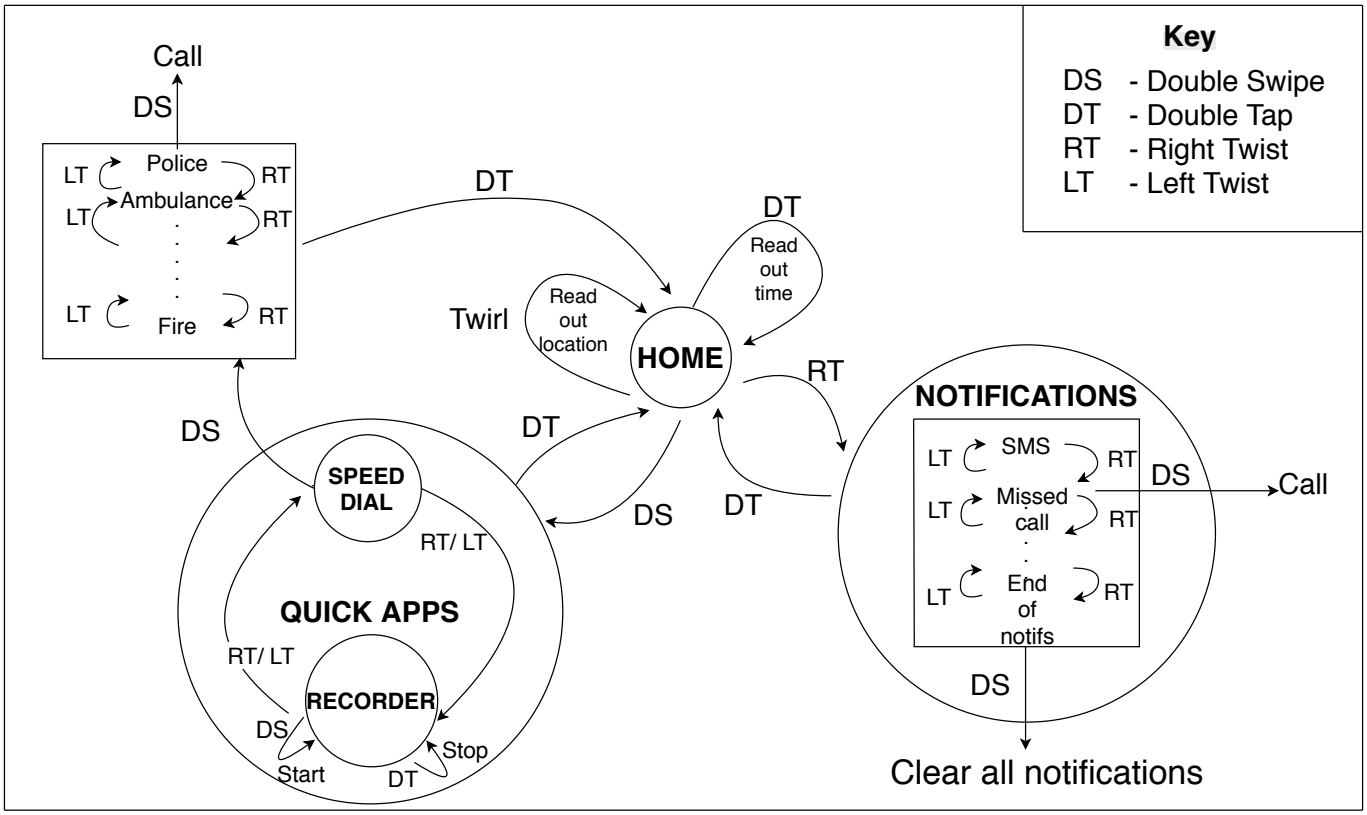

Figure 14. State machine for the Android App.