Real-time Arm Skeleton Tracking and Gesture Inference Tolerant to Missing Wearable Sensors

Yang Liu, Zhenjiang Li
City University of Hong Kong, Hong Kong, China

Zhidan Liu, Kaishun Wu
Shenzhen University, Shenzhen, China

ABSTRACT

This paper presents ArmTroi, a wearable system for understanding and analyzing the detailed arm motions of people primarily by using the motion sensors from wrist-worn wearable devices. ArmTroi can achieve real-time 3D arm skeleton tracking and reliable gesture inference tolerant to missing wearable sensors for enabling numerous useful application designs. We have coped with two major challenges through ArmTroi. First, the skeleton of each arm is determined from the locations of the elbow and wrist, whereas a wearable device only senses a single point from the wrist. We find that the potential solution space is huge. This underconstrained nature fundamentally challenges the achievement of accurate and real-time arm skeleton tracking. Second, wearable sensors may not reliably provide sensory data. For example, devices are not worn by the user, yet the learning tools for gesture inference, such as deep learning, typically have static network structures, which require nontrivial network adaptation to match the input's varying availability and ensure reliable gesture inference. We propose effective techniques to address above challenges, and all computations can be conducted on the user's smartphone. ArmTroi is thus a fully lightweight and portable system. We develop a prototype and extensive evaluation shows the efficacy of the ArmTroi design.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing**.

KEYWORDS

arm tracking, gesture inference, mobile sensing, deep learning

ACM Reference Format:
Yang Liu, Zhenjiang Li and Zhidan Liu, Kaishun Wu. 2019. Real-time Arm Skeleton Tracking and Gesture Inference Tolerant to Missing Wearable Sensors. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307334.3326109

1 INTRODUCTION

This paper presents a wearable system, ArmTroi. It can track the 3D skeleton of the entire arm of a user in **real time**, *e.g.,* the locations of the *elbow* and *wrist* with respect to (*w.r.t.*) the body [41]. ArmTroi uses motion sensors (*i.e.,* accelerometers and gyroscopes) from a wearable device only on the user's wrist (*e.g.,* smart watch), instead of attaching multiple sensors on the user's entire arm. Both arms can be tracked when a wearable device is worn on the wrist of each arm. The tracked skeletons can directly enable numerous useful application designs (§2.1). Moreover, the reconstructed skeletons can be further processed using a carefully designed deep learning network to reliably recognize user's gestures after resolving a unique missing wearable sensor issue. Thus, ArmTroi can be used as a generic platform to comprehensively understand and analyze people's detailed arm motions for enabling useful applications.

The ArmTroi design does not depend on the ambient infrastructure support, such as off-loading expensive computations to clouds or edges [13, 57], due to limitations arising from restricted service coverage, system costs, and privacy concerns. This paper aims to achieve such tracking and inference ability in a fully *portable* and *lightweight* system. In particular, wearable motion data are delivered to the user's smartphone in ArmTroi, and the phone promptly produces the tracking result for the use of applications, which can be processed further by the deep learning network on the phone for gesture inference. In this manner, the skeleton tracking and motion inference abilities can be always available with the user. Many useful applications can also obtain remarkable benefits, *e.g.,* elderly care [50], e-health [14], and human-computer interaction (HCI) [33], in the era of smart cities and Internet of things (IoT).

However, the following technical challenges should be carefully addressed to obtain the aforementioned benefits.

1) We take advantage of kinematic studies [35], wherein ArmTrak [41] recently makes a remarkable contribution to recover user's arm motions from a single watch, to achieve the aforementioned skeleton tracking design; however, the shortcoming is long recovery latency, *e.g.,* a t-time activity requires around $10 \times t$ times to recover even on a desktop PC [41], which is due to the inherent hardness problem, and limits the solution to an off-line analysis.

Skeleton recovery is essentially a search problem (for unknown elbow and wrist locations within a huge space). We observe that the search space can be carefully diminished (without impairing tracking accuracy), and "unlikely" candidates can be intelligently excluded as early as possible to considerably accelerate the search. By this design, skeleton tracking in ArmTroi occurs in real time even on a mobile phone, and can promisingly achieve a higher accuracy than ArmTrak [41]. In addition, recovery errors are not accumulated over time to support continuous tracking.

2) The fast skeleton tracking can be directly used in many useful applications (§2.1). In many cases, we should further understand user's gestures through learning techniques (Figure 1), *e.g.,* deep learning (which has recently exhibited great success), and the tracked skeleton is also used for the gesture inference in ArmTroi.

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06…$15.00
https://doi.org/10.1145/3307334.3326109

But compared with traditional deep learning's inputs for gesture inference (*e.g.,* video [7]), wearable sensory data are not reliably available (*e.g.,* a wristband is removed by the user, thereby resulting in the disappearance of this part of the input). Thus, an adapted network structure is expected to effectively match currently available inputs to prevent training multiple networks for handling all possible missing input combinations, which is non-scalable and storage cost inefficient for the execution platform (*i.e.,* smartphone) [49].

However, once a deep learning network is deployed after training, the network structure is hardly changed. We introduce an attention-based design to achieve a *virtual* network adaptation to address this possible missing input data issue, where attention [27] is a novel technique for adjusting certain parameters of a network (by multiplying a weighting vector where each weight can vary) to enhance the output on the basis of a subset of the selected input data [27]. Motivated by such an opportunity, we introduce and design an attention component and seamlessly integrate it into our system. In case some inputs are missing, the network can automatically increase the weights for valid inputs and substantially decrease the weights for missing parts, *i.e.,* the missing inputs are marginally considered by the network. One network structure can thus handle all types of missing inputs.

Contribution. In summary, this paper makes following contributions. First, we propose novel techniques to improve the heavyweight computation of the state-of-the-art arm tracking model and achieve real-time tracking. Second, we propose an attention-based adaptation mechanism to achieve the dynamic deep learning network adjustment to be tolerant to the input sensor data missing. Third, we develop an ArmTroi prototype and present a comprehensive evaluation with two case studies to show the design's efficacy.

Implementation. In the current ArmTroi, the *full set* of sensors is attached on three body parts: 1) a smart watch or wristband that senses each wrist to recover arm skeletons *w.r.t.* the body, and 2) a Google Glass or belt sensor that estimates the rotation angle (θ in Figure 2) of an arm skeleton due to body's inclination, *e.g.,* forward leaning or reclining. Notably, ArmTroi is positioned to exert the best effort in tracking and analyzing arm skeletons using only available sensors instead of forcing users to wear all the sensors. Because this design position, some sensor data could be missing, and we hence further propose an attention-based method to achieve a robust gesture recognition in this paper.

We develop a prototype system using LG smart watches, Google Glass, SAMSUNG Galaxy S7, and a desktop with Intel i7-6700 CPU and Nvidia GTX 1080Ti GPU (for neural network training) for evaluation. We implement the ArmTroi arm tracking design on a smartphone, which can achieve real-time tracking and outperform ArmTrak [41] (on a PC with 10x recovery latency) by 15.92%. For gesture inference, ArmTroi is also executed on the smartphone, which achieves accurate activity recognition with 92.7% precision and outperforms a set of recent designs. Finally, we examine ArmTroi for fitness and gesture-based control two case studies.

2 OVERVIEW AND PRELIMINARY

A **skeleton** refers to a body's 3D geometric relation, which is uniquely determined by the location and orientation of each joint [41], *e.g.,* a user's arm skeleton can be described by the eight major joints

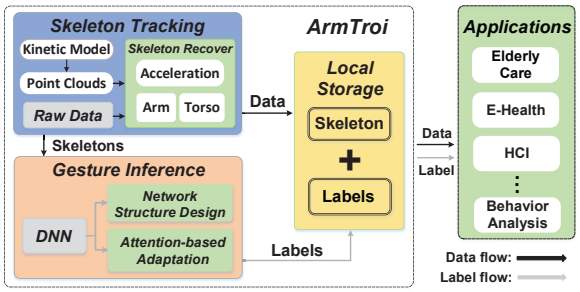

Figure 1: Illustration of the ArmTroi architecture.

(the skeleton model is detailed in §2.2). Thus, a **gesture** is a meaningful skeleton sequence over time.

2.1 Application Scenarios of ArmTroi

ArmTroi can benefit many useful applications atop reconstructed arm skeletons and recognized arm gestures as shown in Figure 1.

1) *Elderly care.* We can observe valuable signs of many diseases, *e.g.,* Parkinson and Alzheimer, by tracking the arm motions of elderly adults. Typical symptoms, such as limb stiffness, slow motion, postural instability and repeated activities, are reflected from the detailed 3D arm skeletons. Doctor can also quantitatively monitor the status of patients after each treatment and obtain a detailed clinic record with a further classification of the observed symptoms. ArmTroi can advance elderly care techniques and industries.

2) *E-health ecosphere.* Detecting meaningful hand gestures can prompt a healthier lifestyle [42, 47], *e.g.,* alert for smoking and reminders to drink sufficient water or eat meals on time. In certain more sophisticated scenarios, such as sport training or fitness [39], full arm skeleton tracking is also required, such as in basketball shooting, weight training, and golf swing analysis. Hence, ArmTroi can considerably expand the sensing ability of existing wearable devices (with simple sensing abilities, such as step counting), which can deeply stimulate the e-health device market.

3) *HCI.* In the wearable computing market, many novel HCI designs have emerged recently (*e.g.,* smart home devices and motion games [51]), and user arms' gestures are the key inputs of these systems. The ArmTroi design can provide a lightweight way to obtain such desired inputs with a substantially extended service coverage and decreased system cost, compared with cameras or depth sensors adopted in existing solutions [26].

2.2 Skeleton Model in ArmTroi

According to [35], the skeleton of people's arms can be described by eight major joints, and these joints can be grouped into three parts, wherein each part $k \in \{\underline{torso}, \underline{left\ arm}, \underline{right\ arm}\}$ is shown in Figure 2. Torso (T) interconnects the two arms, and each arm has three joints. Torso's rotation angle θ determines the overall inclination of the two arms' skeletons.

From Figure 2, we can define each joint's coordinates in the torso coordinate system, *e.g.,* the 3D skeleton obtained from a Kinect device. We select the joint swivel as the origin of the coordinate system and the direction from lumbar to swivel as the z-axis. Then, for any joint j on body part k, we denote the joint's posture p_j^k as the

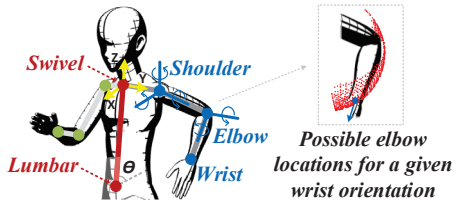

Figure 2: User's arm skeletons can be described by 8 major joints. One point cloud of the elbow is also illustrated.

Categories	Gestures
Daily gestures (4)	*shake hands, make a call, open a door, drink water*
Free-weight (10)	*front raise (a/p), biceps curl (a/p), bent over single arm, chest fly (i/s) bench press (i/s), lateral raise*
Customized (3)	*push, pull, circle*

Table 1: Targeted gestures in ArmTroi. The *a, p, i, s* stand for alternating, in parallel, incline and sitting, respectively.

union of its location loc_j^k and orientation ori_j^k (both are in the torso coordinate system), *e.g.*, $p_j^k = \{loc_j^k, ori_j^k\}$. For example, loc_{wrt}^{ra} and oir_{wrt}^{ra} refer to the location of right wrist and the orientation of right forearm. Thus skeleton is $S = \{p_j^k\}$ for all ks and js.

With these notations, we first show in §3.2 that the skeleton of each arm can be efficiently determined by merely sensing the user's wrist. In addition, the torso's inclination angle θ, *e.g.*, the angle between the horizontal plane and the segment from swivel to lumbar, further introduces an overall rotation (*w.r.t.* the default $\theta = 90°$) for the locations of joints in a global coordinate system. We finally can obtain the user's arm skeletons presented in the global coordinate system[1] by assembling the two aspects.

2.3 Gesture Inference

The gesture set in ArmTroi currently contains three categories of gestures in Table 1 via the following considerations: 1) Many gestures in Table 1 represent widely performed activities in our daily life; 2) The gestures in Table 1 also cover isolated body parts that dominate gestures and gestures that involve multiple body parts. Thus, this selection and organization can allow a full examination of the efficacy of the ArmTroi design; 3) Finally, the scope of the considered gestures includes user's natural and predefined gestures for gesture-based control and interactions in a smart space.

3 SKELETON TRACKING DESIGN

Figure 1 depicts the ArmTroi architecture with the key system modules. We introduce designs of skeleton tracking in this section and gesture inference in the next section (§4).

3.1 Design Principle

A recent study ArmTrak [41] found that if a smart watch is worn on a user's wrist, then the arm's entire skeleton can be recovered from the smart watch's motion data by using kinematic knowledge [35]. Details can be found in [41], but we highlight the recovery principle here given that we also adopt this principle. Moreover, we explain the reason why ArmTrak cannot be used directly in this paper.

Principle. In accordance with the model in §2.2, determining an arm's skeleton essentially confirms two parameters: loc_{elb}: the relative position of the *elbow w.r.t.* the origin of the torso coordinate system *swivel*, and ori_{wrs}: the orientation of the *wrist* in the torso coordinate system, *i.e.,* the manner in which the forearm rotates *w.r.t.* the user's body. Once loc_{elb} and ori_{wrs} are determined, the other parameters associated with all the joints on this arm also become immediately available because the arm is a rigid object [35]. Considering that a smart watch directly senses the user's wrist, its gyroscope can report wrist's orientation in the global coordinate system (denoted as ori_{watch}), which can be further converted to the torso coordinate system [41]. That is, parameter ori_{wrs} can be *indirectly measured* from the watch's gyroscope data ori_{watch}. Thus, the remaining task is to determine loc_{elb} by two phases.

1) *Off-line phase*: From the kinematic model, ArmTrak observes that all possible elbow locations, loc_{elb}, are non-arbitrary given one (indirectly) measured wrist orientation ori_{wrs}. The locations are within a limited range, and different ori_{wrs} values correspond to various ranges (of different sizes). After sampling, each range can be represented by a set of discrete elbow location points and is denoted as a *point cloud* [41] (such as in Figure 2). In the off-line phase, for each wrist orientation ori_{wrs} (of several degree granularity), ArmTrak builds a library to store its corresponding point cloud, which is a one-time effort for the user.

2) *Recovery phase*: When a user's arm is moving, smart watch periodically reports motion data. In addition to the aforementioned gyroscope data $ori_{watch}(t)$ at time t, watch's acceleration $acc_{watch}(t)$ can also be obtained. Considering that the wrist and elbow are connected by the forearm, which is a rigid object, the acceleration of the elbow $acc_{elb}(t)$ can be converted from $acc_{watch}(t)$ in accordance with the skeleton model [41]. That is, $acc_{elb}(t)$ can also be indirectly measured from watch's acceleration data.

Moreover, after T time stamps, we have T point clouds (based on $ori_{wrs}(t)$). We can generate a feasible moving trace of the user's elbow by selecting one location from each point cloud. Given that the sampling interval is established, two consecutive locations lead to elbow's velocity, and two consecutive velocities further derive an acceleration value $\overline{acc}_{elb}$. Thus, we can select the location trace $\{loc_{elb}(t)\}_{t=1}^{T}$ across these T point clouds, whose derived $\{\overline{acc}_{elb}(t)\}_{t=1}^{T}$ best matches the $\{acc_{elb}(t)\}_{t=1}^{T}$ that is indirectly measured from the smart watch's acceleration data (Figure 3).

Complexity. The problem can be formulated using the hidden Markov model (HMM) to search for the most likely result for one unknown across time, *e.g.*, elbow's location $loc_{elb}(t)$ within a large space, *e.g.*, a point cloud, with certain observations related to this unknown, *e.g.*, $acc_{elb}(t)$; then, dynamic programming, *e.g.*, Viterbi, can solve it within $O(S^2T)$ [41], wherein the search space size is S, and the total time step is T. To fulfill the HMM formulation, each circle in Figure 4(a) represents a point cloud at time t of size $O(N)$. As shown in Figure 3, each HMM state of ArmTrak is defined as a pair of elbow locations among two consecutive time stamps. The

[1] It may have a fixed offset with the Earth's north, which it can be compensated for by the user's facing direction estimator [41].

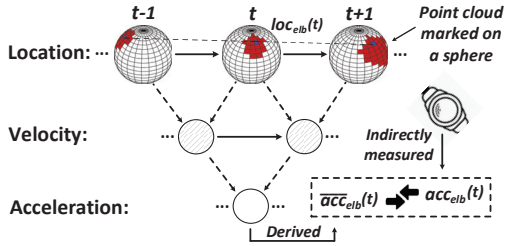

Figure 3: Search elbow locations $\{loc_{elb}(t)\}$, so that derived $\{\overline{acc}_{elb}(t)\}$ best matches $\{acc_{elb}(t)\}$ indirectly from watch.

search space of each HMM state is $O(N^2)$, and the solution complexity is $O(N^4T)$, which can be reduced to $O(N^3T)$ by leveraging the location continuity [41]. To further accelerate the Viterbi search, T can be downsampled to 5 Hz from the default 50 Hz.

To understand the complexity of this solution, the latency on a quad core desktop reported in [41] is that the recovery of 10 *s*, 30 *s*, 1 *min*, and 3 *min* activities takes 98.2 *s*, 289.3 *s*, 9.1 *min*, and 26.9 *min*, respectively. The latency increase is 10x on average. We conduct experiments and obtain consistent results, *e.g.,* 9x to 12x.

3.2 Accelerating Skeleton Recovery

The dominant factor that still slows down recovery is the $O(N^3)$ term as the $O(T)$ term is already downsampled. To this end, one naive solution is to blindly downsample each point cloud. This approach can accelerate recovery while directly sacrificing the resolution of recovered skeletons with deteriorated performance. The following techniques can further diminish the $O(N^3)$ term to prevent this issue:

- HMM state reorganization: reducing $O(N^3)$ to $O(N^2)$.
- Methodology improvement: decreasing N to a considerably smaller n without impairing recovery resolution.

The latency performance can be improved from 10x latency to occur in **real time** even on phones using above two techniques. ArmTroi can achieve an even higher accuracy than ArmTrak. Because with the decreased complexity, ArmTroi can accommodate **denser** point clouds in the search.

Before we present the design details, we note that the speedup for HMM has been studied in various related areas [31, 34], but to our knowledge, the unique challenges and opportunities in ArmTroi have not yet been explored (§7).

3.2.1 HMM State Reorganization. In accordance with the skeleton recovery principle in Figure 3, each HMM state in [41] is

$$s_t \quad = \quad < loc_{elb}(t-1),\ loc_{elb}(t) >,$$

which is depicted in Figure 4(a)[2]. The rationale aims to enumerate the velocities for all location pairs among two adjacent point clouds. Then, two consecutive HMM states can lead to all possible acceleration values $\overline{acc}_{elb}(t)$. Among which, we can select the best matched one *w.r.t.* $acc_{elb}(t)$. However, between two consecutive states, *e.g.,* from s_t to s_{t+1}, only the tuples that share a common location $loc_{elb}(t)$ can be actually transited, thereby indicating that

[2]In Figure 4, t_0 only indicates the start of sensed data. ArmTrak and ArmTroi do not require t_0 to align with an activity's beginning.

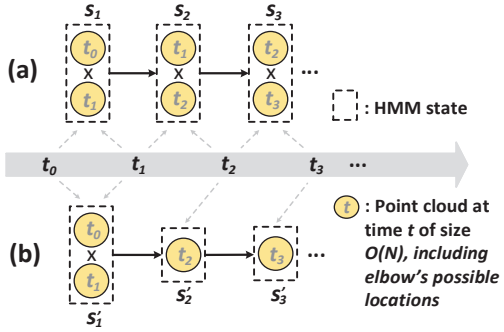

Figure 4: HMM state constructions of size (a) $O(N^2)$, and (b) $O(N)$ for all the states except the first one.

such an HMM state definition involves many infeasible cases. Although ArmTrak [41] improves this condition, *e.g.,* to $O(N^3)$, it can be further diminished.

The transition from s_1 to s_2 in Figure 4(a) is equivalent to that between two new states:

$$s_1' = < loc_{elb}(0),\ loc_{elb}(1) > \quad \rightarrow \quad s_2' = < loc_{elb}(2) >,$$

which also preserves all possible acceleration values cross the first three point clouds (Figure 4(b)). The advantage of this update is that starting from the fourth point cloud (after t_3 in Figure 4(b)), all the remaining states can be defined as $s_t =< loc_{elb}(t) >$, and the HMM state can be reorganized as:

$$s_t = \begin{cases} < loc_{elb}(0),\ loc_{elb}(1) >, & \text{if } t \text{ is 1,} \\ < loc_{elb}(t) >, & \text{otherwise,} \end{cases}$$

which could reduce the possible paths to be considered in the HMM search. This reorganization essentially trades the completeness of the HMM path selection for the search efficiency. Through our experiments, we find that the accuracy loss is only slight (*e.g.,* 0.45 *cm* tracking error increase on average) due to this tradeoff, while this HMM state reorganization design plays an equally important role (as the hierarchical search in §3.2.2) to accelerate the HMM search and ensure the real-time tracking (§5). With the reorganized HMM states, only the first state has size $O(N^2)$, whereas the sizes of all the remaining states are $O(N)$, *e.g.,* the overall search space S is nearly $O(N)$. Hence, search complexity is decreased to $O(N^2T)$.

3.2.2 Hierarchical Search. In addition to the HMM state remodeling, we observe a possibility to further improve the solution complexity without impairing the resolution of the search space, *e.g.,* point clouds. We make the following key observation.

Observation. In the original Viterbi search, we need to explore within a large search space for each time step, but only one location is the correct solution, which implies that most computations are consumed ("wasted") to calculate the likelihoods for all "incorrect" locations so that they can be eventually excluded. Our core idea is thus to exclude incorrect locations as early as possible to minimize computational waste and focus on likely-to-be correct candidates for acceleration.

Proposed solution. To this end, we propose to conduct the search in a hierarchical manner, which is a multi-scale search over the entire search space, starting with a coarse-level estimation of the

point cloud path and then progressively evaluating the path at a finer spatial scale. In particular, we first conduct downsampling for point clouds with a ratio of $\frac{1}{n_1}$, *i.e.,* we group every n_1 nearby locations within all $O(N)$ location points within a cloud into $O(\frac{N}{n_1})$ regions and use the centroid of each region to form a coarse-level search space with a size of $O(\frac{N}{n_1})$. Then, we perform the first round of search over this coarse-level space. In the execution of the first round of search, the most likely region can be determined for each time step, as shown in Figure 5. The complexity for completing this round of search is $O(\frac{N}{n_1})^2T$.

We can immediately launch the next round of search after selecting four regions from the first round (nearly in parallel) while focusing on these selected regions with the output from the first round (*i.e.,* selected regions). That is, the effective point clouds in this new round are merely "shrunk" to these selected regions of size $O(n_1)$, as shown in Figure 5. In principle, we can further divide each selected region into sub-regions by using a downsampling ratio of $\frac{1}{n_2}$ and then search over these sub-regions for the second round. The complexity of the second round search is $O(\frac{n_1}{n_2})^2T$.

This process can be repeated. In the last round, all the original location points in the selected regions from the second last round are used to finalize the elbow's location at each time step. In accordance with the abovementioned design, the hierarchical search excludes the incorrect locations as early as possible (*i.e.,* unselected coarser-level regions) to minimize computational waste, while preserving the original location point resolution in the final result (as the last round search uses the original elbow's location points within the selected region). In summary, if we apply hierarchical search for R rounds, then solution complexity is as follows:

$$O(\sum_{i=1}^{R}(\frac{n_{i-1}}{n_i})^2T), \tag{1}$$

where n_0 is N. To understand the efficacy of our proposed solution to reduce computational complexity, considering that we search two rounds, where $n_1 = 10$ and $n_2 = 1$, *i.e.,* in the second round. Thus, complexity decreases from $O(N^2T)$ to

$$O((\frac{N}{n_1})^2T + (\frac{n_1}{n_2})^2T) = O((\frac{N}{10})^2T + (10)^2T) = O(n^2T),$$

where $n \approx \frac{N}{n_1}$, *e.g.,* $N/10$, and $n_1 \ll N$. This result also indicates that complexity reduction mostly results from the first round of downsampling. We thus adopt a two-layer search ($R = 2$) in the current ArmTroi, and the setting of n_1 is further examined in §5.

Enhancement. We also propose two enhancement designs to achieve further improvements: 1) moderately expanding the selected regions from the first round of search to tolerate certain searching errors and 2) increasing the cloud point density of the selected regions for the last round of search to augment the final resolution. The details are omitted due to page limitation, but we include their performance in §5.

3.2.3 Torso Motion Compensation. For some activities in Table 1, the user's torso inclination angle θ in Figure 2 may not be 90°, *i.e.,* torso's *inclination*. Moreover, some activities may even be performed when the user is moving, *i.e.,* punching while walking for the gesture-based control, which is denoted as torso's *moving*. In the future, if belts widely have motion sensors, then such torso motions can be trivially compensated for. In this study, we examine another complementary opportunity (yet immediately usable) from a wearable that is already available in the market, *e.g.,* smart glasses, to demonstrate the design with the full set of sensors.

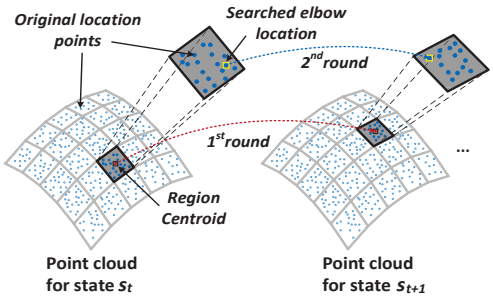

Figure 5: Illustration of the proposed hierarchical search.

Given that the head is directly connected to the torso and we focus on distinguishing several body statuses in similar activities in Table 1, *e.g.,* body is vertical or inclined, smart glass sensors are sufficient for such design.

1) Compensate for torso's θ angle. To estimate the torso's θ angle, the glasses' gyroscope data, after projected onto the x-z plane of the torso in Figure 2, approximately indicate the θ angle, which also excludes user's head horizontal rotation. The user's head may also rotate along the x-z plane, such as when nodding. During activities, however, the θ angle dominates the projected gyroscope data onto the x-z plane, whereas head rotation along this plane only introduces a varying and infrequent component. The average gyroscope data projected onto the x-z plane within the sliding window provide a good approximation because we focus on distinguishing several torso inclinations in Table 1 (§5).

2) Compensate for torso's moving. We can detect a user's walk on the basis of the periodical pattern from the glasses' accelerometer readings. In §5, if the glass gyroscope indicates that a user's head is not rotating, then the acceleration of the smart glasses is close to that of the body's walking. Thus, we can deduct the acceleration data of the glasses from those of the smart watch (in the torso coordinate system) before recovering the skeleton.

Summary. So far, two wrist wearable's data can recover user's arm skeletons[3] and glasses provide torso's inclination (*e.g.,* inclined, vertical, or declined), which can be assembled to form the final skeleton output. The skeleton can be used directly in applications (§5) or to further infer the corresponding gesture labels (§4).

4 GESTURE INFERENCE TOLERANT TO MISSING WEARABLE SENSORS

With complete skeletons as inputs (from two arms and torso), we can train a deep learning network for gesture inference. However, wearable sensors are not reliably available. If a wearable device is removed or out of service [49], then the corresponding body part lacks sensor coverage, and this skeleton portion becomes missing. In such case, we should adapt the network to match the current available inputs to ensure continued performance. To this end, we

[3]When elbow location is known, other joint locations on the same arm can be easily obtained as they are connected by rigid objects.

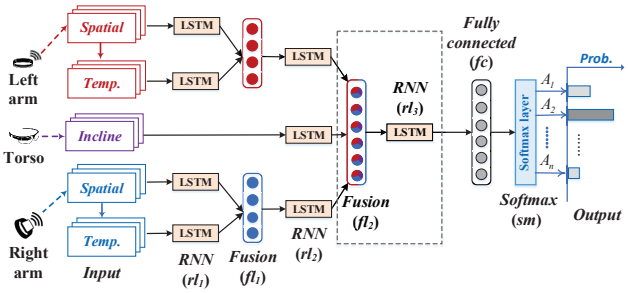

Figure 6: Network structure design to preserve motion features from each body part with all input sensors.

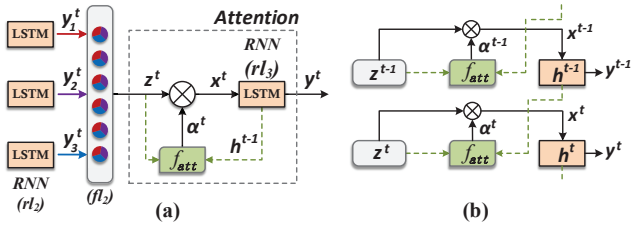

Figure 7: (a) Relation of the attention component with other network modules; (b) Weights update by attention.

first introduce a primary network design with the full set of inputs (§4.1). Moreover, network structure is further augmented by an attention-based design to handle the missing input data issue (§4.2).

4.1 Primary Network Structure

We first propose a primary network structure design, as shown in Figure 6. Because each body part's motion is relatively independent [35], this network can initially extract motion characteristics from individual body parts to preserve features from each part's motion and then gradually fuse them to infer various gestures.

Network inputs. For this primary network, the detailed inputs from the recovered user skeletons include the following:

- skeleton's *spatial* relation: the coordinates of all the eight major joints in the skeleton model (Figure 2) at each time step, which mostly reflect the skeleton's spatial shape.

The following can be further used considering that the user's gestures are essentially a time-domain skeleton sequence:

- skeleton's *temporal* relation: the coordinate difference for each joint in every two consecutive time steps.

Network structure. With the aforementioned inputs, we propose to first apply a recurrent neural network (RNN) to extract the input's individual dynamic motion features [30] and then gradually concatenate them with additional RNNs for further extracting characteristics from an aggregated view, which are finally used for the gesture inference shown in Figure 6 with three-layer RNNs. Given that glasses are only used to infer the torso's inclination, the temporal sequence of torso is omitted from the input.

1) *RNN layer*. The input data and extracted features from the fusion layers fl_1 and fl_2 (which are introduced later) are passed to the RNN layers (rl_1, rl_2, and rl_3) for the motion dynamics analysis, where we adopt the popular long short-term memory (LSTM) [18, 38] for the RNN layers without the vanishing gradient issue [17]. Assuming that the LSTM's input sequence of T time frames is $x_r = (x_r^0, ..., x_r^{T-1})$. Its output is $y_r = (y_r^0, \cdots, y_r^{T-1})$, and each

$$y_r^t = o_r^t \circ f_y(c_r^t),$$

where $c_r^t = f_r^t \circ c_r^{t-1} + i_r^t \circ f_c(W_r^c x_r^t + W_r^y y_r^{t-1} + b_r^c)$ is the cell in an LSTM model; o_r^t, f_r^t and i_r^t are gates in an LSTM model; W_r^c and W_r^y are the weights; b_r^c is a bias vector; f_y and f_c are the activation functions [30]; and r is the index of each LSTM module.

2) *Fusion layer*. Two groups of fusion layers (fl_1 and fl_2) exist. The fl_1 layers integrate the extracted features (by LSTM) from two arms. Then, fl_1's outputs are reprocessed by LSTM, and their consequent outputs with torso's feature are further fused by fl_2.

3) *Fully connected and Softmax layers*. After these two layers, the output is the activity with the highest probability.

4.2 Attention-based Network Adaptation

When some input sensing data are unavailable, instead of training multiple networks such that each one handles a type of missing input (which is nonscalable and also storage cost inefficient, particularly for mobile devices), we introduce an attention-based design to achieve a *virtual* network adaptation, which behaves similar to a series of dedicated networks that handle different types of missing input but without the network structure changes. Only certain parameters are adjusted, and their varying results can automatically match the currently available inputs.

Attention component. The attention model [27] is an emerging technique for dynamically adjusting deep learning network's "focuses" by multiplying a weighting vector that each weight can vary, such that the output is mostly derived on the basis of a subset of the input data (also known as context). The network differentiates its input data and always utilizes the most effective portion (context) to generate outputs. We can thus leverage this ability to automatically increase the weights for valid inputs and substantially decrease the weight for missing parts (the missing inputs are marginally considered in the network). Hence, the attention component is a suitable technique for handling the missing input sensor data issue.

Solution. ArmTroi introduces an attention component in the dashed rectangle in Figure 6 between fusion layer fl_2 and the last LSTM. Originally, the temporal feature extracted from each body part via LSTM (rl_2), *i.e.*, y_r^t, where $r = 1,\ 2$ *and* 3, in Figure 7(a), is equally fused into $z = \{z_1^t, z_2^t, z_3^t\}$, where each z_r^t is from y_r^t. This feature vector z serves as the input x^t of the last LSTM. But with attention in Figure 7(a), each z_r^t is differentiated in x^t as follows [53]:

$$x^t = \phi(\{z_r^t\}, \{\alpha_r^t\}), \quad (2)$$

where $\phi(\cdot)$ aims to combine the feature vector $z = \{z_r^t\}$ and their corresponding weights $\{\alpha_r^t\}$. As explored in prior studies [1, 53], function $\phi(\cdot)$ is typically computed as the weighted average $x^t = \sum_{r=1}^{3} \alpha_r^t \cdot z_r^t$, where the weights $\alpha^t = \{\alpha_r^t\}$ are obtained by an attention function $f_{att}(\cdot)$ as follows:

$$\alpha^t = f_{att}(z^t, h^{t-1}), \quad (3)$$

where h^{t-1} is an internal descriptor in the last LSTM (rl_3) for describing the decoded features for each activity to be recognized until the previous time step $t-1$, in Figure 7(b). The key insight from the

aforementioned equations is that the value of each weight α_r^t differentiates the contributions of every network input to the recognition of each gesture. In Eqn. (3), the attention component applies $f_{att}(\cdot)$ to quantify the likelihood of the current feature $\{z_r^t\}$ (from every body part) in aligning with the activity descriptor h^{t-1} until the last time step $t-1$ on the basis of the current features extracted from each individual body part $z = \{z_r^t\}$. Function $f_{att}(\cdot)$ essentially prioritizes the contribution of each z_r^t (thus far) to determine the correct output from all the activity candidates. After the iterative updating (as t increases) in Eqn. (3), the weight α_r^t for a highly contributed z_r^t will be gradually increased in $x^t = \sum_{r=1}^{3} \alpha_r^t \cdot z_r^t$ by function $f_{att}(\cdot)$. In case some inputs are missing, its associated weight α_r^t will be gradually decreased by the attention function, *i.e.,* the missing parts will be marginally considered by the network. Hence, one network structure can address all types of missing input.

The attention function $f_{att}(\cdot)$ essentially functions similar to another neural network to fulfill weight adaptation [1]. To reduce computation, $f_{att}(\cdot)$ is typically realized as a single-layer multilayer perceptron, such as $\tanh(\cdot)$ and $ReLu(\cdot)$ [1, 53]; however, it still follows the same principle as a complete neural network, *i.e.,* all the inputs are first linearly combined and then undergo the nonlinear function. In ArmTroi, the $f_{att}(z^t, h^{t-1})$ function is designed as:

$$att^t = Relu(z^t + W \cdot h^{t-1} + b), \quad (4)$$

$$\alpha^t = f_{soft\text{-}max}(U \cdot att^t), \quad (5)$$

where att^t is an intermediate variable; $\alpha^t = \{\alpha_r^t\}$, $f_{soft\text{-}max}(\cdot)$ scales the weights α^t to the range [0,1]; and W, b and U are the parameters to be determined in the training phase.

5 PERFORMANCE EVALUATION

5.1 Experiment Setup

We develop ArmTroi using LG watches (with Invensense MPU-6515 six-axis motion sensors), Google Glass, SAMSUNG Galaxy S7, and a desktop with Intel i7-6700 CPU and Nvidia GTX 1080Ti GPU.

Implementation. For the skeleton tracking part, we develop all the designs in §3 and deploy them on the smartphone. We also deploy a desktop version to demonstrate the possibility of further improving the performance when a more powerful CPU is available. In particular, we increase the density of cloud points on the desktop by 1.4x compared with the mobile version. For the HMM formulation in the search, we set three probabilities as suggested in [41]. 1) The prior probability $\mathbf{P}_{pri}$ is uniform for both rounds because we are unaware of the initial elbow location, and each location is possible. 2) The transition probability $\mathbf{P}_{tra}$ is Gaussian [41] for the errors between the computed $\overline{acc}_{elb}$ and the measured acc_{elb}. 3) The emission probability $\mathbf{P}_{emi}$ is equal to 1 because $\mathbf{P}_{tra}$ already contains acceleration observations. In addition, we implement ArmTroi's deep learning network using TensorFlow. After network development and training on the desktop, the executable network is deployed on the smartphone.

Methodology. We recruit 7 volunteer users (3 females and 4 males), and their information to generate their point clouds (from user 1 to 7) is as follows: a) Torso length: 52.6, 54.9, 55.2, 57.0, 56.1, 48.7, 47.2; b) Shoulder breadth: 33.6, 34.6, 36.4, 41.4, 33.6, 31.2, 32.4; c) Upper arm: 27.1, 26.3, 26.1, 25.2, 27.1, 25.2, 24.2; d) Lower arm: 23.9, 23.3, 22.8, 24.3, 23.9, 22.1, 21.5, where the unit of all these lengths is *cm*. Point clouds are personalized to be used in the hierarchical search.

Before the experiment, a short tutorial is provided for the devices and all the 17 activities in Table 1, which form two activity sets: 10 for free-weight (FW) and 7 for daily activities (DA). Note that DA contains daily and customized gestures in Table 1. Then, the users perform each activity 30 times and they will manually start and end the data collection for each activity (obtaining labels for gesture recognition evaluation). We use Kinect 2.0 to collect ground truth and observe each activity takes 5 to 10 *s* usually. For skeleton tracking, we also collect additional free motions of the user's arms.

For the deep learning evaluation, we train two networks for the FW and DA activity sets, respectively, used for all seven users. The input data size contains five data stream channels (Figur 6) with the 10-second input data batch size. Each layer has 64 and 256 neurons for DA and FW networks, respectively. We use 70% of the data collected from five users for training. Then we evaluate ArmTroi's performance on the rest 30% of the data from these five users and also other two users, whose data are not used in the training.

5.2 Skeleton Recovery Performance

In this subsection, we first evaluate the overall skeleton recovery accuracy by comparing the following methods:

- **ArmTroi**: our design and its desktop version (ArmTroi-D), where both versions work in real time.
- **ArmTrak** [41]: state-of-the-art method primarily working on a desktop with nearly 10x latency increase.

Overall recovery accuracy. Figure 8(a) shows the CDF of skeleton errors for the elbow and wrist (including both arms), respectively, using ArmTrak on desktop and ArmTroi on smartphone. When the sensor data is input to ArmTrak and ArmTroi, we let both of them to traceback at the current time stamp in the HMM search to report locations every second (five intermediate location values in the last second). ArmTroi could output results in real time, while ArmTrak can only conduct an offline recovery due to high latency. Overall, ArmTroi outperforms ArmTrak for elbow and wrist tracking. For the elbow, the median errors of ArmTrak and ArmTroi are 12.94 and 10.53 *cm*, respectively. For the wrist, their errors are slightly higher[4]. Compared with ArmTrak, ArmTroi can reduce the elbow and wrist recovery errors by 18.6% and 13.2%, respectively.

To further understand the design of two methods, Figure 8(b) also plots the tracking accuracy when HMM traceback is performed over each entire motion sequence (global optimization [41]), hence intermediate HMM states benefit from the future data [41]. The result shows that the elbow (wrist) errors for ArmTrak and ArmTroi can be reduced to 11.45 (14.28) and 9.6 (11.64) *cm*, respectively.

In Figure 8(c), we further observe that if ArmTroi is executed using the same desktop CPU as ArmTrak, the elbow and wrist errors (ArmTroi-D) can be further reduced to 8.56 and 11.56 *cm* respectively, as we can accommodate denser point clouds in the search due to the decreased computation complexity.

Parameter setting. In Figure 9(a), we examine the parameter setting of ArmTroi's search, *i.e.,* the manner by which many cloud points are grouped as one region for the first round search. For each

[4]Similar as [41], this might be due to Kinetic's sensitivity to different joints.

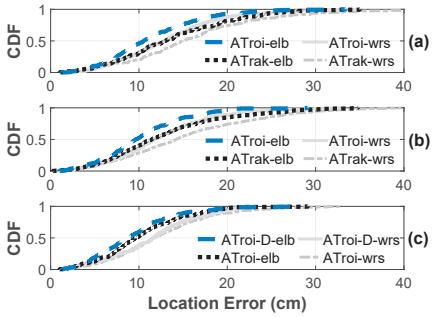

Figure 8: (a) CDF of methods' skeleton recovery errors. (b-c) Understand tracking performance in various settings.

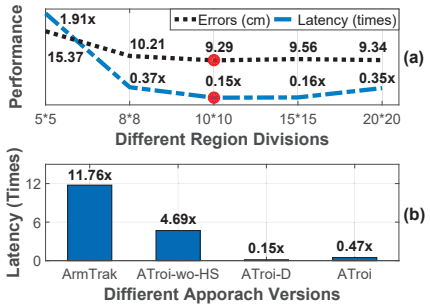

Figure 9: (a) Setting for ArmTroi's hierarchical search. (b) Detailed latency reduction gains in ArmTroi.

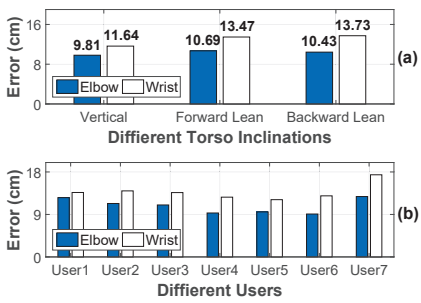

Figure 10: Skeleton recovery errors (a) under different torso inclinations and (b) for different users.

point cloud, points are plotted on a sphere (as depicted in Figure 3). Thus, we divide the sphere into different regions to examine this parameter's setting in Figure 9(a). If the division is coarse, *e.g.,* $5 * 5$, then the final recovery error is large because each region is only represented by its centroid in the first round search, and the selected regions from the first round can be highly inaccurate. Meanwhile, the latency is also high because the search space (the size of each selected region) is large as well. As shown in Figure 9(a), we observe that when region division is sufficiently dense, *e.g.,* $10 * 10$, small errors and latency can be simultaneously achieved. If the division becomes even denser, then the error remains stable, but latency will increase again. Thus, we select $10 * 10$ as the default setting.

In Figure 9(b), we further examine the performance gains of two complexity reduction techniques proposed in ArmTroi. We set the latency of ArmTrak as the benchmark. In our implementation, an activity with duration t requires $t \times 11.76$ times to be recovered by ArmTrak for one arm. However, the latency is decreased to $t \times 4.69$ with only HMM state remodeling, *i.e.,* without the hierarchical search ("ATroi-wo-HS" in the figure). For the full version of ArmTroi on the desktop ("ATroi-D"), the latency is $t \times 0.15$. Even on the mobile phone ("ATroi"), the latency is $t \times 0.47$. Figure 9(b) indicates the efficacy of our complexity reductions, which could thus support real-time tracking for both versions.

Body inclinations and different users. In Figure 10(a), we further examine the activities in the free-weight category that involves different torso inclinations. We find that compared with the case when the user's torso is vertical, *e.g.,* standing or sitting, the median elbow and wrist errors slightly increase by 1.09x and 1.06x, respectively, due to torso inclinations. In Figure 10(b), we also plot the detailed recovery errors for all seven users, where ArmTroi consistently performs well for different users, *i.e.,* median elbow and wrist errors are 10.53 *cm* and 12.94 *cm* respectively.

In the context of edge-based services. We compare ArmTroi finally in the context of edge-based services, with a recent design MUSE [40]. Figure 11(a) shows that MUSE can achieve a slightly lower tracking error than ArmTroi, where the median errors of MUSE and ArmTroi are 9.69 *cm* and 10.53 *cm*, respectively. We also configure two methods to output locations every second (five location values in the last second). Figure 11(b-c) depicts that ArmTroi can output immediately in real time, while for MUSE, its output time is postponed and the latency increases, since it needs an extra $1.1 \times t$ latency to generate output for a t-time activity. Figure 11 inspires the benefits of ArmTroi to the applications from privacy and efficiency two aspects compared with edge-based systems (§6).

Figures 8 to 11 show ArmTroi achieves good recovery performance, thereby benefiting both gesture inference and applications.

5.3 Gesture Recognition Performance

In this subsection, we continue to examine the deep learning part by comparing the following methods:

- **ArmTroi**: our method with the attention component.
- **Lasagna** [24]: directly uses raw, sparse sensing data.
- **MULT**: we train multiple networks (*e.g.,* six). Each network handles one combination of missing inputs: 1) complete inputs, 2) input from the left or right arm (LA/RA), 3) inputs from two body parts (three combinations).
- **Dropout** [49]: a recent design that uses the dropout technique to cope with the missing data issue.

Overall accuracy. In Figure 12(a), with the complete set of input sensors for FW and DA (two activity sets), Lasagna achieves 83.9% and 89% for the two data sets, and ArmTroi can further improve the results by 8.8% and 2.4%, respectively. This finding is attributed to the recovered postures embracing "out-of-band" kinematic information, which is not owned by raw data. With the complete set of sensor inputs, ArmTroi and MULT can achieve comparable performance, *e.g.,* 91.4% to 92.7%, considering that input information is complete in this case. Figure 12(b) plots the accuracy of ArmTroi on different users. In general, ArmTroi exhibits a stable performance with a high accuracy across all users. The performance is slightly worse for the last two users because their data are not used in the training phase. We envision leveraging semisupervised learning to improve their performance in the future.

Comparison with MULT. In Figure 13 (a), we further examine the impact of missing inputs on the inference accuracy of ArmTroi. As a benchmark, we also plot the performance of MULT, leading to the best performance can be achieved when input missing happens. We find that with the attention-based design, ArmTroi can promisingly achieve comparable performance as training separate networks. In Figure 13, FW, DA, and AT represent free weight, daily activity, and

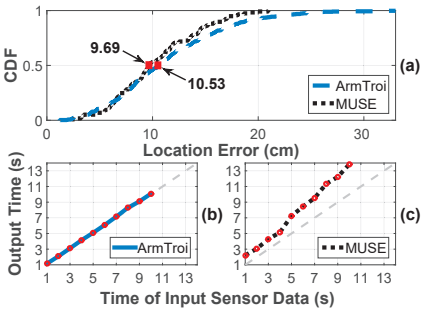

Figure 11: Performance comparison between ArmTroi and MUSE for (a) accuracy, and (b-c) latency.

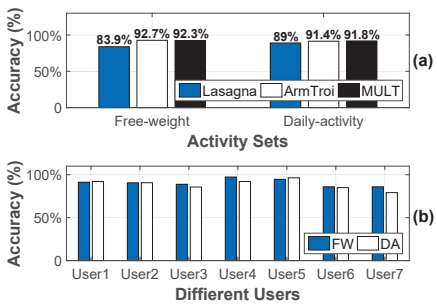

Figure 12: Accuracy comparisons (a) among three different approaches, and (b) among different users.

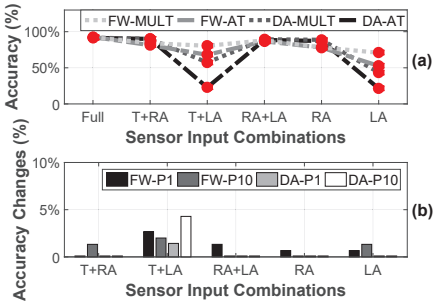

Figure 13: Accuracy comparisons with (a) MULT, and (b) different padding values for missing sensors in ArmTroi.

ArmTroi, respectively. When only the sensing data from two body parts are available (*e.g.,* LA, RA and T stand for left-arm, right-arm and torso, respectively), although ArmTroi never sees such input combinations in the training phase, the accuracy losses of ArmTroi for T+RA and RA+LA are moderate (*e.g.,* <3%) compared with those of MULT, which is benefitted by the attention-based network adaptation. For T+LR, both methods are considerably degraded because right-hand gestures dominate in most of these activities in our daily life. Moreover, when only the sensing data from one arm is available, the accuracy could drop more. In §5.4, we further consider and address this issue in our first case study.

Different padding values for missing inputs. When the input missing occurs, we need to pad certain values for these missing inputs. Padding zeros could be a natural option (our default setting), while we also examine the performance by padding non-zero values. In Figure 13(b), we treat the accuracy achieved by padding zero as a baseline and plot the accuracy differences by padding non-zero values one (FW-P1 and DA-P1) and ten (FW-P10 and DA-P10). We can see from the result that the accuracy differences are nearly 0% for most input combinations. This indicates the attention component can effectively adjust the weight vector to match the currently available inputs, regardless of padding values.

Different strategies to handle missing inputs. In the literature, there are two possible ways to handle missing inputs (using one network): 1) Dropout [49] and 2) one neural network trained using the data with all types of input missing combinations (*i.e.,* ArmTroi without attention). We find that the second method achieves the worse performance, which is thus used as the baseline to examine ArmTroi and Dropout. In Figure 14(a), for the FW activity set, ArmTroi outperforms Dropout on all sensor input combinations from 0.7% to 12%. For the DA activity set in Figure 14(b), ArmTroi outperforms Dropout on all sensor input combinations from 1.4% to 7.4%. Dropout is more effectively when there are multiple sensors providing redundant sensor readings [49], while our system setting does not include redundant sensors. The results show that ArmTroi can handle this missing sensor issue effectively.

Efficacy of attention component. In Figure 15, we further analyze the performance of the attention-based design. We first show a detailed attention weight updating process to match the missing input data. In Figure 15(a), only LA's sensing data are available. Originally, the weights of three body parts have their initial values (being different in the recognition of different activities). After LA's sensing data are entered into the network, the three weights are gradually adjusted by the attention function. In particular, the weight for RA is originally small and remains constant as RA's input is missing. The weight for T is large in the beginning. However, its corresponding input is also missing, and thus, its weight keeps decreasing. LA's weight gradually increases and finally overwhelms the other two to generate the output.

In Figure 15(b), we provide more examples (*e.g.,* 20), wherein each red square represents the sensor input to the network. During the network's execution, we record all the weights in the attention component for each body part and plot the dominant weight as "x" in Figure 15(b) when we obtain the inference output. The result shows that the weights can match the input's availability, *i.e.,* network can achieve correct attention to the most meaningful inputs to generate the output, thereby ensuring good performance against the input's varying availability.

Analysis of sensing and learning designs. ArmTroi contains the designs from the two major aspects of wearable sensing (skeleton tracking) and deep learning (with attention-based adaptation). We then analyze ArmTroi's overall performance losses each aspect contributes to. In Figure 16(a), after we replace the recovered skeletons with the ground truth collected using Kinect, denoted as "Kinect-ATroi", the accuracy of gesture inference (still using our deep learning design) improves by approximately 1.4%. If we further use MULT to replace our deep learning part, denoted as "Kinect-MULT", the best performance can be achieved when input is missing, and accuracy further increases by up to 5.5%. This result indicates our sensing and learning designs contribute to 20.3% and 79.7% overall performance losses, respectively.

Effect of input setting. We further study the effect of different network input settings on performance. As depicted in Figure 6 on page 6, two types of inputs are available for each body part, *i.e.,* the skeleton itself and the skeleton difference across time. In Figure 16(b), we individually investigate the two aspects by removing the skeleton inputs ("ATroi-wo-S") and skeleton difference inputs ("ATroi-wo-SD") from the network. Figure 16(b) shows that atop the

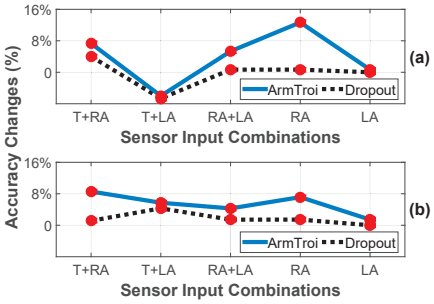

Figure 14: Accuracy comparisons among ArmTroi, Dropout and w/o attention on (a) FW and (b) DA activity sets.

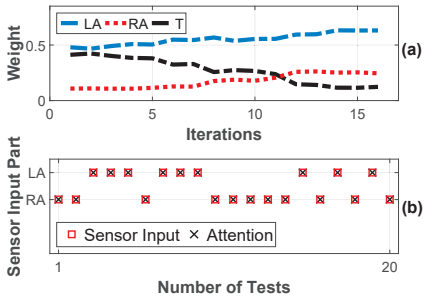

Figure 15: (a) Illustration of attention weight changes. (b) Matching between sensor inputs and the dominant weight.

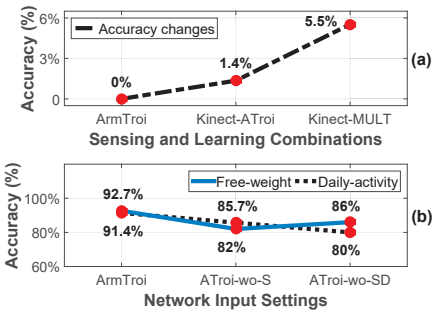

Figure 16: (a) Performance losses from sensing and learning parts respectively. (b) Impact of different input settings.

original skeleton inputs, explicitly providing the skeleton difference can improve performance by 7% to 11% because each gesture is a sequence of meaningful arm motions.

5.4 Case Studies

5.4.1 Fitness Activity Assessment. Many recent designs propose to use wearables to facilitate fitness[6, 12] like Figure 17. Two main tasks such applications are: *fitness guidance* and *activity recognition*.

Fitness guidance. With the arm skeleton information, the fitness coach can provide the trainee with a detailed free-weight activity template that illustrates the angle information of major joints in each arm. A substantially concrete guidance can be achieved for the trainee to improve his/her performance by visualizing the activities of the coach and trainee. The coach can also provide a detailed assessment score for each activity, as shown in Figure 18 for the comparisons of two shoulder's (a and b) and two elbow's (c and d) angles between the same activities of the coach and the trainee. The high assessment score based on such a fine-grained comparison indicates that the trainee performs well, which is considerably more indicative and useful than using the raw, sparse sensing data.

Long-term recovery errors. Figure 19(a) shows the location errors of the wrist and elbow along time for a long-term activity (lasting for 2 *min*). The results show that the errors are bounded rather than continuously increased. This property, *i.e.,* errors are not accumulated over time, is also important for such applications.

Semantic label recognition. Another task for fitness is to recognize training activities as a part of the fitness log for quantifying workout outcome, *e.g.,* calorie consumption, and understanding or updating the fitness plan. In §5.3, when certain inputs are missing, some activities may not be reliably recognized even with our attention adaptation. Instead of recording these potentially less reliable results into the fitness log, we observe the opportunity to benefit output utility for this case study. Lasagna [24] recently observes that many human activities have layered semantic meanings. Free-weight activities evidently have this attribute. Figure 17 depicts a partial (due to the page limit) layered semantic meaning tree for the free-weight activities listed in Table 1. We incorporate this idea, with an extended design as follows, into ArmTroi for this case study to determine the appropriate semantic meaning as output, instead of insisting on the original activities but producing incorrect results. To this end, for each input combination, we compute the true positive r_i^{tp} and false positive r_i^{fp} rates for every activity A_i in the training set on the basis of their labels. Then, we can mark the semantic tree using two steps.

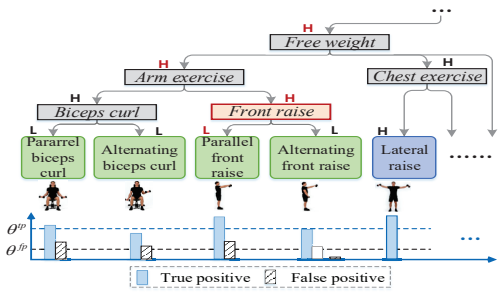

Figure 17: Illustration of a (partial) semantic tree covering the activities in the free-weight activity set.

Step 1: If a leaf node (activity A_i) has a high true (*e.g.,* $r_i^{tp} \geq \theta^{tp}$) and low false (*e.g.,* $r_i^{fp} < \theta^{fp}$) positive rates, it is marked as "H" (High confidence); otherwise, it is marked as "L" (Low confidence), where θ^{tp} and θ^{fp} are empirically set as 0.6 and 0.4 in §5, respectively. Thus, all the leaf nodes are marked by either "H" or "L".

Step 2: All the intermediate nodes can also be recursively marked by either "H" or "L" from the bottom (leaves) to the top (root). For each intermediate node, if all of its children nodes are "H", then it is immediately marked as "H"; otherwise, it must contain "L" children. Then, we virtually aggregate all its children nodes as one and calculate the true and false positive rates for this parent node, *i.e.,* whether the higher-level semantic meaning can be reliably detected. If yes, this parent node is marked as "H"; otherwise, "L".

After the two steps above, all semantic tree nodes are marked by "H" or "L". Then, for the input data in real usage, we can first apply the network to obtain its original inference output. If the output is on an "H" leaf node, then we consider it the final result; otherwise, we will trace back on the marked tree and output the first encountered "H" node. Figure 19(b) shows that with this design, recognized semantic labels become highly accurate for nearly all

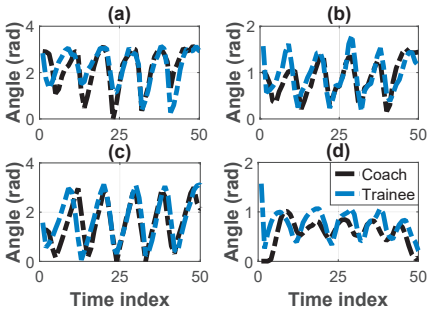

Figure 18: Comparisons between the coach's and a trainee's activities for a detailed assessment.

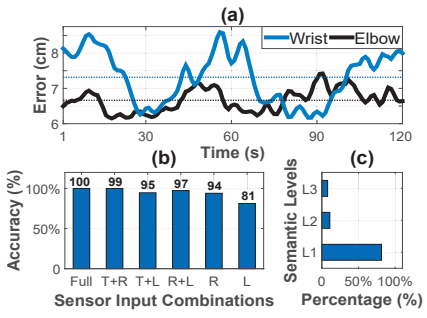

Figure 19: (a) Location errors of a long-term activity. (b) Semantic-level accuracies. (b) Percentages on each level.

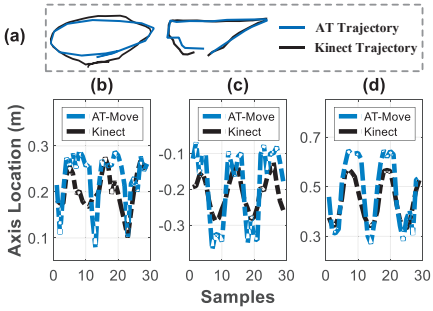

Figure 20: (a) Gesture-based command instances. (b-d) Recovered elbow locations in 3 axes during the walk.

sensor input combinations. Figure 19(c) further plots for all those correctly recognized activities, 81% and 19% are from leaf (L1) and parent nodes, respectively.

5.4.2 Gesture-based Control. In this subsection, we develop one more case study, in which certain arm gestures are predefined as the commands used for the gesture-based control in the smart space.

Figure 20(a) first depicts several commands that the user performs, including a circle and triangle, which are close to the ground truth measured from the Kinect. From Figure 20(b) to (d), we further evaluate gesture command recovery when the user is walking, *e.g.,* circling in the air. In the experiments, if the glass gyroscope indicates the user's head is not rotating, then the acceleration of the smart glasses is close to the body during the walk. Thus, we deduct the acceleration data of the glass from those of the smart watch before skeleton recovery. Figure 20(b-d) shows that the tracking error only increases about 1 *cm* compared with the static setting.

5.5 System Overhead

Memory usage and execution time. For the skeleton tracking part, the average memory usage values are 780 MB and 450 MB on the desktop and smartphone, respectively, which can be substantially accommodated by existing mobile platforms. The execution time is already reported in Figure 9(b). For the deep learning part, the memory usage and execution time are 419 MB and 169 *ms*, respectively, thereby indicating ArmTroi's deep learning design is also lightweight. In addition, we also examine the execution time on the desktop, which can be further decreased to 76 *ms*.

Energy consumption. We measure the energy consumption of ArmTroi on SAMSUNG S7 by Monsoon power monitor in Figure 21. As a benchmark, the device's energy consumption, *i.e.,* working current (*mA*), in idle state with screen on is about 90 *mA*.

Similar as the setting in §5.2, ArmTroi conducts the hierarchical search every second to report five location values in the last second. Figure 21 shows the energy consumption for a 10-second activity tracking ("Track-Full"). To have a comprehensive understanding of this energy profile, we further configure ArmTroi in two modes, "Track-State" and "Track-First", to measure the energy consumption of the smart watch state estimation (*e.g.,* wrist orientation, etc.) and the first-round HMM search only, respectively. The result shows that "Track-State" increases the working current up to around 350 *mA*, and "Track-First" further increases it to more than 1000 *mA*. Figure 21 also unveils that "Track-Full" consumes similar energy as "Track-First", which implies that the second-round search does not incur extra energy overhead obviously. When the hierarchical search is launched in "Track-Full", the average working current is around 480 *mA* and the peak value climbs up to around 1100 *mA*, while it lasts for less than 0.5 *s* due to our HMM acceleration design. The average working current of "Track-Full" is only 270 *mA*.

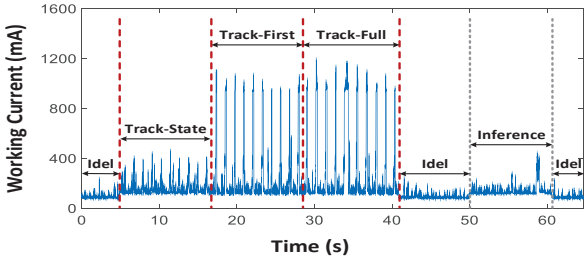

Figure 21: Energy consumption measurement for ArmTroi.

In Figure 21, we also examine the energy consumption of the deep learning network ("Inference"), which takes a 10-second skeleton traces as the input data batch (§5.1). The result shows that its working current is about 400 *mA* (lasting less than 200 *ms*). On the basis of this experiment, we also discuss the possible ways in the future to further reduce the ArmTroi's energy consumption in §6.

6 POINTS OF DISCUSSION

Pre-knowledge of user information. Similar as the previous work ArmTrak [41], our system ArmTroi also requires the pre-knowledge of some user-specific information to generate the point clouds for each individual user, including the torso length, shoulder breadth, upper-arm length and lower-arm length. Fortunately, the generation of point clouds is an one-time effort for each user.

Energy consumption. To further improve the energy efficiency of ArmTroi, two potential ways can be explored in the future. For skeleton tracking, we can leverage the continuity of arm's motion to also reduce the search space. Supposing that we have estimated the current arm location, its near-future location will not be extremely

far away and we can thus directly shrink the first-round search range. Therefore, two aspects can be examined in the future: 1) the closeness of the first-round search range cross consecutive time stamps, and 2) how often this space reduction mechanism can be applied. For the gesture recognition, recent neural network compression techniques, such as [25, 54], can effectively reduce the network size, thereby decreasing the energy consumption, but still preserve good accuracy. This opportunity can also be explored for the gesture recognition design in ArmTroi in the future.

Tracking accuracy. Recently, people can utilize cameras to track user's 2D upper limbs or even the whole body with errors of about 2 *cm* to enable person re-identification, video re-targeting and robot teaching applications [3, 56]. Meanwhile, 3D recovery is achieved using depth cameras [55] or VICON [43] for the applications, such as the sport analysis with 2.5 *cm* error. Recent studies, like [56], also propose to use wireless signals and deep learning to enable the through-the-wall tracking with the error of 7.6 *cm*. Camera or wireless devices typically have a limited service coverage. They may also have higher system costs and/or privacy concerns. However, they generally achieve higher accuracy than the wearable-based methods, which indicates the room that the wearable-based methods can be improved. Further improving the tracking accuracy will be one useful and important next step in the future.

Relation to edge-based services. ArmTroi can also benefit even when the edge-based service becomes popular from two aspects. First, the application provider could leverage the lightweight computation of ArmTroi to support more users simultaneously with the same CPU resource consumption. The second benefit appears when the edge-based system is not available or not preferred to be used. Moreover, even the edge-based service is available, some users, who have a high standard on the user privacy and data security, may still not be willing to use such a third-part platform. In this case, the local execution on the user's phone could be a preferred option.

7 RELATED WORK

Sensor-based skeleton recovery. In the literature, the studies [36, 46] leverage multiple sensors to recover body pose, but performance highly depends on the similarity of testing and training data due to methodology limitation [41]. Existing works [4, 8] have focused on upper limbs, but they also require multiple sensors on the arm.

ArmTrak [41] is one of the most practical wearable-based solutions at present. It recovers user's arm motions using only a smart watch. However, its recovery latency is long (10x). Recently, the authors of [41] propose MUSE [40], which achieves slightly better location estimation accuracy than ArmTrak and also cuts computation. However, according to [40] and our evaluation in §5, MUSE achieves slightly better tracking accuracy than ArmTroi but it is still not sufficiently lightweight for both desktop and smartphone. As discussed in §6, ArmTroi provides the opportunity to enhance applications from privacy and efficiency perspectives in practice.

With regard to techniques, HMM has long been applied to infer user's pose [20]. Prior works [20, 34] mostly apply HMM to classify various activities, but precise skeleton tracking using wearable sensors is considerably challenging [41]. Various efforts are currently available to accelerate HMM in a hierarchical search. However, technical insights and challenges vary for different input data and applications. For example, image data can be projected from 3D to 2D for activity recognition to accelerate HMM search [31], and words' correlation can be used for speech recognition [5]. Our proposed HMM acceleration design has not been explored before, wherein the hierarchical search technique can be further used for different the trajectory estimation problems.

Missing deep learning input data. Recent studies find that missing input data may degrade deep learning's performance [48]. Training separate networks for each combination of missing inputs [11, 42] is nonscalable and also storage cost inefficient [49]. Some existing work has studied the missing input data issue using domain knowledge, such as the joint utilization of video and audio for speech classification [32], input selection for speech recognition with the neural network trained with the input stream dropout [28], and clinical diagnosis [23]. For mobile and wearable sensors, recent multilayer perceptron [49] finds that when multiple modalities of sensors serve as the deep learning's inputs (with redundancy), the dropout technique [44] can be effectively resilient to losing certain inputs. However, ArmTroi does not include redundant sensors.

The attention technique is widely used in the fields of natural language generation and visual object recognition [45]. Typical applications include language translation [1], news summarizer [37], and picture caption generation [53], mostly using textual contents or images as inputs to generate target outputs. ArmTroi leverages this technique to address the missing input data issue in a novel way that one network can behave like multiple networks to automatically handle all types of missing inputs. This ability can be applied to other cases when multiple devices' data need to be fused.

Activity recognition by deep learning. Deep learning is widely applied in applications [6, 9, 14, 16, 24, 50, 52] without using handcrafted features [2, 22]. Parallel to the application designs, numerous network improvement designs are also available [29], *e.g.,* resource optimization [10, 15, 19] and network compression [21, 25, 54].

8 CONCLUSION

This paper presents a wearable system, called ArmTroi, to understand and analyze user's arm motions. We propose novel techniques to enable real-time 3D arm skeleton tracking and gesture inference tolerant to missing wearable sensors. The ArmTroi design can serve as a generic platform to enable numerous arm motion-oriented applications. We implement an ArmTroi prototype. Extensive evaluations on the prototype, along with two concrete case studies, demonstrate the efficacy of ArmTroi, which achieves promising tracking and inference performance.

ACKNOWLEDGEMENTS

We sincerely thank our shepherd, Dr. Aruna Balasubramanian, and anonymous reviewers for their helpful comments. This work is supported by the ECS (CityU 21203516) and GRF (CityU 11217817) grants from Hong Kong RGC. It is also supported by NSFC Grant (No.61802261 and No.61872248), NSF Grant of Shenzhen University (No.2018061), Tencent "Rhinoceros Birds" - Scientific Research Foundation for Young Teachers of Shenzhen University, Guangdong NSF 2017A030312008, Fok Ying-Tong Education Foundation for Young Teachers in Higher Education Institutions of China (No.161064) and GDUPS (2015). Corresponding author is Zhenjiang Li.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

[2] Yoshua Bengio. 2013. Deep learning of representations: Looking forward. In *Proc. of Springer SLSP*.

[3] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. 2017. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In *Proc. of IEEE CVPR*.

[4] Andrea Giovanni Cutti, Andrea Giovanardi, Laura Rocchi, Angelo Davalli, and Rinaldo Sacchetti. 2008. Ambulatory measurement of shoulder and elbow kinematics through inertial and magnetic sensors. *Springer Medical & biological engineering & computing* (2008).

[5] Neeraj Deshmukh, Aravind Ganapathiraju, and Joseph Picone. 1999. Hierarchical search for large-vocabulary conversational speech recognition: working toward a solution to the decoding problem. *IEEE Signal Processing Magazine* (1999).

[6] Han Ding, Longfei Shangguan, Zheng Yang, Jinsong Han, Zimu Zhou, Panlong Yang, Wei Xi, and Jizhong Zhao. 2015. Femo: A platform for free-weight exercise monitoring with rfids. In *Proc. of ACM SenSys*.

[7] Yong Du, Wei Wang, and Liang Wang. 2015. Hierarchical recurrent neural network for skeleton based action recognition. In *Proc. of IEEE CVPR*.

[8] Mahmoud El-Gohary and James McNames. 2012. Shoulder and elbow joint angle tracking with inertial sensors. *IEEE Transactions on Biomedical Engineering* (2012).

[9] Biyi Fang, Nicholas D Lane, Mi Zhang, Aidan Boran, and Fahim Kawsar. 2016. BodyScan: Enabling radio-based sensing on wearable devices for contactless activity and vital sign monitoring. In *Proc. of ACM MobiSys*.

[10] Petko Georgiev, Nicholas D Lane, Kiran K Rachuri, and Cecilia Mascolo. 2016. LEO: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources. In *Proc. of ACM MobiCom*.

[11] John J Guiry, Pepijn Van de Ven, and John Nelson. 2014. Multi-sensor fusion for enhanced contextual awareness of everyday activities with ubiquitous devices. *Multidisciplinary Digital Publishing Institute Journal on Sensors* (2014).

[12] Xiaonan Guo, Jian Liu, and Yingying Chen. 2017. FitCoach: Virtual fitness coach empowered by wearable mobile devices. In *Proc. of IEEE INFOCOM*.

[13] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2014. Towards wearable cognitive assistance. In *Proc. of ACM MobiSys*.

[14] Nils Yannick Hammerla, James Fisher, Peter Andras, Lynn Rochester, Richard Walker, and Thomas Plötz. 2015. PD Disease State Assessment in Naturalistic Environments Using Deep Learning.. In *Proc. of AAAI*.

[15] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proc. of ACM MobiSys*.

[16] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. 2013. Accelerometer-based transportation mode detection on smartphones. In *Proc. of ACM SenSys*.

[17] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* (1997).

[19] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In *Proc. of ACM MobiSys*.

[20] Doo Young Kwon and Markus Gross. 2007. A framework for 3D spatial gesture design and modeling using a wearable input device. In *Proc. of ACM ISWC*.

[21] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proc. of ACM/IEEE IPSN*.

[22] Oscar D Lara and Miguel A Labrador. 2013. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials* (2013).

[23] Zachary C Lipton, David C Kale, and Randall Wetzel. 2016. Modeling missing data in clinical time series with rnns. *Machine Learning for Healthcare* (2016).

[24] Cihang Liu, Lan Zhang, Zongqian Liu, Kebin Liu, Xiangyang Li, and Yunhao Liu. 2016. Lasagna: towards deep hierarchical understanding and searching over mobile sensing data. In *Proc. of ACM MobiCom*.

[25] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-Demand Deep Model Compression for Mobile Devices: A Usage-Driven Model Selection Framework. In *Proc. of ACM MobiSys*.

[26] Roanna Lun and Wenbing Zhao. 2015. A survey of applications and human motion recognition with microsoft kinect. *World Scientific on International Journal of Pattern Recognition and Artificial Intelligence* (2015).

[27] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. of EMNLP*.

[28] Sri Harish Mallidi and Hynek Hermansky. 2016. Novel neural network based fusion for multistream ASR. In *Proc. of IEEE ICASSP*.

[29] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware. In *Proc. of ACM MobiSys*.

[30] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*.

[31] Ramanan Navaratnam, Arasanathan Thayananthan, Philip HS Torr, and Roberto Cipolla. 2005. Hierarchical Part-Based Human Body Pose Estimation.. In *Proc. of BMVC*.

[32] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *Proc. of ICML*.

[33] Qifan Pu, Sidhant Gupta, Shyamnath Gollakota, and Shwetak Patel. 2013. Whole-home gesture recognition using wireless signals. In *Proc. of ACM MobiCom*.

[34] Muhannad Quwaider and Subir Biswas. 2008. Body posture identification using hidden Markov model with a wearable sensor network. In *Proc. of ICST BodyNets*.

[35] Nancy Berryman Reese and William D Bandy. 2016. *Joint Range of Motion and Muscle Length Testing-E-Book*. Elsevier Health Sciences.

[36] Qaiser Riaz, Guanhong Tao, Björn Krüger, and Andreas Weber. 2015. Motion reconstruction using very few accelerometers and ground contacts. *Elsevier Graphical Models* (2015).

[37] Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proc. of EMNLP*.

[38] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* (1997).

[39] Chew Zhen Shan, Eileen Su Lee Ming, Hisyam Abdul Rahman, and Yeong Che Fai. 2015. Investigation of upper limb movement during badminton smash. In *Proc. of IEEE ASCC*.

[40] Sheng Shen, Mahanth Gowda, and Romit Roy Choudhury. 2018. Closing the Gaps in Inertial Motion Tracking. In *Proc. of ACM MobiCom*.

[41] Sheng Shen, He Wang, and Romit Roy Choudhury. 2016. I am a Smartwatch and I can Track my User's Arm. In *Proc. of ACM MobiSys*.

[42] Muhammad Shoaib, Stephan Bosch, Hans Scholten, Paul JM Havinga, and Ozlem Durmaz Incel. 2015. Towards detection of bad habits by fusing smartphone and smartwatch sensors. In *Proc. of IEEE PerCom Workshops*.

[43] Leonid Sigal, Alexandru O Balan, and Michael J Black. 2010. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *Springer Journal on International journal of computer vision* (2010).

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* (2014).

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proc. of NIPS*.

[46] Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. 2011. Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics* (2011).

[47] Edison Thomaz, Irfan Essa, and Gregory D Abowd. 2015. A practical approach for recognizing eating moments with wrist-mounted inertial sensing. In *Proc. of ACM Ubicomp*.

[48] Yonatan Vaizman, Katherine Ellis, and Gert Lanckriet. 2017. Recognizing detailed human context in the wild from smartphones and smartwatches. *IEEE Pervasive Computing* (2017).

[49] Yonatan Vaizman, Nadir Weibel, and Gert Lanckriet. 2018. Context Recognition In-the-Wild: Unified Model for Multi-Modal Sensors and Multi-Label Classification. *Proc. of the ACM on IMWUT* (2018).

[50] Praneeth Vepakomma, Debraj De, Sajal K Das, and Shekhar Bhansali. 2015. A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities. In *Proc. of IEEE BSN*.

[51] Tran Huy Vu, Archan Misra, Quentin Roy, Kenny Choo Tsu Wei, and Youngki Lee. 2018. Smartwatch-based Early Gesture Detection 8 Trajectory Tracking for Interactive Gesture-Driven Applications. *Proc. of ACM IMWUT* (2018).

[52] Sijie Xiong, Sujie Zhu, Yisheng Ji, Binyao Jiang, Xiaohua Tian, Xuesheng Zheng, and Xinbing Wang. 2017. iBlink: Smart Glasses for Facial Paralysis Patients. In *Proc. of ACM MobiSys*.

[53] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of ICML*.

[54] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework. In *Proc. of ACM SenSys*.

[55] Zhengyou Zhang. 2012. Microsoft kinect sensor and its effect. *IEEE multimedia* (2012).

[56] Mingmin Zhao, Yonglong Tian, Hang Zhao, Mohammad Abu Alsheikh, Tianhong Li, Rumen Hristov, Zachary Kabelac, Dina Katabi, and Antonio Torralba. 2018. RF-based 3D skeletons. In *Proc. of ACM SIGCOMM*.

[57] Pengfei Zhou, Yuanqing Zheng, and Mo Li. 2012. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of ACM MobiSys*.