Toward Realistic Hands Gesture Interface: Keeping it Simple for Developers and Machines

Eyal Krupka[1] Kfir Karmon[1] Noam Bloom[1]
Daniel Freedman[1] Ilya Gurvich[1] Aviv Hurvitz[1]
Ido Leichter[1] Yoni Smolin[1] Yuval Tzairi[1]
Alon Vinnikov[1] Aharon Bar Hillel[2]

[1] Microsoft Research
{eyalk, kfirk, t-noblo, danifree, ilyagu, avivh, idol, yonis, yutzairi, alvinn}@microsoft.com

[2]Ben-Gurion University, Israel
barhille@bgu.ac.il

ABSTRACT

Development of a rich hand-gesture-based interface is currently a tedious process, requiring expertise in computer vision and/or machine learning. We address this problem by introducing a simple language for pose and gesture description, a set of development tools for using it, and an algorithmic pipeline that recognizes it with high accuracy. The language is based on a small set of basic propositions, obtained by applying four predicate types to the fingers and to palm center: direction, relative location, finger touching and finger folding state. This enables easy development of a gesture-based interface, using coding constructs, gesture definition files or an editing GUI. The language is recognized from 3D camera input with an algorithmic pipeline composed of multiple classification/regression stages, trained on a large annotated dataset. Our experimental results indicate that the pipeline enables successful gesture recognition with a very low computational load, thus enabling a gesture-based interface on low-end processors.

ACM Classification Keywords

H.5.2. User Interfaces: Input devices and strategies, Prototyping; I.5.4. Pattern Recognition applications: Computer vision; I.4.9. Image processing and Computer Vision: Applications

Author Keywords

Hand gesture recognition; Hand gesture NUI development

INTRODUCTION

Hand gestures are a natural communication mode for humans, and a promising direction for a human-computer interface. Scenarios of interest range from personal computers to mobile devices and to emerging virtual and augmented reality platforms[4, 8]. In addition, advances in depth camera imaging and computer vision have made such an interface possible in recent years [27, 14, 15, 16, 17, 26]. However, a practical gesture-based interface still faces severe difficulties, due to conflicting demands on both the development and run-time environments. Development of a rich gesture-based interface is currently a considerable effort requiring long development cycles with teams skilled in computer vision and machine learning. A common alternative is to use a mature tuned system (like [14] or [1]), but these typically offer a small predefined set of gestures to choose from, thus limiting the uniqueness and richness of the experience. In the run-time environment, the dilemma is between system accuracy and its computational demands. Several recent articles describe systems that already achieve real time performance using multi-core CPUs. However, a useful system should utilize only a fraction of the CPU power, so it does not disturb other running applications, yet maintain accuracy and responsiveness. These contradicting demands are exacerbated in low-power, low-end CPUs used in mobile devices.

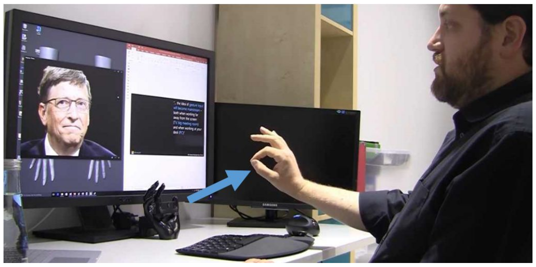


Figure 1. The camera and user setting we address - providing a gesture-based user interface for common laptop and desktop experiences.

We propose to address these difficulties by introducing a simple language for the definition of poses and gestures, and by developing a set of tools and algorithms demonstrating that the language is practical for both development and algorithmic recognition. Our contribution is three-fold. First, we propose a language. In its design, our primary guideline was to keep it simple enough so that development is easy, yet expressive enough to enable most of the gestures coming to one's mind. Second, we developed a set of tools that enable natural development of a gesture-based interface, without prerequisite knowledge in algorithms or machine vision. These tools enable gesture definition using code, XAML files or an editing GUI, and include visualization and verification tools.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CHI'17, May 6–11, 2017, Denver, USA.
Copyright © 2017 ACM ISBN/14/04...$15.00.
DOI http://dx.doi.org/10.1145/3025453.3025508

Third, and most important, we have developed an algorithmic pipeline which enables recognition of the defined gestures from a 3D camera stream, with high accuracy. This pipeline includes many extremely fast predictors, operating in multiple stages, and trained on a large annotated data corpus. With this pipeline, we achieve real time performance with a single thread, using only a small fraction of the CPU.

The language we propose is based on four basic predicates which are naturally used to describe hand poses, applied to the six main interest points of the hand: the five fingertips and the palm center. The predicates are: pointing direction ('the thumb points up'), relative location ('the index is above the middle'), fingertip touching ('the ring finger touches the thumb') and finger flexion ('the pinky is folded'). Using these predicates, 102 basic propositions are created, which serve as the basic binary building blocks of the calculus. A hand pose (sometimes termed 'posture') is defined mainly as a conjunction of the basic propositions, with disjunctions partially allowed in certain cases. A gesture, in turn, is defined plainly as a sequence of hand poses. Defining a gesture in this language is fairly straightforward, due to its proximity to pose description in natural language, and the gesture developer is not asked to state continuous parameters like distances or angles. Despite its qualitative nature, the language is very expressive. For example, it can express without difficulty the basic signs in the American Sign Language (ASL) phonology [24], and the basic poses used in several current commercial systems. See figure 2 for some examples.

Based on the proposed language, a gesture based interface can be built using several possible tools. For a *C#* programmer, a pose class can be defined in a few lines of code, and a gesture class can be constructed once all of its constituent poses are defined. Above the code interface, additional layers are added to enable gesture definitions by non-programmers. We developed a simple text parser which enables definition of poses and gestures using XAML code. The parser then creates the appropriate *C#* classes. In addition, we have designed a visual gesture editor displaying the poses as states in a sequence. The editor enables pose manipulations using context sensitive menus, and its output can be exported into XAML format. For visualizing the edited poses in real time we developed a fast inverse kinematics algorithm, utilizing the language's simplicity. The algorithm produces a pose satisfying the chosen propositions, which is then rendered using a generic hand model. Finally, this system aids in identifying non-valid hand definitions.

The algorithmic pipeline which recognizes the language has to resolve the accuracy versus speed tension mentioned earlier. To ease this tension, we use Convolutional Table Ensemble (CTE) classifiers and regressors [14, 3]. These are extremely fast predictors, typically processing an image in less than a millisecond. As shown in [14], the CTE architecture enables trading training sample size for speed and accuracy, that is: by using larger sample size at the training stage, the run-time predictor can be made faster while keeping the same accuracy. The pipeline includes several stages, each employing a set of CTEs. In the first stage, the position of the hand center is found and the image is centered around it. Then the global hand orientation is found, framed as a classification problem into 16 discrete pose clusters, and is then refined. At a third stage the location and direction of fingertips are found, by applying a cluster-specific regressor. This regressor in turn includes several stages of regressing the fingertip location, centering the image around the tip and regressing again for refinement. Finally, the truth value of the basic 102 language propositions is inferred from the fingertips and palm center locations.

In order to obtain the speed and accuracy benefits from the CTE architecture, a large dataset is required for training. In our proposed system, however, this training is a one time event and no machine learning effort is required from the gesture developer. We collected more than $360,000$ annotated images for the pipeline training, using a custom-built dome-shaped structure equipped with multiple cameras. Since the target camera uses IR, colors which are IR-invisible were used to mark interest points on the hands of the subjects. Some annotation was then automatically collected using a set of surrounding RGB cameras, while another portion, like exact fingertip locations, required manual tagging.

We evaluated the accuracy of our algorithm on several levels: estimation of fingertip locations, recognition of the language propositions, and recognition of full gestures. Hand pose estimation is usually evaluated in the literature by considering statistics of the distance between fingertips position and their algorithmic estimates. We evaluated our algorithm using these metrics in two publicly available datasets, NYU [27] and Dexter [22]. Using NYU, which is the larger and more challenging dataset, our method is comparable to the best method, and using Dexter it is ranked third among nine methods. This accuracy is obtained in 14 millisecond per image on a single CPU thread - roughly an order of magnitude faster than any other method of similar accuracy. This degree of accuracy and speed enables a practical, real-time, gesture-based interface in a variety of scenarios. Moreover, unlike other leading methods, we recognize the hand pose without using temporal information (single frame), which makes the measured accuracy robust to fast pose changes and short gestures.

Beyond fingertip localization errors, for real gesture recognition using a language of the type we define here, the important statistics are the probabilities of correctly detecting basic propositions and full gestures. We estimated our capabilities for recognition of the basic propositions using a test set of $61,397$ images containing random hand poses. Our system is able to recognize basic propositions 92% of the time with a false positive rate lower than 1.4%. For full gestures, we tested our system using a set of 12 selected gestures, performed multiple times by 10 different persons, and an additional set of non-gesture hand activity clips for false alarm rate estimation. Our pipeline achieves an average detection rate of 96% for users after a few minutes of practice.

RELATED WORK

Our contribution is in language and development tools for hand gesture interface, as well as in the hand pose estimation

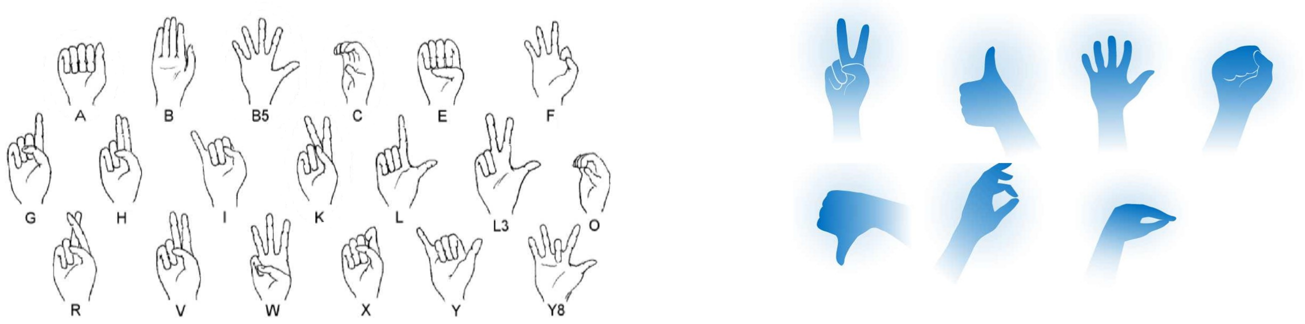


Figure 2. Pose dictionaries: (Left) The hand shapes found in a phonemic analysis of ASL. According to the analysis, these (together with location and motion elements) were found to be basic patterns characterizing the symbolic vocabulary of the language. Each of these hand poses can be well characterized, and distinguished from the others, using a few basic propositions utilizing our four base predicates: finger/palm direction, finger folding state, finger tangency and relative direction. (Right) The hand shapes used in the static gestures of a popular commercial system [1]. These can also be naturally described using our simple language.

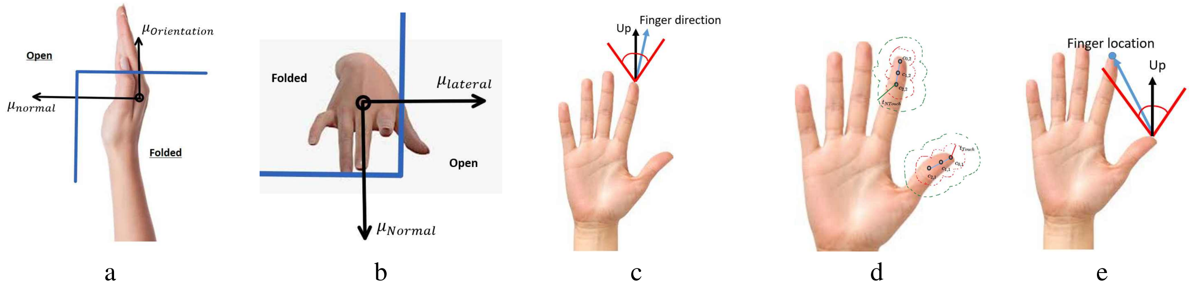


Figure 3. Predicate operational definitions: (a) A non-thumb finger is considered folded if its tip resides in the marked quarter, defined by planes whose normal is the palm direction and palm orientation. (b) The folding region of the thumb (c) A finger is pointing 'up' if its direction is in a cone with the base placed at the fingertip. An analogous definition is given for the other 5 directions. (d) Touching and not-touching relations are determined by the intersection of multiple balls around the distal phalanges of the two fingers (e) Relative directions are determined using a cone whose axis is aligned with canonical direction. Here the index is above the thumb because it falls in this cone for 'up' direction.

algorithm. Here we briefly refer to the relevant literature in these two domains.

Gesture language and development tools: In [19] the advantages of using a declarative gesture language compared to procedural gesture programming are elaborated, but in the context of full body and touch gestures. Three such advantages are mentioned: easier feedback providing during the gesture, avoiding code division among components (termed 'spaghetti code' there), and convenient treatment of compositionality and gestures with partial overlap. Among these, our method enjoys the two latter advantages, but does not currently address user feedback during a gesture - an action is performed only at the end of the poses sequence.

The Gesture Description Language (GDL), presented in [11], uses a context free grammar to define a language describing full body gestures. The primitive joints relate to large body entities, but not to hand parts or fingers, and the language is recognized using a Microsoft Kinect system followed by an inference engine. Unlike in our methodology, describing gestures requires the developer to state numerical parameters and relations like exact angles or distances between joints. Another context free grammar for full body gesture specification is presented in [10]. The goal of the suggested language is not to ease interface development, and no actual recognition engine for the language is suggested. Instead the focus is on privacy, security and reliability: verifying that the gestures suggested are valid, not empty, there are no collisions between registered gestures, etc. Most similar to our method in spirit, the 'Proton++' system [13] includes a language and a recognition algorithm for multi-touch gestures. Like in our language, they describe a gesture as a sequence of basic elements (finger strokes) characterized using a small set of propositions. A basic stroke is defined by its type (finger up/down/move), the finger index, and several stroke attributes (movement direction, pressure, etc.). Like in our system, a state machine handles the recognition and disambiguation of multiple gestures.

More related to hands-based interface, in [5] a stochastic context free grammar is suggested for the composition of single poses into gestures. This grammar assigns probabilities to gestures given an input sequence of detected primitive poses, and so it enables gesture detection in cases of non-accurate execution. However, the language does not include description of general poses - instead a set of only 4 primitive poses is used (fist, open hand, one or two fingers raised). In another work [7] hand gestures based on hand motion are described as compositions of 14 primitive motion elements known from the ASL phonemic decomposition [24]. This work is orthogonal to ours, as hand internal pose is not addressed at all -

the hand is assumed to be rigid. In [20] the dexterity and independence properties of the different fingers are examined, and an efficient language for text dictation via finger folding is devised.

Hand pose estimation algorithms: A lot of work [15, 16, 12, 27, 17, 26, 22, 21, 14, 25, 6, 2] has been done in recent years. See [9] for a review. We compare our results to many of these methods. There are several quite different approaches to the hand pose estimation problem. One line of work uses a 3*D* hand model and addresses the problem as model-based tracking [17, 2, 26]. These methods are usually accurate, but require high computational effort. Another direction uses ensembles of trees or ferns for pose estimation from a single frame [12, 14, 25, 18]. The advantages are the ability to run with a low computational budget and to comprehend fast motion. Specificly in [18] a method based on multi-stage random forest was used to successfully discriminate between 6 pre-defined poses on smartphones with a simple RGB camera. Our work belongs to this algorithmic family, and significantly improves its accuracy. Some work is based on combining modules from model-based tracking and single-frame fast detection, such as [22, 21]. Finally, several groups have used Convolutional Neural Networks (CNN) [27, 15, 16] with increasing degrees of sophistication and accuracy over time.

LANGUAGE

The language we propose is based on a set of qualitative basic propositions, such as "the index and middle are not touching". These propositions are close to gesture descriptions in natural language, and thereby enable easy pose characterization for the developer. To contrast, quantitative descriptions such as "keep the Index and Middle tips 3cm away from each other" include parameters that are hard for the developer to estimate as they often do not have good intuition for their values. Moreover, even when they do estimate such values well for their own hands, the result is not likely to generalize well to people with different hand size or different morphology/flexibility. We believe tuning these parameters is better left for the system.

Basic propositions

The propositions are obtained by applying two one-argument predicates (direction, flexion) and two two-argument predicates (relative direction, tangency) to one or two of six interest points on the hand: the fingertips and palm center. Direction and direction relationships are quantized to 6 canonical values: "Left (of the subject)","Right", "Up", "Down", "Forward" and "Backward". The other two predicates, flexion and tangency, are naturally binary. Here are the basic propositions in detail:

- **Palm pose**:
  - Palm direction: This is the direction of the normal of the palm, pointing out of the forehand. 6 propositions of the form "the palm points in direction X" are defined, for the 6 canonical directions.
  - Palm orientation: The direction pointing from the wrist to the base of the middle finger. Again 6 propositions are defined.

  The operational definition of '*pointing toward a canonical direction*' is that the pointing direction has a small angle with the canonical direction. However, the parameter, as any other parameter in the system, is not exposed to the developer. See figure 3.a for visualization of the palm direction (μ_{normal} in the figure) and orientation ($\mu_{Orientation}$) concepts.

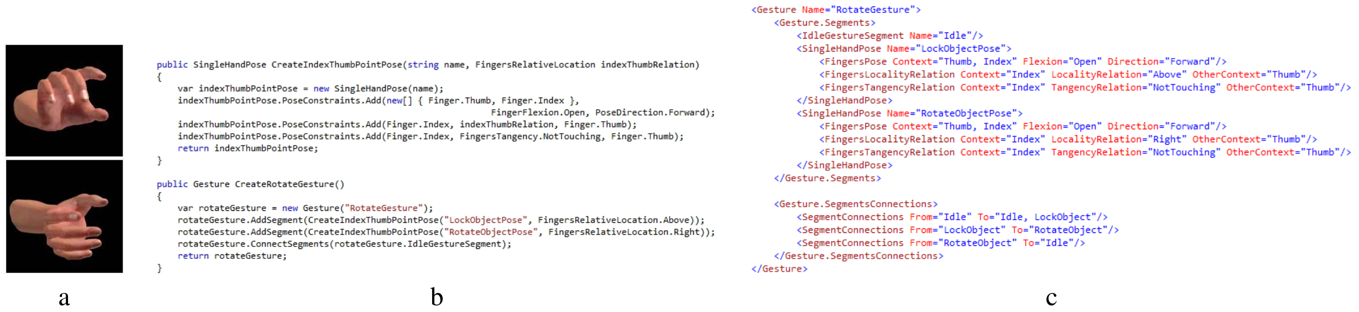
- **Fingers**:
  - Finger direction: For each finger we define the 6 propositions of pointing in the canonical directions, for a total of 30 propositions. (See figure 3.c) for the direction operational definition.
  - Finger flexion: For each finger two states are defined, as '*open*' and '*folded*', giving 10 propositions. A non-thumb finger is declared folded if its tip is in the quarter of the space near the hand, as defined by two planes (See figure 3.a). A similar characterization is given to the thumb. (See figure 3.b).
  - Finger tangency: For each of the 10 possible finger pair combinations, an '*a is touching b*' proposition is defined, as well as an '*a is not touching b*', for a total of 20 propositions. To evaluate these propositions, we define the distance between fingers as follows: *K* points, *d* millimeters apart from each other, are defined along the ray starting from the tip and pointing in the opposite of the finger direction. Two fingers are considered touching if the minimal distance between a point on one finger and the tip of the other is lower than a threshold (See figure 3.d), and 'not touching' when this minimal distance is higher than a different, greater threshold, such that '*not touching*' and '*touching*' are not a simple negation of each other: typically there are cases where neither of them applies.
  - Finger relative position: For each finger pair, a proposition is defined stating that '*a is in direction C from b*' where C is one of the 6 canonical directions. Since propositions like '*middle is above the thumb*' and '*thumb is below the middle*' are equivalent, this yields altogether 30 propositions. A proposition is satisfied if point a is in a cone whose base is at point b and its central axis is in the direction C (See figure 3.e).

Poses and gestures

Static hand poses are defined as conjunctions of propositions, where a proposition is either one of the basic 102 propositions, or a '*direction-disjunction*' over them. A *direction disjunction* is a statement such as '*the thumb points either left or up*', that is: the disjunction is over several basic propositions differing only in the direction stated. A static hand pose can be detected from a single frame. Gestures are defined simply as sequences of poses in time. To complete a gesture, the user has to go through the sequence of defined poses, with the time interval between the poses no longer than a threshold parameter.

DEVELOPMENT TOOLS

Based on the proposed language, we have created a set of development tools to enable easy hand gesture interface development. Gestures can be programmed in *C#* using a set of



```
public SingleHandPose CreateIndexThumbPointPose(string name, FingersRelativeLocation indexThumbRelation)
{
    var indexThumbPointPose = new SingleHandPose(name);
    indexThumbPointPose.PoseConstraints.Add(new[] { Finger.Thumb, Finger.Index },
                                            FingerFlexion.Open, PoseDirection.Forward);
    indexThumbPointPose.PoseConstraints.Add(Finger.Index, indexThumbRelation, Finger.Thumb);
    indexThumbPointPose.PoseConstraints.Add(Finger.Index, FingersTangency.NotTouching, Finger.Thumb);
    return indexThumbPointPose;
}

public Gesture CreateRotateGesture()
{
    var rotateGesture = new Gesture("RotateGesture");
    rotateGesture.AddSegment(CreateIndexThumbPointPose("LockObjectPose", FingersRelativeLocation.Above));
    rotateGesture.AddSegment(CreateIndexThumbPointPose("RotateObjectPose", FingersRelativeLocation.Right));
    rotateGesture.ConnectSegments(rotateGesture.IdleGestureSegment);
    return rotateGesture;
}
```

```
<Gesture Name="RotateGesture">
    <Gesture.Segments>
        <IdleGestureSegment Name="Idle"/>
        <SingleHandPose Name="LockObjectPose">
            <FingersPose Context="Thumb, Index" Flexion="Open" Direction="Forward"/>
            <FingersLocalityRelation Context="Index" LocalityRelation="Above" OtherContext="Thumb"/>
            <FingersTangencyRelation Context="Index" TangencyRelation="NotTouching" OtherContext="Thumb"/>
        </SingleHandPose>
        <SingleHandPose Name="RotateObjectPose">
            <FingersPose Context="Thumb, Index" Flexion="Open" Direction="Forward"/>
            <FingersLocalityRelation Context="Index" LocalityRelation="Right" OtherContext="Thumb"/>
            <FingersTangencyRelation Context="Index" TangencyRelation="NotTouching" OtherContext="Thumb"/>
        </SingleHandPose>
    </Gesture.Segments>

    <Gesture.SegmentsConnections>
        <SegmentConnections From="Idle" To="Idle, LockObject"/>
        <SegmentConnections From="LockObject" To="RotateObject"/>
        <SegmentConnections From="RotateObject" To="Idle"/>
    </Gesture.SegmentsConnections>
</Gesture>
```

Figure 4. Gesture description in formal language: (a) A 'Rotate Right' gesture is composed of the two shown poses, starting with the index above the thumb, and ending with the index right of the thumb (from the user's perspective). (b) *C#* code that generates a 'Rotate Right' gesture object. The two comprising poses are defined using a single method 'CreateIndexThumbPointPose()' accepting the required index-to-thumb relation as input argument. 'Rotate Right' is then defined as a short sequence of the two poses. (c) 'Rotate Right' gesture defined in XAML format. The same propositions are applied, but without the *C#* syntax.

classes, or text-edited by non-programmers. On top of these tools, we have devised a visual gesture builder tool, allowing gesture editing with a graphical user interface. The builder includes a visualization tool, which renders hand poses based on their definition in the language.

The runtime environment

Our suggested hand pose estimation algorithm (described in the next section) is an efficient routine running on the user's machine. In order to work with it, the developer builds and registers a *Gesture* object, which includes a gesture definition and a pointer to a callback function. Upon frame arrival, the runtime system computes the relevant basic propositions - a subset of the 102 propositions that is relevant for currently registered gestures. The execution of each registered gesture is tracked using a simple finite-state machine, monitoring which of the poses were already executed and what pose is expected next. Upon execution of the last pose, the callback function registered with the gesture is called. The direct interface for gesture building is programming in *C#*, and linking to the runtime library.

A *C#* interface

Writing a new pose class is done by inheriting the new class from an abstract *SingleHandPose* class, and adding the actual propositions content of the pose using predefined enum types. A gesture class is then defined by concatenating a predefined set of pose classes into a sequence. An example of a '*Rotate Right*' gesture written in *C#* is shown in figure 4, including two poses. Since the two poses are similar, they are defined in a single method, accepting as parameter the direction relation required between the thumb and the index. The gesture is defined in a few lines of code, and the lines describing the poses are fairly intuitive, reflecting the tight relation to natural language descriptions.

XAML text interface

The programming interface is the most straightforward, but it requires programming skills in *C#* and it mixes the programming work with gesture design, while the two tasks require different expertise and are usually performed by different people. Hence we consider an option to write gesture definitions in an XAML format, which is independent of a specific programming language. '*Rotate Right*' written in an XAML format is shown in figure 4.c. These files are easier to write, and provide a convenient middle layer for the next design level: a visual gesture builder.

A Visual Gesture Builder

A more natural way for building gestures is using a visual editor, providing immediate visual feedback. We have developed such a tool for gesture design in the suggested language. The gesture is presented as a visual sequence of poses (see figure 5). Once a pose is selected, the developer can choose one of the six areas of interest - the palm or one of the fingers - and edit it using a context menu. This menu allows choosing the item's direction and its flexion state (for fingers). For fingers there are four additional items in the menu, allowing one to choose a second finger and establishing the required relation between the two fingers. These relations are chosen from a second-level context menu, and allow specifying fingertip touching and directional relations between the chosen fingers.

When a menu item choice changes, the gesture builder calls a quick inverse kinematics algorithm to find a pose (a vector of joint angle values) which meets the new set of constraints, and renders it instead of the previous pose. When conflicting constraints exist, the inverse kinematics fails to find a valid pose satisfying the constraints, and a warning message is issued. For example, this happens if the thumb and index are instructed to point forward, but the ring is instructed to point left (see figure 5.d(bottom)). The developer may rotate the camera view of the hand at any time by dragging the mouse inside the pose circle, to better understand the finger locations in complex, partially occluded poses. Once editing is done, the gesture is saved as a XAML file, from which classes representing the new gesture are automatically generated. This tool enables trial and error experimentation in the space of pose definitions, and can significantly accelerate gesture development.

The inverse kinematics algorithm mentioned above has to solve a hard satisfaction problem including non-convex constraints, and do it immediately to enable real time feedback. We use a coarse-to-fine greedy approach, starting from a base-

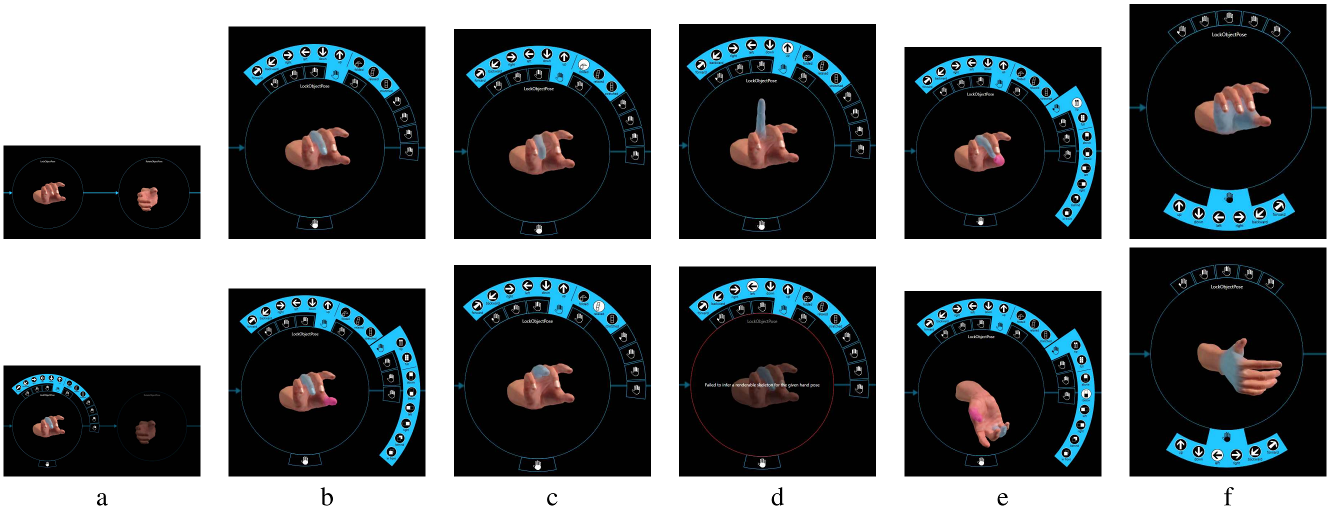

LockObjectPose
Failed to infer a renderable skeleton for the given hand pose
a
b
c
d
e
f

Figure 5. Visual gesture builder: (a) The builder presents the poses of the gesture as a sequence (top) and allows the developer to choose a pose to edit using a mouse click (bottom). (b) A selected finger is marked in blue, and opens a context menu (top), from which a second menu enables choosing a second finger, marked in red (bottom). (c) A selected finger can be constrained to be in folded (top) or open (bottom) state. (d) Finger direction can be chosen, up in the example (top), but if it cannot be reconciled with current constraints an alert message is displayed instead (bottom). (e) When two fingers are marked, relations of tip touching (top) and relative locations (bottom) can be stated. (f) the palm can be marked, then rotated to a different direction.

Algorithm 1 Ferns Ensemble computation

```
Input: An image I of size $S_x \times S_y$,
Classifier parameters $\{\Theta^m, A^m, W_c^m, \beta_c\}_{m=1,c=1}^{M,C}$
$\Theta^m = \{\delta_k^{1,m}, \delta_k^{2,m}, t_k^m\}_{k=1}^{K}$, $A^m \subset \{1,..,S_x\} \times \{1,..,S_y\}$, $W_c^m \in \mathbb{R}^{2^K}$, $\beta_c \in \mathbb{R}^C$
Output: Class scores vector $Score$
Initialization: For $c = 1,.,C$ $Score[c] = -\beta_c$
For all ferns $m = 1,..,M$
    For all pixels $p \in A^m$
        Compute a k-bit index $b_p^m$
        For $c = 1,.,C$ $Score[c] = Score[c] + W_c^m[b_p^m]$
Return $Score$
```

line pose and enforcing constraint families one at a time. First we look for a pose satisfying the palm direction and relative finger direction constraints, as these two constraint types put strong limitations on the global hand orientation. Then finger flexion, finger direction and finger tangency constraints are enforced, in that order. In most cases, this process successfully finds a pose satisfying all the constraints if there is one.

HAND POSE ESTIMATION ALGORITHM

Our practical hand pose estimation algorithm is based on two high level ideas: multiple stage problem breakdown, and prediction based on clever memorization. The pose estimation task is broken into several stages, each with a small scope problem to solve. At each stage we use a set of very fast predictors (the CTE family) whose activation essentially amounts to indexing a set of tables. These tables, created during training, memorize hand pose information and enable fast answer prediction by gathering the votes across an ensemble.

Convolutional Table Ensembles (CTE)

A CTE predictor extracts codeword indices from multiple positions in the image, and uses them to index multiple tables. The tables' votes are combined linearly to predict the output of interest. At [14] these classifiers were introduced in which the codeword indices are extracted using a set of independent questions, in which case the index computing structure is called a '*fern*', and the classifier is termed a Discriminative Ferns Ensemble (DFE). We also use some of the improvements suggested in [3], except that for codeword computation we use ferns rather than trees, as ferns are faster.

The ferns ensemble predictor operates on an image patch I, consisting of P pixels. For a single fern and pixel location $p \in \mathbb{N}^2$, a local descriptor for p's neighborhood is computed using a set of $k = 1,..K$ binary questions of the form

$$b_k = \sigma(I[p + \delta_k^1] - I[p + \delta_k^2] - t) \quad (1)$$

where $\delta_k^1, \delta_k^2 \in \{-s,...,s\}^2$ are location offsets, $t \in \mathbb{R}$ is a threshold, and $\sigma(\cdot)$ is the Heaviside function. These are simple and computationally-light questions, comparing the difference between two pixels to a threshold. For fern m, location p and question k denote the bit obtained by $b_{p,k}^m$. Concatenating the K bits together, a K-bit codeword b_p^m is obtained at for every fern and pixel location.

Histogram of Bit Vectors: In order to obtain translation invariance a spatial histogram of codewords over pixel locations is computed. Denote the histogram for the m^{th} fern by $H^m(I)$. An entry $b \in \{0,1\}^K$ of H^m is defined by

$$H_b^m = \sum_{p \in A^m} \delta(b_p^m - b) \quad (2)$$

where δ is a discrete delta function, and $A^m \subset \{1,..,P\}$ is the spatial aggregation region for fern m. Note that H^m is a sparse vector, with at most P non-zero entries.

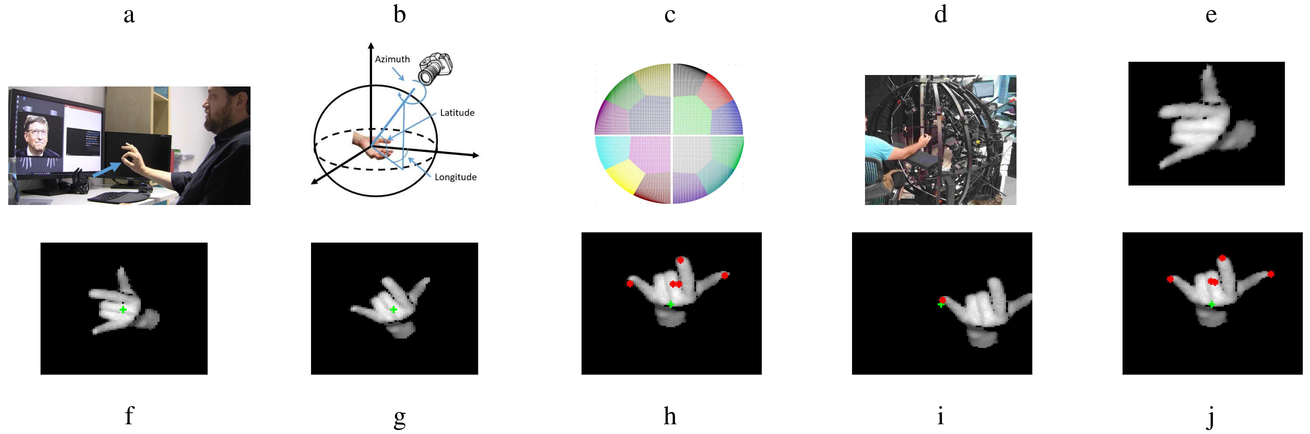


Figure 6. Hand estimation pipeline: (a) in the user setting we address the hand is typically the dynamic object closest to the camera in the direction shown. (b) Latitude, longitude and azimuth in a hand-centered coordinate system. (c) The clusters we use on the (latitude, longitude) sphere, viewed from the right axis in b. (d) The iron dome used for data gathering. (e) A segmented hand patch. (f) The hand patch after centering. Estimated hand center marked in green. (g) The hand after coarse (latitude, longitude) finding and azimuth correction. (h) The hand after global hand pose refinement and initial fingertip location regression. (i) A finger-centered image sent for finger refinement. (j) The final fingertip locations estimation.

Histograms concatenation: The final output is computed by a linear classifier or regressor applied to the concatenation of the M fern histograms.

$$f(I) = W^T H(I) = \sum_{m=1}^{M} \sum_{b \in \{0,1\}^K} w_b^m H_b^m - \beta \qquad (3)$$

with $H(I) = [H^1(I), \ldots, H^M(I)] \in \mathbb{N}^{M2^K}$, a weight vector $W = [W^1, \ldots, W^M] \in \mathbb{R}^{M2^K}$, and a bias term $\beta \in \mathbb{R}$. When multiple outputs are required (as in multiclass classification), C weight vectors and biases $\{W_c, \beta_c\}_{c=1}^{C}$ are trained and used to obtained C output scores using the same set of ferns.

Run time classifier/regressor: Algorithm 1 describes the operation of a DFE at test time. The pipeline is extremely simple. For each fern and each pixel in the fern's aggregation region we compute the codeword index, and access the tables of all classes with this index to get the codeword's score contribution. The complexity is $\mathrm{O}\big(M\overline{A}(K+C)\big)$ where $\overline{A}$ is the average number of pixels per aggregation region.

CTE training: As described in [14, 3], a CTE is trained by iterating between adding a fern and training a global predictor over the current set of ferns' features. The global optimization is done with SVM when classification is handled, and an SVR loss when regression is required. Adding a fern, which adds 2^K features to the predictor, requires optimization over the bit function parameters Θ^m, and the aggregation area A^m. These are chosen greedily so that the added features align well with the gradient of the current predictor. Following [3], beyond bit functions comparing two pixels, we use also bit functions comparing one pixel to a threshold, as well as spatial bits providing information regarding the pixel location.

Hand Pose Estimation Pipeline

The algorithmic pipeline consists of multiple stages: hand segmentation, hand centering, global orientation classification, global pose refinement, fingers pose estimation, and finally, estimation of basic propositions truth values. The main stages (hand pose and finger pose estimation) rely on CTE predictors.

Hand segmentation: Hand finding is based on two assumptions: hand movement and hand proximity to the camera. We start by dropping pixels far from the camera and static pixels, using a fast background subtraction method. Among the remaining pixels, the pixel with lowest projection on the 3D direction $(0,-1,2)$ is chosen as the hand's '*anchor pixel*', reflecting the relative position of the camera and user in our setting (see figure 6.a). The hand is then segmented by taking the set of pixels whose $3D$ distance from the anchor pixel is smaller than a threshold (See figure 6.e).

Hand centering: The hand center of mass in $3D$ is found and a $3D$ transformation is computed so as to rotate it about the shortest rotation direction onto the camera principal axis. This transformation, together with affine scaling, is then applied to all the hand pixels, and the hand is re-rendered. This process maps all hands to frame center and roughly equal size, thus reducing variation due to perspective projection and distance from camera (see figure 6.f).

Global hand orientation classification: We term the following 6 parameters by 'global hand pose': the $3D$ palm center location and $3D$ hand orientation. Hand orientation is a main source of variance in hand pose estimation: hands seen from different viewpoints have very different appearances of the palm, the fingers and their mutual occlusion patterns. Therefore we consider the coarse global orientation determination as a classification problem, and the pipeline in the next stages is split to different paths according to the classification decision. Instead of thinking about the hand as rotating, we fix it in a canonical pose at $(0,0,0)$, and consider the possible camera positions and rotations on the viewing sphere (See figure 6.b). The camera longitude and latitude determine its position on the unit sphere, and the azimuth is related to the camera rotation around its principal axis (in-plane rotation). Viewpoints in which the hand is seen from the arm direction (first person) are not considered, since they rarely occur in our scenario.

We treat longitude/latitude and azimuth differently in our classification. The viewing half-sphere is divided into 16 (latitude, longitude) clusters (See 6.c), and the azimuth is independently divided to 8 equidistant clusters centered at rotations of $0^\circ, 45^\circ, ..315^\circ$. During training we use the ground truth hand rotation to assign an image to one of the $16 \times 8 = 128$ possible labels. A single CTE classifier is learned with 16 classes, where output i is trained to discriminate images with azimuth cluster 0 and (latitude, longitude) cluster i from all other images. At test time, the image is rotated 8 times in 45° intervals, and submitted to the classifier in each rotation. Since the classifier was trained to discriminate images from a single rotation, 7 of the rotated images are expected to get low scores in all their 16 outputs, and only the correct rotation gets a high i-th output. The highest scoring class among the 128 outputs determines the (latitude, longitude) orientation cluster, and the azimuth cluster. The hand image is then rotated so as to cancel out the predicted azimuth. (see figure 6.g).

Global hand pose refinement: This stage refines the coarse hand orientation and location (known from orientation classification and hand centering stages respectively). The refinement is done in two regression stages, each with 6 outputs. At each stage, the hand image is re-centered and re-rotated using the current estimates. Then the residual difference between the current estimates and true (center, orientation) values is regressed. During training, two such consecutive stages are trained for each (longitude, latitude) cluster, for a total of $16 \times 2 = 32$ CTE-regressors. However, when testing only 2 regressors corresponding to the chosen cluster are activated.

Fingers regression: This part of the pipeline includes 3 regression stages (see figure 6.h-j) trained separately for each (longitude, latitude) cluster. The first stage operates on the rotated and centered hand image and regresses the rough location of the 5 fingertips. Following that, two finger refinement stages take place for each finger separately. At each stage, the image is translated to have the finger of interest centered according to the current estimation, and the residual translation of the real finger is regressed. At the second stage, the position of the distal finger joint is also regressed, in order to get the finger direction by subtracting it from the fingertip. Overall there are $1 + 2 \times 5 = 11$ regressors activated at this stage, and $11 \times 16 = 176$ regressors are trained for all clusters.

Basic propositions truth value: As described earlier, each of the 102 propositions has an operational definition in terms of global palm direction, fingertip locations or fingertip directions. Given the estimations of the latter, the truth value of the relevant basic propositions can be readily estimated.

Overall, the estimation system contains 209 CTE predictors, but only 21 CTE activations are performed at test time per frame. Such a multi-classifier approach is possible due to the very low computational cost of CTE predictors. For example, a classifier with $m = 20$ ferns, $C = 10$ class, $K = 12$ bits and an aggregation area of 64×64 pixels runs at $550\mu S$ on a single thread of an $i7 - 3720QM$ CPU@$2.6GHz$ processor.

Data and annotation gathering

In the CTE framework, gathering a large annotated dataset is the key for test time speed, since a larger data set allows usage of larger tables (larger K) and therefore fewer ferns (lower M) - see [14] for the details. Our data was gathered using Intel's RealSense SR300 camera [1], providing 640×480 depth and IR images using coded light technology. In order to get a large dataset, we constructed an iron dome, with up to 23 affixed Intel cameras and 8 high definition RGB cameras, all pointing toward the dome center (See figure 6.d). All of the cameras were jointly calibrated and synchronized (synchronization is required due to the active nature of the cameras, which may cause interference). Using this construction, a hand pose in the center of the dome provides 23 depth images, and annotation obtained can be readily propagated between them.

The RGB cameras are used to obtain the annotation needed: global hand pose, as well as location of fingertips and distal finger joints. We marked the positions of key joints on each subject's right hand using colors not seen by the IR cameras. Specifically, 3 points and one short line were marked on the back of the hand, for determination of the global hand pose. When the hand is placed at the dome's center, each such point is seen by at least 2 RGB cameras, so its $2D$ image position was automatically detected, and its $3D$ point location was found by triangulation. In this manner, the global pose annotation is found automatically. For fingertips and other joint locations we could not achieve this automatic annotation due to marker confusion and occlusion problems, and so we resorted to manual annotation.

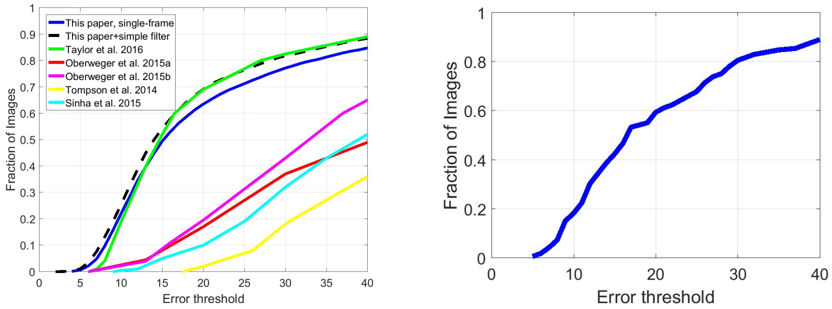
Altogether $89,333$ images were collected and automatically tagged for global hand pose training. To this we added virtual samples, created from the original samples using in-plane rotation. For fingertips detection $274,068$ images were manually annotated. This large a sample is required since 16 different fingertip detection pipelines are trained, one per (longitude, latitude) cluster, using mutually exclusive sub-samples.

In addition to the training data, two datasets were gathered for evaluation purposes. The first includes $61,397$ fully annotated images of random hand poses, used for estimation of fingertip location accuracy and basic proposition estimation. A second dataset includes 507 clips of 12 gestures, performed multiple times by 10 different subjects. A $3,500$ random subset of the first dataset, and the entire second dataset are made publicly available[1]. Prior to recording, the subjects were allowed to train on the 12 gestures for three minutes, to simulate the steady state of experienced users. Clips were annotated with tags marking the temporal intervals in which poses of interest were maintained. In addition, 17 minutes of intensive non-gesture hand activities were recorded. This dataset is used for estimation of gesture-detection statistics: detection and false alarm rates. See the appendix for a more detailed description.

EMPIRICAL RESULTS

In most of the hand pose estimation literature [27, 15, 16, 17, 26, 25], system performance is measured using statistics of $3D$ deviations between true finger locations and their estimates.

[1] https://aka.ms/atli-hands-chi2017



Method	Error (mm)
Taylor et al. 2016 [26]	9.5
Tagliasacchi et al. 2015 [2]	9.8
This paper, simple filter	12.7
This paper, single frame	14.8
Sharp et al. 2015 [17]	15.0
Sridhar et al. 2015 [21]	19.6
Sridhar et. al 2014 [23]	24.1
Choi et al. 2015 [6]	25.3
Sridhar et al. 2013 [22]	31.8
Tang et al. 2014 [25]	42.4

Figure 7. Joint deviation results: (Left) Max deviation CDF results on NYU dataset. See text for explanation. (Center) Max deviation CDF on a random subset of our test data, consisting of $3,500$ images. (Right) Average error rates on Dexter data for our method and alternatives. Our accuracy is obtained in 14 millisecond per image on a single CPU thread - roughly an order of magnitude faster than any other method of comparable accuracy. Our pose recognition is based on single-frame. When adding even a simple temporal median filter on adjacent frames, the accuracy increases (see Method "This paper, simple filter"), demonstrating further potential for improvements.

We report our results using these measures on our test data, and compare to other methods on two publicly available datasets. However, for practical gesture recognition performance beyond deviation statistics should be measured. We measure the accuracy of our system on two higher level tasks: estimating the truth value of our 102 basic propositions, and detecting actual gestures phrased in our language.

3D finger deviations - comparison to state of the art

We experimented with two publicly available datasets: NYU [27] and Dexter [22]. NYU is a large dataset with $N = 8252$ test images including challenging poses. Dexter is a smaller ($N = 2931$) and easier dataset, where the hand is frontal in most of the poses and all fingertips are usually visible. We use the methodology of [26] to map the output joints of our method to ground truth joints of the specific dataset. In both these datasets, the hand movement is often relatively slow compared to natural gestures and the importance of temporal information is emphasized. On the contrary our method, which is a single-frame algorithm designed to cope with fast gestures, does not use temporal information at all. To make the comparison fair, we show our results both for the pure algorithm, and for the algorithm after applying a simple temporal median filter independently for each joint location, with a window of 5 frames.

In figure 7.Left we compare our NYU results to several alternatives. The graph shows a CDF of maximal deviation in a frame: for each joint the deviation in millimeters between estimated and true location is computed, and the graph shows the fraction of frames in which the maximal deviation is smaller than a threshold. The table in figure 7.Right lists the average fingertip estimation error in millimeters on Dexter dataset for our algorithm and alternative methods. Figure 7.Center shows the maximal deviation CDF on a subset of our own test set. This set contains diverse, random hand poses taken from the covered half of the viewing sphere.

The comparison to alternatives shows that our method is comparable to the state-of-the-art method [26] on NYU, and slightly lower on Dexter. The higher relative ranking on NYU indicates that our strength is mostly with hard poses, where finger occlusion is prevalent. When comparing to [26] several points should be noted: First, our method runs in $11.5ms$ on a single thread (laptop with i7-4810MQ @2.8GHz), while the method of [26] is reported to consume most of the CPU on an 8-core, 16-thread machine. Our method is therefore at least one order of magnitude faster. This is also true regarding the methods accurate on Dexter [17, 2], which run on GPU. Second, unlike these alternatives our method inferences using a single frame, and is therefore more robust for fast and abrupt hand motions which are prevalent in natural gesturing.

Basic proposition accuracy

We tested the algorithm in the task of estimating the truth value of the 102 basic language propositions, using our test set of $61,397$ images. For each image the basic proposition's truth value was computed using the algorithm estimations, and compared to the truth value computed using the ground truth joint locations. Since we quantize continuous hand orientation angles into discrete directions, we allowed margin of +/-15 degrees between positive and negative zones, and ignore borderline cases with such low margins. In addition, for finger posture propositions, we excluded cases where the palm direction is away from the camera. For example, if a pose is defined with the palm backward, it does not make sense to add a fingertip touching condition, as fingers are occluded. This does not factor out all types of occlusions, as one finger may occlude other fingers. However, our hand pose recognizer can deal with most of these types of occlusion. Detection and false alarm rates for families of basic propositions are reported in figure 8.Left. Among the basic proposition types, finger tangency and finger relative location are the hardest to detect, since successful detection requires accurate estimation of two fingertip locations, both of which may be occluded.

Gesture recognition accuracy

Detection rates on the new gestures dataset are shown in figure 8.(Right). Most gestures are detected well, with detection rates above 90%, with the 'Swipe down' gesture as an exception with a detection rate of 78%. We found that one of the main reasons for failures is that subjects do not perform a gesture as intended, even after it is shown to them. This highlights the importance of a real time feedback mechanism

Propositions	Detection	False Alarm
Palm direction	0.99	0.001
Finger direction	0.96	0.006
Finger flexion	0.91	0.043
Finger tangency	0.86	0.019
Finger non-tangency	0.98	0.005
Finger relative location	0.87	0.018
Overall	0.92	0.014

Gesture	Detection Rate	Gesture	Detection Rate
Hang Up	1.00	Explode	1.00
Swipe Up	1.00	Mute	0.97
Like	1.00	Shoot	0.94
Lock	1.00	Flute	0.92
Bloom	1.00	Tap	0.91
Rotate Right	1.00	Swipe Down	0.78

Figure 8. (Left) Detection and false alarm rate for basic propositions. (Right) Detection rates for 12 gestures using our new data set. The average detection rate is 96%.

as part of the user experience. More generally, the system needs to understand the users' intent, not just what they do.

The false alarm rate in real-usage scenario is very low, since in realistic scenarios the users keep their hands down most of the time. In addition, most of the gestures are registered only in a specific context (for example, when a certain window is in focus), so they are active only for a fraction of the activity time. Modeling the realistic hand activity distribution is hence very inefficient as it would require gathering many hours of mostly irrelevant data. Instead, our data includes 17 minutes of intensive non-gesture hand activity, on which the false alarm rate is 1.21 (false alarms)/minute.

DISCUSSION

Our method can be measured using several different metrics:

- Can a non-expert define poses and gestures? How long does it take?
- How expressive is the proposed language?
- What is the gesture recognition accuracy?

Regarding the first question above, we have shared the gesture buidling tools with 5 software engineers and designers, with no experience in computer vision. They were all able to learn from sample code, and define poses and gestures on their own in a few minutes. Nevetheless, a more comprehensive study is required in order to estimate finer distinctions, like the preference between visual and text-based gesture definition. Regarding language expressivity, we believe an evidence for the strength of the coverage is that ASL poses are covered almost entirely, as well as standard commercial gestures.

The issue of gesture recognition accuracy is complex to estimate, and may be seperated into two levels: the accuracy at recognizing poses and gestures performed according to the formal definitions, and the accuracy at understanding user intent. As can be seen in figures 7,8.(Left), the algorithmic pipe has state of the art accuracy in finger location, and high recognition rates of the language's basic propositions. Despite being multi-staged, the system is very robust, mainly because solving for the palm global parameters, which is done in the first stages, is much easier than fingertip location done later. Specifically, the accuracy of the first stages, including palm orientation detection is very high, with 98.6 of the cases ending with very low deviations.

Understanding and interpreting correctly the user intent is more involved. For 11 of 12 defined gestures we achieve high detection rate ($>$ 90%, see Table 8). The main cause of false negatives is that some subjects performed the gesture not as the developer intended. Bridging the gap between ideal definition of a gesture and the way different subjects actually perform it, is challenging and requires expertise. Our approach conceals this complexity from the developer. For this purpose, we collected gestures from multiple subjects and tuned tolerance parameters. Still, more work is needed to improve it.

FUTURE WORK

Our language enables a rich scope of hand gestures based on varying the hand pose, but currently it lacks motion elements. Adding such elements to the language as additional basic propositions is an important route to follow. The development tools we presented here can also be improved, e.g. the current XAML representation can gain from a mechanism to avoid repeating lines of code when describing two similar poses that have a minor difference. The gesture builder lacks a clear presentation of the currently chosen basic propositions, as choices are often hidden in non-visible context menus.

In terms of interface usefulness, our work shows that general detection of certain hand poses is still challenging even with state-of-the-art hand pose estimation accuracy. Specifically, propositions of fingertip relations may benefit from additional accuracy improvements. With the current accuracy some care has to be taken to choose sets of gestures which are different enough from each other, and gesture registration is better limited to predefined contexts to avoid excessive false alarms.

There are several clear avenues for increased pose estimation accuracy in our system. One direction is the incorporation of temporal information and/or a generative 3D model, as in [26, 2]. This reasoning is highly complementary to the single-frame discriminative reasoning currently used in our system. Another element with a significant potential for improvement is adding a fingertip detection module to complement the currently used regression stages. This can improve accuracy for the cases where the fingertips are visible.

CONCLUSIONS

We presented a simple language for the design of a hand-gesture-based user interface, and a set of tools enabling rapid development. The algorithmic pipeline, based on fast CTE classifiers, is able to combine high speed with state of the art accuracy, and enables recognition of general gestures expressed in the proposed language. We believe the system represents an important step forward in the development and employment of general and practical gesture-based interfaces, accessible to any developer.

REFERENCES

1. 2016. Intel RealSense SDK Design Guidelines. https://software.intel.com/en-us/articles/intel-realsense-sdk-ux-design-guidelines. (2016).

2. Tagliasacchi A., Schroder M., Tkach A., Bouaziz S., Botsch M., and Pauly M. 2015. Robust articulated-ICP for real-time hand tracking. In *Computer Graphics Forum*.

3. Aharon Bar-Hillel, Eyal Krupka, and Noam Bloom. 2016. Convolutional Tables Ensemble: classification in microseconds. *CoRR* abs/1602.04489 (2016). http://arxiv.org/abs/1602.04489

4. Henry Chen, Austin S. Lee, Mark Swift, and John C. Tang. 2015. 3D Collaboration Method over HoloLens™and Skype™End Points. In *Proceedings of the 3rd International Workshop on Immersive Media Experiences*. 27–30. http://doi.acm.org/10.1145/2814347.2814350

5. Qing Chen, Nicolas D. Georganas, and Emil M. Petriu. 2007. Real-time Vision-based Hand Gesture Recognition Using Haar-like Features. In *IMTC*.

6. Chiho Choi, Ayan Sinha, Joon Hee Choi, Sujin Jang, and Karthik Ramani. 2015. A Collaborative Filtering Approach to Real-Time Hand Pose Estimation. In *ICCV*.

7. Konstantinos G. Derpanis, Richard P. Wildes, and John K. Tsotsos. 2004. Hand Gesture Recognition within a Linguistics-Based Framework. In *ECCV*.

8. Parth Rajesh Desai, Pooja Nikhil Desai, Komal Deepak Ajmera, and Khushbu Mehta. 2014. A Review Paper on Oculus Rift-A Virtual Reality Headset. *International Journal of Engineering Trends and Technology (IJETT)* 13 (2014). https://arxiv.org/ftp/arxiv/papers/1408/1408.1173.pdf

9. Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, and Xander Twombly. 2007. Vision-based hand pose estimation: A review. *Comput. Vis. Image Underst.* 108, 1-2 (2007), 52–73.

10. Lucas Silva Figueiredo, Benjamin Livshits, David Molnar, and Margus Veanes. 2016. Prepose: Privacy, Security, and Reliability for Gesture-Based Programming. In *IEEE Symposium on Security and Privacy*.

11. Tomasz Hachaj and Marek R. Ogiela. 2014. Rule-based approach to recognizing human body poses and gestures in real time. *Multimedia systems* 20 (2014), 81–89.

12. Cem Keskin, Furkan Kirac, Yunus Emre Kara, and Lale Akarun. 2012. Hand pose estimation and hand shape classification using multi-layered randomized decision forests. In *ECCV*. 852–863.

13. Kenrick Kin, Bjarn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012. Proton++: a customizable declarative multitouch framework. In *The 25th annual ACM symposium on User interface software and technology (UIST)*. 477–486.

14. Eyal Krupka, Alon Vinnikov, Ben Klein, Aharon Bar Hillel, Daniel Freedman, and Simon Stachniak. 2014. Discriminative Ferns Ensemble for Hand Pose Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

15. Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. 2015a. Hands Deep in Deep Learning for Hand Pose Estimation. In *Proc. Computer Vision Winter Workshop (CVWW)*.

16. Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. 2015b. Training a Feedback Loop for Hand Pose Estimation. In *Proceedings of the International Conference on Computer Vision*.

17. Toby Sharp, Cem Keskin, Duncan Robertson, Jonathan Taylor, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, Alon Vinnikov, Yichen Wei, Daniel Freedman, Pushmeet Kohli, Eyal Krupka, Andrew Fitzgibbon, and Shahram Izadi. 2015. Accurate, Robust, and Flexible Real-time Hand Tracking. In *CHI*.

18. Jie Song, Gabor Soros, Fabrizio Pece, Sean Ryan Fanello, Shahram Izadi, Cem Keskin, and Otmar Hilliges. 2014. In-air Gestures Around Unmodified Mobile Devices. In *The ACM Symposium on User Interface Software and Technology*.

19. Lucio Davide Spano, Antonio Cisternino, Fabio Paterna, and Gianni Fenu. 2013. GestIT: a declarative and compositional framework for multiplatform gesture definition. In *The ACM SIGCHI symposium on Engineering interactive computing systems (EICS '13)*. 187–196.

20. Srinath Sridhar, Anna Maria Feit, Christian Theobalt, and Antti Oulasvirta. 2015a. Investigating the Dexterity of Multi-Finger Input for Mid-Air Text Entry. In *The ACM Conference on Human Factors in Computing Systems*. 3643–3652.

21. Srinath Sridhar, Franziska Mueller, Antti Oulasvirta, and Christian Theobalt. 2015b. Fast and Robust Hand Tracking Using Detection-Guided Optimization. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*. http://handtracker.mpi-inf.mpg.de/projects/FastHandTracker/

22. Srinath Sridhar, Antti Oulasvirta, and Christian Theobalt. 2013. Interactive Markerless Articulated Hand Motion Tracking Using RGB and Depth Data. In *ICCV*.

23. Srinath Sridhar, Helge Rhodin, Hans-Peter Seidel, Antti Oulasvirta, and Christian Theobalt. 2014. Real-time Hand Tracking Using a Sum of Anisotropic Gaussians Model. In *Proceedings of the International Conference on 3D Vision (3DV)*. 8. http://handtracker.mpi-inf.mpg.de/projects/ellipsoidtracker_3dv2014/

24. W.C. Stokoe, D. Casterline, and C. C. Croneberg. 1965. *A Dictionary of American Sign Language*. Linstok Press, Washington, DC.

25. D. Tang, H. J. Chang, A. Tejani, and T.-K. Kim. 2014. Latent regression forest: Structured estimation of 3d articulated hand posture. In *CVPR*. 3786âĂŞ3793.

26. Jonathan Taylor, Lucas Bordeaux, Thomas Cashman, Bob Corish, Cem Keskin, Toby Sharp, Eduardo Soto, David Sweeney, Julien Valentin, Benjamin Luff, Arran Topalian, Erroll Wood, Sameh Khamis, Pushmeet Kohli, Shahram Izadi, Richard Banks, Andrew Fitzgibbon, and Jamie Shotton. 2016. Efficient and Precise Interactive Hand Tracking Through Joint, Continuous Optimization of Pose and Correspondences. *ACM Trans. Graph.* 35, 4 (July 2016), 143:1–143:12.

27. Jonathan Tompson, Murphy Stein, Yann Lecun, and Ken Perlin. 2014. Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks. *ACM Trans. Graph.* 33, 5 (2014).