Real Time Hand Pose Estimation using Depth Sensors

Cem Keskin, Furkan Kıraç, Yunus Emre Kara and Lale Akarun
Boğaziçi University
Computer Engineering Department, 34342, Istanbul, Turkey
keskinc@cmpe.boun.edu.tr, {kiracmus, yunus.kara, akarun}@boun.edu.tr

Abstract

This paper describes a depth image based real–time skeleton fitting algorithm for the hand, using an object recognition by parts approach, and the use of this hand modeler in an American Sign Language (ASL) digit recognition application. In particular, we created a realistic 3D hand model that represents the hand with 21 different parts. Random decision forests (RDF) are trained on synthetic depth images generated by animating the hand model, which are then used to perform per pixel classification and assign each pixel to a hand part. The classification results are fed into a local mode finding algorithm to estimate the joint locations for the hand skeleton. The system can process depth images retrieved from Kinect in real–time at 30 fps. As an application of the system, we also describe a support vector machine (SVM) based recognition module for the ten digits of ASL based on our method, which attains a recognition rate of 99.9% on live depth images in real–time [1].

1. Introduction

After the release of multi–touch enabled smart phones and operating systems, there has been a renewed interest in natural interfaces and particularly in hand gestures. Hand gestures are used in these systems to interact with programs such as games, browsers, e–mail readers and a diverse set of tools. Vision based hand gesture recognition, and particularly sign language recognition have attracted the interest of researchers for more than 20 years. Yet, a framework that robustly detects the naked hand and recognizes hand poses and gestures from colored images has continued to be elusive. This can be attributed mostly to the large variance of the retrieved images caused by changing light conditions, and to the difficulty of distinguishing the hand from other body parts. With the release of Kinect [1], a depth sensor that can work in absolute darkness, the hand detection and segmentation processes are considerably simplified. Thus, libraries for basic hand gesture recognition tasks have been developed. However, these only consider hand movement, and not hand pose. The estimation of the hand skeleton configuration has largely remained unsolved.

Recently, Kinect has been used to achieve real–time body tracking capabilities, which has triggered a new era of natural interface based applications. In their revolutionary work, Shotton *et al.* fit a skeleton to the human body using their object recognition based approach [14]. The idea is applicable to the hand pose estimation problem as well, but there are some notable differences between the human body and hand: i) The projected depth image of a hand is much smaller than that of a body; ii) A body can be assumed to be upright but a hand can take any orientation; iii) In the case of hands, the number of possible meaningful configurations is much higher and the problem of self occlusion is severe. On the other hand, the inter–personal variance of the shape of hands is much smaller compared to the huge differences between fully clothed human bodies.

Most approaches to hand pose estimation problem make use of regular RGB cameras. Erol *et al.* [7] divide the pose estimation methods into two main groups in their review: partial and full pose estimation methods. They further divide the full pose estimation methods into single frame pose estimation and model-based tracking methods. Athitsos *et al.* [2] estimate 3D hand pose from a cluttered image. They create a large database of synthetic hand poses using an articulated model and find the closest match from this database. De Campos and Murray [5] use a relevance vector machine [19] based learning method for single frame hand pose recovery. They combine multiple views to overcome the self-occlusion problem. They also report single and multiple view performances for both synthetic and real images. Rosales *et al.* [13] use monocular color sequences for recovering 3D hand poses. Their system maps image features to 3D hand poses using specialized mappings. Stergiopoulou and Papamarkos [16] fit a neural network into the detected hand region. They recognize the hand gesture us-

[1]This work has been supported by research grants Tubitak 108E161 and BU-BAP 09M101.

ing the grid of the produced neurons. De La Gorce *et al*. [6] use model-based tracking of the hand pose in monocular videos. Oikonomidis *et al*. [12] present a generative single hypothesis model-based pose estimation method. They use particle swarm optimization for solving the 3D hand pose recovery problem. Stenger *et al*. [15] apply model-based tracking using an articulated hand model and estimate the pose with an unscented Kalman filter.

A number of approaches have been reported to estimate the hand pose from depth images. Mo and Neumann [11] use a laser-based camera to produce low-resolution depth images. They interpolate hand pose using basic sets of finger poses and inter-relations. Malassiotis and Strintzis [10] extract PCA features from depth images of synthetic 3D hand models for training. Liu and Fujimura [9] recognize hand gestures using depth images acquired by a time-of-flight camera. The authors detect hands by thresholding the depth data and use Chamfer distance to measure shape similarity. Then, they analyze the trajectory of the hand and classify gestures using shape, location, trajectory, orientation and speed features. Suryanarayan *et al*. [18] use depth information and recognize scale and rotation invariant poses dynamically. They classify six signature hand poses using a volumetric shape descriptor which they form by augmenting 2D image data with depth information. They use SVM for classification.

In this work, we largely follow the approach in [14]. Adopting the idea of an intermediate representation for the tracked object, we generate synthetic hand images and label their parts, such that each skeleton joint is at the center of one of the labeled parts. We form large datasets created from random and manually set skeleton parameters, and train several randomized decision trees (RDT) [3], which are then used to classify each pixel of the retrieved depth image. Finally, we apply mean shift algorithm to estimate the joint centers as in [14]. The resulting framework can estimate the full hand poses in real time. As a proof of concept, we demonstrate the system by using it to recognize American Sign Language (ASL) digits.

In Section 2 we describe the methodology used for fitting the skeleton. Section 3 lists the details of conducted experiments and presents our results on ASL digit recognition. Finally, we conclude the paper and discuss future work in Section 4.

2. Methodology

The flowchart of the system can be viewed in Figure 1. Our framework handles automatic generation and labeling of synthetic training images by manually setting or randomizing each skeleton parameter. It can then form large datasets by interpolating poses and perturbing joints via addition of Gaussian noise to each joint angle without violating skeleton constraints. Multiple decision trees are then trained on separate datasets, which form forests that boost the per pixel classification accuracy of the system. Finally, posterior probabilities of each pixel are combined to estimate the 3D joint coordinates, and the corresponding skeleton is rendered. The application can connect to Kinect and perform skeleton fitting in real–time without experiencing frame drops.

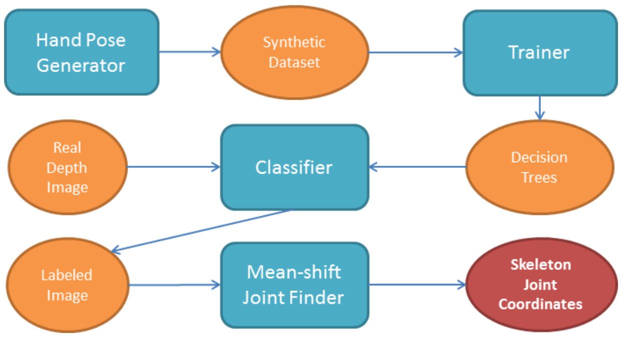

Figure 1. Flowchart of the system

The overall accuracy of the system largely depends on the efficiency of the individual RDTs. If the tree height is excessive or if training images do not reflect the variety of hand poses encountered in real life, the trees overfit and cannot generalize well. A solution is to feed a large number of real life training images to the system. However, Shotton *et al*. reported in [14] that synthetic images are more useful for generalization, since real images of unlikely configurations are rare, and most of the possible configurations do not exist in real datasets. Given that the inter–personal variance is much lower for hands, we decided to put more effort into generating realistic synthetic images that reflect the variance well, and trained the system using these images.

2.1. Data

We use a 3D skinned mesh model with a hierarchical skeleton, consisting of 19 bones, 15 joints and 21 different parts as viewed in Figure 2. The hand parts are chosen in a way to ensure that all significant skeleton joints are in the centers of some parts. Hence, the thumb contains three parts and all the other fingers contain four parts that signify each bone tip. The palm has been divided into two different parts, namely, upper and lower palm, so that the topology of the hand can be inferred better. The model can be animated and rendered with or without shading to generate depth images and labeled images, respectively.

We trained several RDTs on synthetic datasets of size ranging from 60K to 200K images. These datasets are designed with target applications in mind, so that the trained trees can generalize well to previously unseen hand poses that can be encountered during common tasks. These contain mostly common hand poses used for games, natural in-

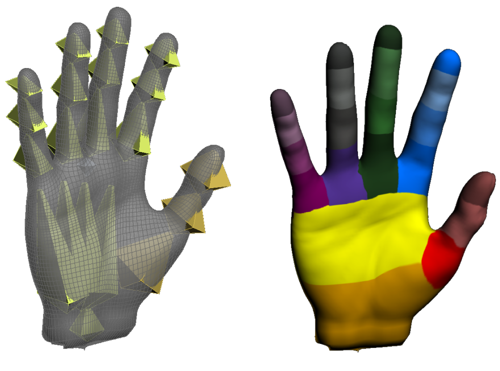

Figure 2. 3D hand model with hierarchical skeleton and labeled parts

terfaces and sign languages, such as the letters of ASL, all of which are manually modeled. The animator tool can interpolate between these poses using the hierarchical skeleton model, and add slight variations to each frame by perturbing joint locations, while changing the camera pose. Skeletal constraints are applied to each interpolated pose, ensuring that the resulting configurations are feasible.

2.2. Features

Since we perform per pixel classification, the complexity of the features has a large impact on the efficiency of the system. We use the same simple depth comparison features as in [14], since their choice proved to be very fast and efficient. For a pixel at coordinate $\boldsymbol{x}$, two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, and depth image I, where $I(\boldsymbol{c})$ gives the depth of the pixel at the coordinate $\boldsymbol{c}$, a feature $F_{\boldsymbol{u},\boldsymbol{v}}(I,\boldsymbol{x})$ can be computed as follows:

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

The offsets $\boldsymbol{u}$ and $\boldsymbol{v}$ are relative to the pixel in question, and normalized according to the depth at $\boldsymbol{x}$. This ensures that the features are 3D translation invariant. Note that, they are neither rotation nor scale invariant, and the training images should be generated accordingly. As background segmentation is easy for depth images, we assume that the pixels corresponding to the hand are known, and the depth values for the rest of the pixel locations, including the outside of the image, are taken to be a large constant value.

2.3. Per Pixel Classification

Per pixel classification is done with RDF [3]. RDF is an ensemble of random decision trees, which improves the stability and accuracy of individual RDT considerably. RDTs on the other hand, perform stochastic discrimination, since they select a random subset of features in each step, which enables controlled variation during training. Moreover, RDTs, and therefore RDFs are very fast, and can be implemented on the GPU.

During the training phase, the nodes of the RDTs learn to split the data into their left and right children in a way that produces *purer* nodes at each step. At a pure node, only samples that belong to the same class remain. In our case, each node learns a set of offsets $\boldsymbol{u}$ and $\boldsymbol{v}$, and a threshold τ, and splits the data consisting of $(I,\boldsymbol{x})$ pairs into two sets for each child, as follows:

$$C_L(\boldsymbol{u},\boldsymbol{v},\tau) = \{(I,\boldsymbol{x})|F_{\boldsymbol{u},\boldsymbol{v}}(I,\boldsymbol{x}) < \tau\} \qquad (2)$$
$$C_R(\boldsymbol{u},\boldsymbol{v},\tau) = \{(I,\boldsymbol{x})|F_{\boldsymbol{u},\boldsymbol{v}}(I,\boldsymbol{x}) >= \tau\} \qquad (3)$$

In the training phase, a large number of pixels and their depth images are input to the RDF. The features and the thresholds are chosen randomly by each node, and the split is scored by the total decrease in entropy:

$$S(\boldsymbol{u},\boldsymbol{v},\tau) = H(C) - \sum_{s\in\{L,R\}} \frac{|C_s(\boldsymbol{u},\boldsymbol{v},\tau)|}{|C|} H(C_s(\boldsymbol{u},\boldsymbol{v},\tau)) \qquad (4)$$

where $H(K)$ is the Shannon entropy estimated using the normalized histogram of the labels in the sample set K. At each node, several random selections for $(\boldsymbol{u},\boldsymbol{v},\tau)$ are made, and the one producing the maximum score is taken as the split criterion for the node.

All trees are trained with 2500 random pixels from each training image of size 160x160. Pixel selection is not forced to be uniform over class labels, as some of the hand parts such as finger tips are significantly smaller than the others. Length of offset vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are selected uniformly between zero and 60 pixels. Threshold τ is selected randomly from a range between a positive and negative depth value corresponding to 200mm, which is roughly the size of a hand. We try up to 4000 different combinations of $\{\boldsymbol{u},\boldsymbol{v},\tau\}$ before choosing the best split for each node.

Classification of a pixel $(I,\boldsymbol{x})$ is performed by starting at the root node and assigning the pixel either to the left or the right child recursively until a leaf node is reached. Each leaf node is associated with a set of posterior probabilities $P(c_i|I,\boldsymbol{x})$ for each class c_i learned during the training phase. For the final decision, the posterior probabilities estimated by all the trees in the ensemble are averaged:

$$P(c_i|I,\boldsymbol{x}) = \frac{1}{N}\sum_{n=1}^{N} P_n(c_i|I,\boldsymbol{x}) \qquad (5)$$

where N is the number of trees in the ensemble.

To enhance the classification results, we evaluate the entire set of pixels on the RDTs to recalculate the class label histograms at the leaves, which can increase the per pixel classification accuracy by 2–3% for individual trees and by 6–8% for forests.

2.4. Joint Position Estimation

After each pixel is assigned posterior probabilities, these are used to estimate the joint positions. Note that this approach is likely to produce many false positives. These outliers have a large effect on the centroid of the pixel locations belonging to a hand part. To reduce the effect of outliers, the mean shift local mode finding algorithm [4] is preferred over finding the global centroid of the points belonging to the same class. Mean shift algorithm estimates the probability density of each class label with weighted Gaussian kernels placed on each sample. Each weight is set to be the pixel's posterior probability $P(c_i|I, \boldsymbol{x})$ corresponding to the class label c_i, times the square of the depth of the pixel, which is an estimate of the area the pixel covers, indicating its importance. The joint locations estimated using this method are on the surface of the hand and need to be pushed back to find an exact match for the actual skeleton. Since the thickness of the hand is nearly uniform, this transformation becomes a simple translation of the entire hand.

Mean shift algorithm uses a gradient ascent approach to locate the nearest maximum point. As the maxima are local, several different starting points are used and the one converging to the highest maximum is selected.

2.5. Occlusion Handling

Since hands are highly articulate and flexible objects, self occlusion of entire hand parts is natural and happens very frequently. Note that, since each pixel is assigned a posterior probability for every class, each class label most likely receives some non–zero probability mass in practice. A decision regarding the visibility of the joint is made by thresholding the highest score reached during the mean shift phase. The effect of the thresholding process is shown in Figure 2.5. Here, the images on the left are the original pixel classification results. The images in the middle are the same images with the corresponding joint locations. These images show spurious joints, which are caused by false positive pixels. The images on the right are produced by thresholding the confidence of each joint. However, if the threshold is too high, this process might eliminate proper joints as well. In the top row, the threshold is too high, and in the bottom row, the threshold is too low.

Several analytical or machine learning based methods can be employed to infer the locations of the missing joints from the known ones. In particular, we trained Artificial Neural Network (ANN) regressors to estimate missing finger joints from the fingers, which are trained on commonly used hand poses to ensure realistic results. For pose classification tasks, however, the descriptive power of the skeleton is high enough even without the missing joints. Assigning the location of each occluded joint to the nearest visible joint proved to be a simple and efficient heuristic for pose classification.

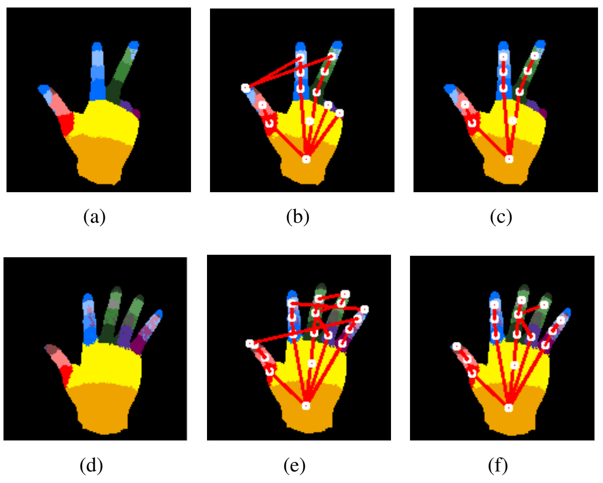

Figure 3. Fitting of the hand skeleton and effect of thresholding the confidence scores of each joint on two examples.

3. Experiments

3.1. Datasets

3.1.1 Synthetic data

Synthetic datasets are used to train the RDFs for per pixel classification task. Performance of RDFs depends heavily on the training set provided. Each training image set consists of a depth image synthetically rendered from the hand model and a corresponding labeled image. A utility is developed for creating individual hand poses and animating an interpolation between such poses. The generated image sequences are then fed to the RDF training module. We used ASL letters, ASL digits and many well known and commonly used hand shapes to generate the animation sequences. Each of these poses are interpolated and viewed from different angles using several extra frames, and each joint in the skeleton is perturbed using Gaussian noise to account for the inter-personal variance of hand shapes. We limited the size of each such dataset to 200,000 images due to memory constraints.

3.1.2 Real data

For pose classification tasks, we collected a dataset consisting of real depth and label images. A dataset for ten ASL digits are captured from ten different people. Each shot takes ten seconds, amounting to a total of 300 frames for each digit per person. To generate the label images, RDFs trained using synthetic datasets are used. Mean shift algorithm is used to estimate the joint locations of the real images, and occluded joints are mapped to the closest joints as explained in Section 2.5. These finalized skeletons form the dataset that is used for pose classification tasks.

3.2. Per pixel classification results

We investigated the effects of tree height, pixel count, $\boldsymbol{u}$, $\boldsymbol{v}$, and τ parameters on the per pixel classification accuracy. Figures 4, 5, and 6 show these effects.

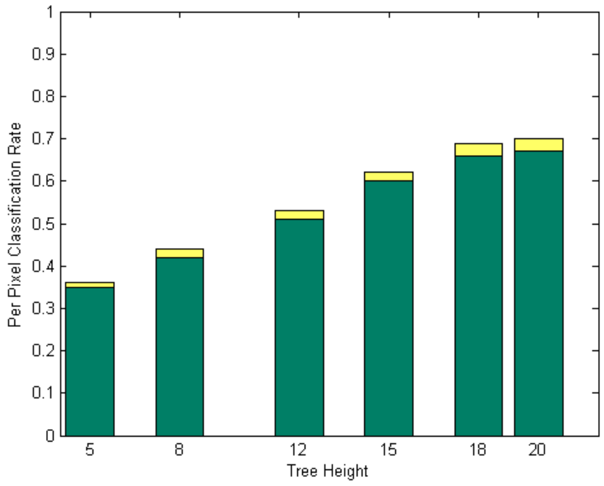

Figure 4. The effect of tree height on per pixel classification accuracy. The extensions over the bars depict the effect of enhancing the trees using the entire dataset. The positive effect of increasing the tree height beyond 20 is outweighed by the impact of increased need for memory.

Increasing the tree height has a positive effect on per pixel classification accuracy as expected, but at the cost of considerable amount of memory for each new level. Therefore, the height of the trees used in the system is limited to 20, and the corresponding per pixel classification rate is 67% after the training and 70% after enhancing the trees. Pixel count needs to be as high as possible, as some parts of the hand are very small, and are under-represented if the number of pixels per image is low. For $\boldsymbol{u}$ and $\boldsymbol{v}$, the best size is found to be 60, and the effect of changes in τ has been found to be negligible as long as it is larger than 20.

3.3. Proof of Concept: American Sign Language Digit Recognizer

To test the system on a real world application, we developed a framework for classifying ASL digits in real–time. The method described in Section 2 gives estimates of the hand skeleton as output. The pose classifier uses these estimates to recognize the digits by mapping the estimated joint coordinates to known hand poses.

To train the classifiers, we first train the RDFs and let them label the real ASL images acquired from the Kinect. The joint location estimates are found using these labels and input to several classifiers that categorize the hand configuration into one of the ASL poses.

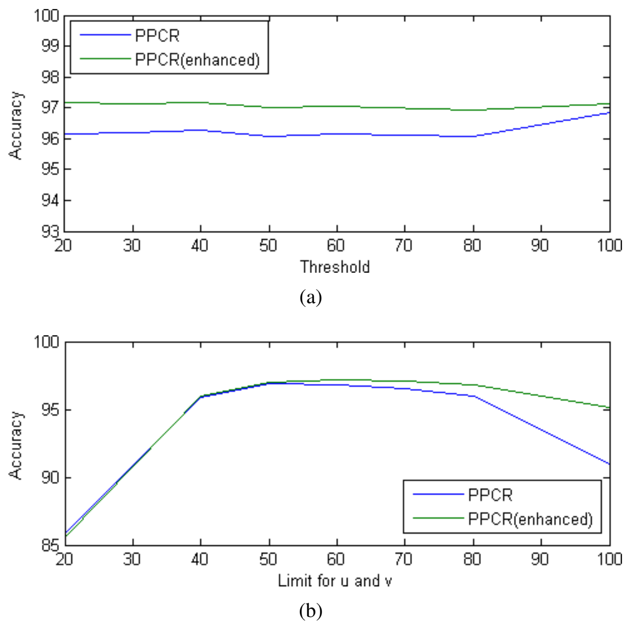

Figure 5. The effects of $\boldsymbol{u}$, $\boldsymbol{v}$, and τ on accuracy

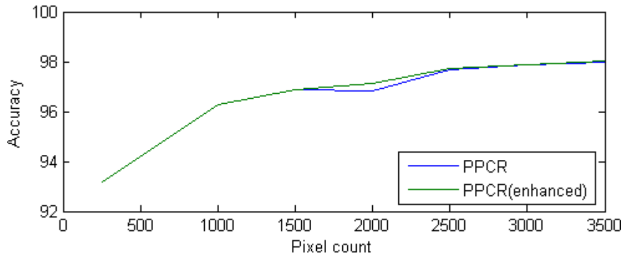

Figure 6. The effect of pixel count on accuracy

3.3.1 Hand Pose Classifiers

As the intended usage of the system is real–time recognition of ASL digits, speed is as important a concern as the recognition rate. For all the classifiers tested, parameter optimization is performed first, and the optimal parameters are used to conduct 5x2 cross validation tests next.

We trained and tested the system on both real and synthetic data. The results on synthetic data are used mainly to optimize the parameters, test the effect of occlusion handling methods as discussed in Section 2.5 and to test the feasibility of tested classifiers for real–time execution. We trained ANNs, SVMs and decision forests on the synthetic data. On real data, we only use ANNs and SVMs.

3.3.2 ASL Digit Classification Results on Synthetic Data

The synthetic dataset used to train the classifiers consists of 20,000 samples, formed by 2000 synthetic images for each of the ASL digits. The images are generated by perturbing the manually designed pose templates using Gaussian noise per joint angle, and by rotating the camera around the hand, so that the same hand pose is viewed from different angles.

For the ANN, the optimum number of hidden nodes is estimated to be 20. For SVMs, the optimal parameters are found to be 2^6 for the cost parameter and 2^{-4} for the Gaussian spread (γ). For decision forests, we used 128 trees to match the accuracy of SVMs.

The classification accuracies and classification times for each of the models are listed in Table 1. The first column gives the average accuracies achieved by the cross–validation tests. The second column gives the running times on a single core. Judging from these results, we prefer SVMs over RDFs as they proved to be more accurate.

The intermediate phases and the final skeletons for several examples are given in Figure 7.

Method Name	Accuracy	Classification Duration (ms)
ANN	**99.89**	**0.0045**
SVM	**99.96**	0.3
RDF-128	**99.94**	0.32

Table 1. Classification rates and evaluation times of each classifier on the ASL digit dataset consisting of 20,000 synthetic images.

3.3.3 ASL Digit Classification Results on Real Data

We conducted 5x2 cross validation and grid search to estimate the optimal parameters of the methods again for the real dataset. Table 2 shows the parameters tested. In Table 3, we show the optimal values of the parameters and the respective accuracies.

Method Name	Parameter Values
ANN	H = {5,10,15,20,25,30,35,40,45,50,55}
SVM	C = $\{2^{-1},2^{0},2^{1},2^{2},2^{3},2^{4},2^{5},2^{6},2^{7}\}$
SVM	$\gamma=\{2^{-5},2^{-4},2^{-3},2^{-2},2^{-1},2^{0},2^{1}\}$

Table 2. Tested parameter values (H: hidden nodes, C: SVM cost, γ: Gaussian spread)

Table 3 lists the optimal parameters and recognition rates on training and validation sets for ANNs and SVMs for real data. SVMs outperform ANNs and reach near perfect accuracy on the validation set, indicating that the descriptive power of the skeleton is sufficient for this task.

Method Name	Optimal Parameters	Training Accuracy	Validation Accuracy
ANN	Hidden nodes = 40	**99.27**	**98.81**
SVM	Cost=2^5, $\Gamma = 2^{-2}$	**100**	**99.90**

Table 3. Optimal parameters and average training and validation accuracies.

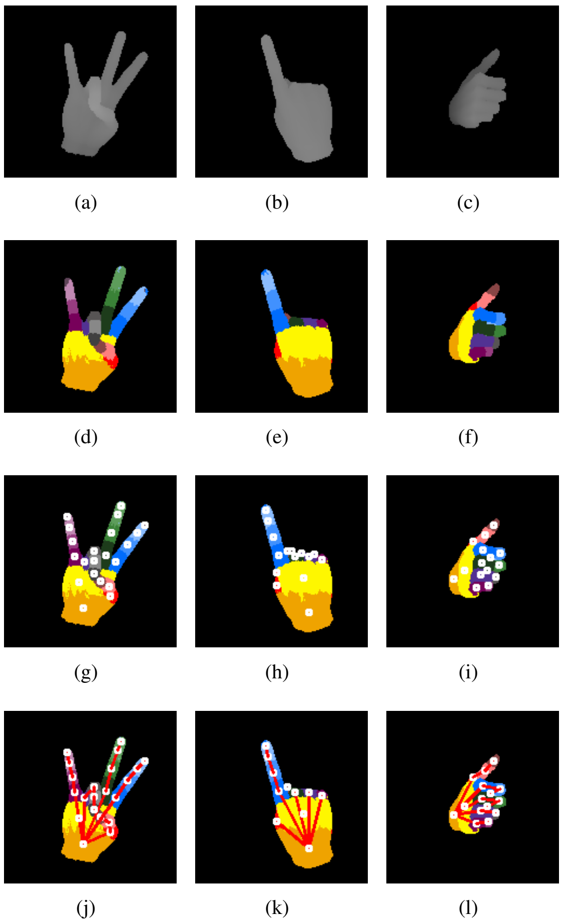

Figure 7. Examples of fitted skeletons on synthetic ASL images. Top row lists the depth images. Second row shows the per pixel classification results. Third row displays the estimated joint locations on top of the labeled images. The finalized skeleton is shown in the bottom row.

4. Discussions and Conclusion

In this study, we described a depth image based real–time skeleton fitting algorithm for the hand, using RDFs to classify depth image pixels into hand parts. To come up with the huge amount of samples that are needed to train the decision trees, we developed a tool to generate realistic synthetic hand images. Our experiments showed that the system can generalize well when trained on synthetic data, backing up the claims of Shotton *et al.* in [14]. In particular, just by feeding manually designed hand poses corresponding to ASL digits to the RDFs, the system learned how to correctly classify the hand parts for real depth images of hand poses that are close enough. This in turn enabled us to collect real data labeled by the RDFs that can be used for further pose classification tasks. We demonstrated the efficiency of this approach by reaching a recognition rate of 99.9% using SVMs on real depth images retrieved with Kinect. The features used by SVMs are the mean shift based joint estimates calculated in real time from the per pixel classification results. We also demonstrated that as long as the number of target classes are low, and the poses are distinct enough, occluded joints carry no importance for classification tasks. We achieved near perfect classification results by mapping each occluded joint to its nearest visible neighbor in the skeleton.

We focused on optimizing the speed and accuracy of the system, in particular by performing grid search over all model parameters. The resulting framework is capable of retrieving images from Kinect, apply per pixel classification using RDFs, estimate the joint locations from several hypotheses in the mean shift phase, and finally use these locations for pose classification at 30 fps, which is the limit of Kinect. The system is optimized for multicore systems and is capable of running on high end notebook PCs without experiencing frame drops. Further enhancement is possible through the utilization of the GPU, and this framework can be used along with more CPU intensive applications such as games and modelling tools. This method is the first to retrieve the full hand skeleton in real time using a standard PC and a depth sensor, and has the extra benefit of not being affected by illumination.

The main focus of this paper is skeleton fitting to the hands from a single frame. Consequently, temporal information is ignored, which can certainly be used to enhance the quality of the fitted skeleton, via methods such as Kalman [20] or particle filtering [8]. Inverse kinematics or some machine learning based inference method can also be used to precisely locate the occluded joints. Such enhancements are indeed possible, and will be the focus of our future work.

References

[1] Microsoft Corp. RedmondWA. Kinect for Xbox 360. 1

[2] V. Athitsos and S. Sclaroff. Estimating 3D hand pose from a cluttered image. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, pages II–432–9, 2003. 1

[3] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 2, 3

[4] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603 –619, may 2002. 4

[5] T. de Campos and D. Murray. Regression-based Hand Pose Estimation from Multiple Cameras. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, pages 782–789, 2006. 1

[6] M. de La Gorce, D. J. Fleet, and N. Paragios. Model-Based 3D Hand Pose Estimation from Monocular Video. *IEEE transactions on pattern analysis and machine intelligence*, pages 1–14, Feb. 2011. 2

[7] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108(1-2):52–73, Oct. 2007. 1

[8] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998. 7

[9] X. Liu and K. Fujimura. Hand gesture recognition using depth data. *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.*, pages 529–534, 2004. 2

[10] S. Malassiotis and M. Strintzis. Real-time hand posture recognition using range data. *Image and Vision Computing*, 26(7):1027–1037, July 2008. 2

[11] Z. Mo and U. Neumann. Real-time Hand Pose Recognition Using Low-Resolution Depth Images. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, pages 1499–1505, 2006. 2

[12] I. Oikonomidis, N. Kyriazis, and A. Argyros. Markerless and efficient 26-DOF hand pose recovery. In *Proceedings of the 10th Asian conference on Computer vision-Volume Part III*, pages 744–757. Springer, 2011. 2

[13] R. Rosales, V. Athitsos, L. Sigal, and S. Sclaroff. 3D hand pose reconstruction using specialized mappings. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2000, pages 378–385. IEEE Comput. Soc, 2001. 1

[14] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, 2011. 1, 2, 3, 6

[15] B. Stenger, P. Mendonça, and R. Cipolla. Model-based 3D tracking of an articulated hand. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, pages II–310–II–315. IEEE Comput. Soc, 2001. 2

[16] E. Stergiopoulou and N. Papamarkos. Hand gesture recognition using a neural network shape fitting technique. *Engineering Applications of Artificial Intelligence*, 22(8):1141–1158, Dec. 2009. 1

[17] P. Suryanarayan, A. Subramanian, and D. Mandalapu. Dynamic Hand Pose Recognition Using Depth Data. *2010 20th International Conference on Pattern Recognition*, pages 3105–3108, Aug. 2010. 2

[18] M. E. Tipping and A. Smola. Sparse bayesian learning and the relevance vector machine, 2001. 1

[19] G. Welch and G. Bishop. An introduction to the kalman filter, 1995. 7