Recognizing Hand Gestures through Random Forest

By

Huang Rongli, Liu Chunguang, Wang He (Group 4)

National University of Singapore

Abstract

This paper tackles the problem of recognizing hand gesture from single depth images. The proposed three-step approach includes hand image preprocessing, feature extraction and prediction through random forest. In practice, our approach works at 20 frames per second on a machine which has a 3.40GHz Intel(R) Core (™) i7-6700 CPU. It can achieve an accuracy level of 0.65 for the given data set. We present a detailed analysis on the results to verify the level of consistency of the mode.

1. Introduction

Hand gesture recognition plays an important role in human-computer interaction. Its challenge lies in the complexity of hand articulations and the variety of poses that hands can make. Traditional methods which model hands as 2 dimensional images, ignore a lot of critical information about 3D hand objects and are unable to deal with self-occlusions. The development of depth camera in recent years, such as the Microsoft Kinect, has facilitated researchers to obtain accurate synthetic 3D hand pose images. We take advantage of the depth cameras technologies and propose an approach to estimate poses from single depth hand images.

Our data training session has three steps: initial preprocessing step detects hand patch and remove background from each image; Feature extraction step extract distance-invariant from each of the image pixels; Model generation step take depth features as input and output a random forest tree. In the data testing session, initial preprocessing step and feature extraction step is followed by the estimation step when depth features are input to the random forest tree we have obtained and output a hand pose estimation. Our approach has been demonstrated to be efficient and can achieve 65 percent of accuracy. The main contributions are:

- Our preprocessing step has deleted some of the low quality images, reduced noises, extracted hand patches and removed background. It provides future researchers with cleaner and clearer images.
- The problem of in hand occlusion is solved by using depth features.
- The model can be extended to human pose recognition, 3D object recognition and other problems which 3D information is available.

Related work

We have reviewed different approaches from the literature. Regression forest, bag of visual words, the fingertips approach, pixel matching and approaches using RGB information. Some of the methods requires very careful extraction of the minor details on the image, such as calculating the coordinates of each finger tips and the angel each finger pointing to. This makes the method less robust because the minor details may vary across different images and comprehensive rules are required to cover all possible cases.

Comparing to the above mentioned method, the method used in [1] and [2] shows a much neater feature extraction, treating each input image as a whole and perform feature extraction without considering the very minor details as mentioned above. It uses regression forest to perform hand pose estimation with high accuracy. Being inspired by this method, we decided to use random forest to perform gesture recognition.

2. Proposed Approach

2.1. Preprocessing: detect hand patches

Our approach starts with preprocessing the images. It includes the following three operations:

1. Gaussian smoothing. The original image has a lot of noises, which would be wrongly regarded as contour in later stage.
2. Otsu thresholding. It assumes the image only contains two classes of pixels and calculate the optimum threshold to separate the pixels. The output is a binary image which has only 2 pixel values, 0 and 255. Since the hand is closer to the camera than the background and has lower pixel values, we assume hand patch has pixel values of 0 and the background has pixel values 255.
3. Recover the depth values. Pixels inside the contour are recovered to its original value and pixel outside the contour are set to 255, because the background is assumed the furthest distance from the camera. Note that even though there are still some noises inside and at the edge of the hand patch, we do not conduct Gaussian blur here, because that will also remove the slight depth differences among pixels and reduce our accuracy.

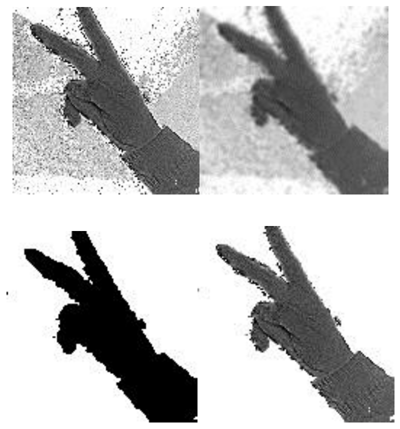


Figure 1. Four images from left to right are respectively: original image cropped according to the given hand location, Gaussian blurred image, hand

patch detected by otsu thresholding, recover hand pixel and reset background pixels.

2.2. Feature extraction

We employ the same depth feature extraction method as [3]. Given pixel location *x* in image I and a mapping to depth value *d(x)*, the feature value is defined as:

$$f_I(x) = d_I\left(x + \frac{u}{d_I(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right).$$

Where *u* and *v* are arbitrary two dimensional offset vectors. The normalization factor $\frac{1}{d(x)}$ ensures the features are depth invariant. As a small variation to [3], if an offset pixel lies outside the bounds of the image, we mod the pixel location by the size of the image to relocate it to a pixel that lies inside the image frame, as opposed to the original method that just assign *d(x')* a large positive constant value. The reason of doing this is that the range of u, v are chosen to be large enough in order for *i'* to have equal probability to be located in each pixel in the image frame (we decide the range to be [0, 20000], otherwise *x* will only be located close to *x*), and this leads to *x'* to be pixels outside the image frame which we are not interested in. Hence, we conduct the modulo to cast *x'* back within the image frame.

With one set of u, v, the formula converts a depth image into a feature matrix of the same size. We future concatenate the rows of the feature matrix to make it a one dimensional array. We continue to choose n sets of *u, v* and derive a final feature matrix of *n* rows and *p* columns, where *p* is the number of pixels chosen from the depth image.

For each training image, we select 65 x 65 uniformly sampled pixels and 10 combinations of predefined random (*u*, *v)*, rendering a feature matrix of 4225 x 10. Each *(u,v)* combination will give a predicted result. The final predict result will be the majority among the 10 combinations of (*u*, *v)*. We are adding one more level of voting, on top of the voting mechanism inherent from random forest itself.

2.3. Training with a random forest

Random Forest classifier is used to perform the prediction. Random Forest has good performance when samples have large number of dimensions. It utilizes multiple decision trees, where each tree performs prediction based on a certain proportion of the features. Each tree will generate an output as a vote, and the final predicted result will be the voted majority from all the trees.

Random Forest has several advantages:

1. It is naturally suitable for categorical prediction
2. Speed efficient, quick to train compared to other models such as neural network.
3. Able to categorize non-linear classification boundaries, because it uses decision trees.
4. It can indicate the importance of each feature

Considering that we have 4225 features for

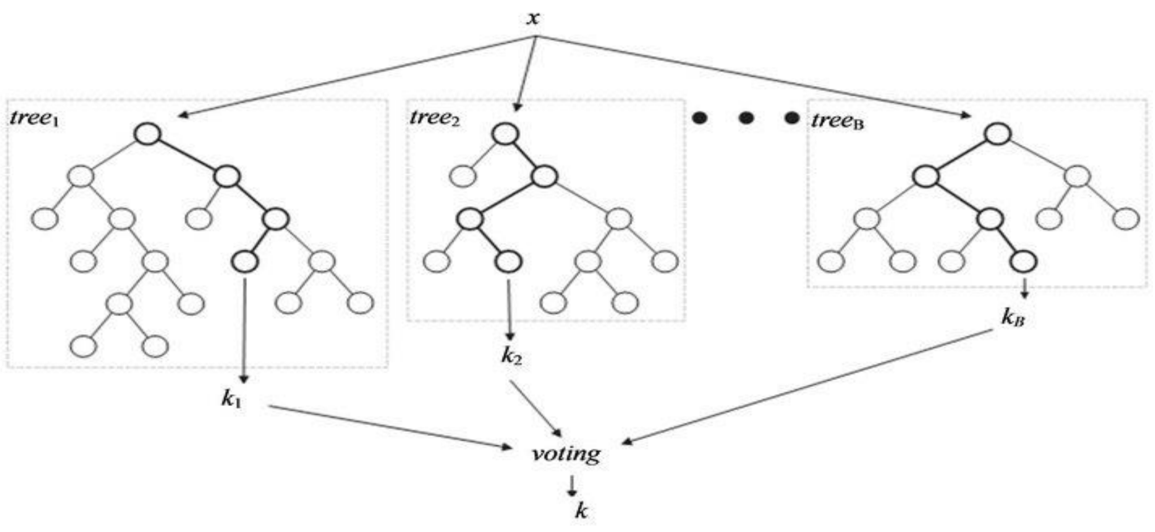


Figure 2. Random Forest classifier

each image, we configure our random forest classifier to be using 100 trees. If there are total N samples, when growing each tree, sampling is performed N times with replacement. Thus, the number of training samples when growing each tree is N, but some samples may appears multiple times while some samples may not appear in the training samples for this tree. This is also called bootstrapping, which is used to reduce variance of the base learner. Each node uses a proportion of features (in this case *sqrt(M)=65* features), split based on entropy information gain.

3. Experiments and Results

3.1 Dataset and Setup

The dataset we are going to use is provided by Xu, C. and Cheng, L.[1]. It has 49 subject folders collected from 49 human hands. In each subject folder, there are 35 folders of sorted gesture images, each is represented by a gesture label {1, 2, …, 10, A, B, C, …, Y, wave}. Similar gestures have been merged together, which are {8, L}, {6, Y}, {5, Wave}, {9, X}, {3,F}, {10,S}, {V,2}. Under each gesture folder, there are around 20 images of the same gesture but at different orientation. We performed two simulations:

1. Randomly select half of the image in each gesture folder as test dataset and the rest are training dataset. In total 15322 images for training, 14556 images for testing
2. Randomly select 3 images from each gesture folder as test dataset and the rest are training dataset. In total 26116 images for training, 3792 images for testing.

3.2. Result

Simulation 1: Accuracy varies from around 0.2 to 0.8 across different subject folders. Overall accuracy is 0.58, and each subject folder has 300 images tested on average.

Time consumed: 1hr generating training data + 10 mins fitting + 10mins predicting.

Simulation 2: Accuracy varies from around 0.9 to 0.3 across different subject folders. Overall accuracy is 0.65, and each subject folder has 300 images tested on average. Note that for some folders, accuracy reach as high as 0.95. Time consumed: 3hrs generating training data + 20 mins fitting + 4 mins predicting

Each data point on the below graph indicates the accuracy for a subject folder. Notably, different subject folders have different accuracy.

It is expected that *Simulation 2* gives better accuracy than *Simulation 1*, because *simulation 2* uses more images for training.

We identified some factors that reduce the accuracy.

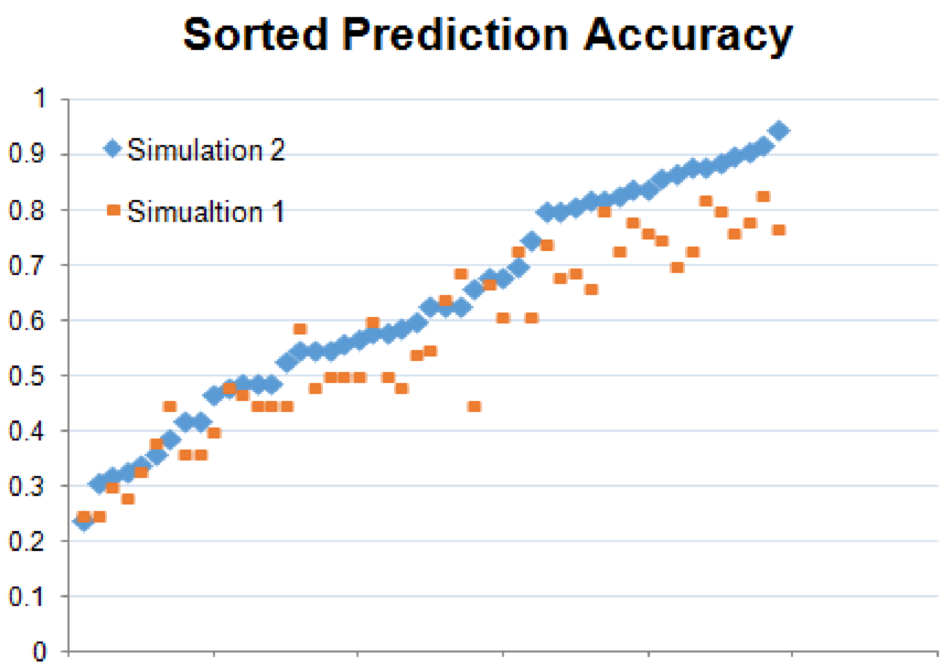


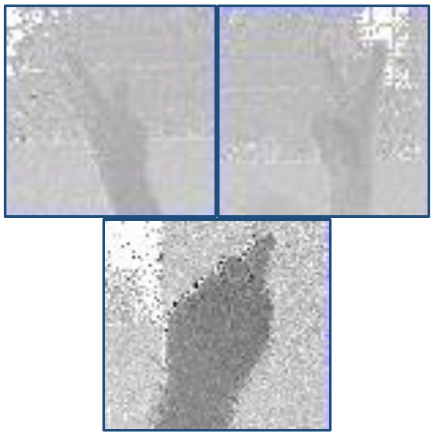
Figure 3. Accuracy by subject folders. X-axis: Different subject folders, in order from low accuracy to high accuracy

4. Analysis

It is observed that the accuracy of different subject folders vary greatly, from as low as 0.3 to as high as 0.95. See appendix a table of accuracy by subject folder, sorted from high to low. After inspecting some high accuracy folders and low accuracy folders,

4.1. Image noise

It is noticed that the image quality is not consistent across different subject folders, resulting in different estimation accuracy. Below are examples of low-quality images whose hand patch or gesture can hardly be determined.



4.2. Self-occlusion

Hand orientation, especially rotation in y-axis (axis along the elbow) also makes it difficult to estimate hand gesture. Perfect top down hand orientation gives higher accuracy of around 0.8-0.9, while rotation in y-axis result in large self-occlusion, lowering accuracy down to around 0.2-0.3.

5. Implementation and Efficiency

The program is written in python, packages used are cv2 and sklearn and numba. In order to improve the speed for this scripting language, we use '@jit' feature to compile the most computation intensive function--the feature extraction function, improving speed by 7 folds. During simulation, we also disable other non-related processes on the computer. After optimization, total running time is 1.5 hr for simulation 1 and 3.5 hr for simulation 2.

6. Conclusion and Future Work

Besides random forest, other methods can also be applied to this problem. Deep learning is another popular model, which uses convolutional neural network, and is capable of achieve around 0.8-0.9 accuracy. Although it is higher than our overall result, the capability is close because some of subject folders shows 0.95 accuracy in our model, which match the accuracy of deep learning.

Thus, one possible area for improvement could be image preprocessing on lower quality images, which can improve the data quality and the overall result should be close to deep learning.

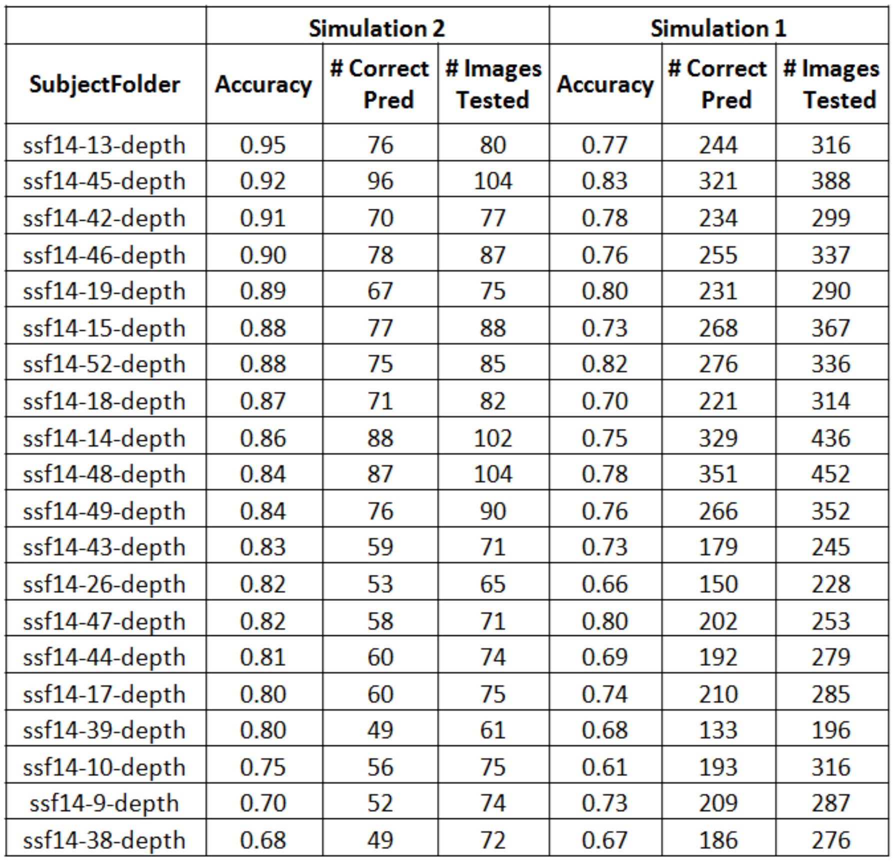
Reference

1. Xu, C., Cheng, L.: Efficient hand pose estimation from a single depth image. In: ICCV (2013)

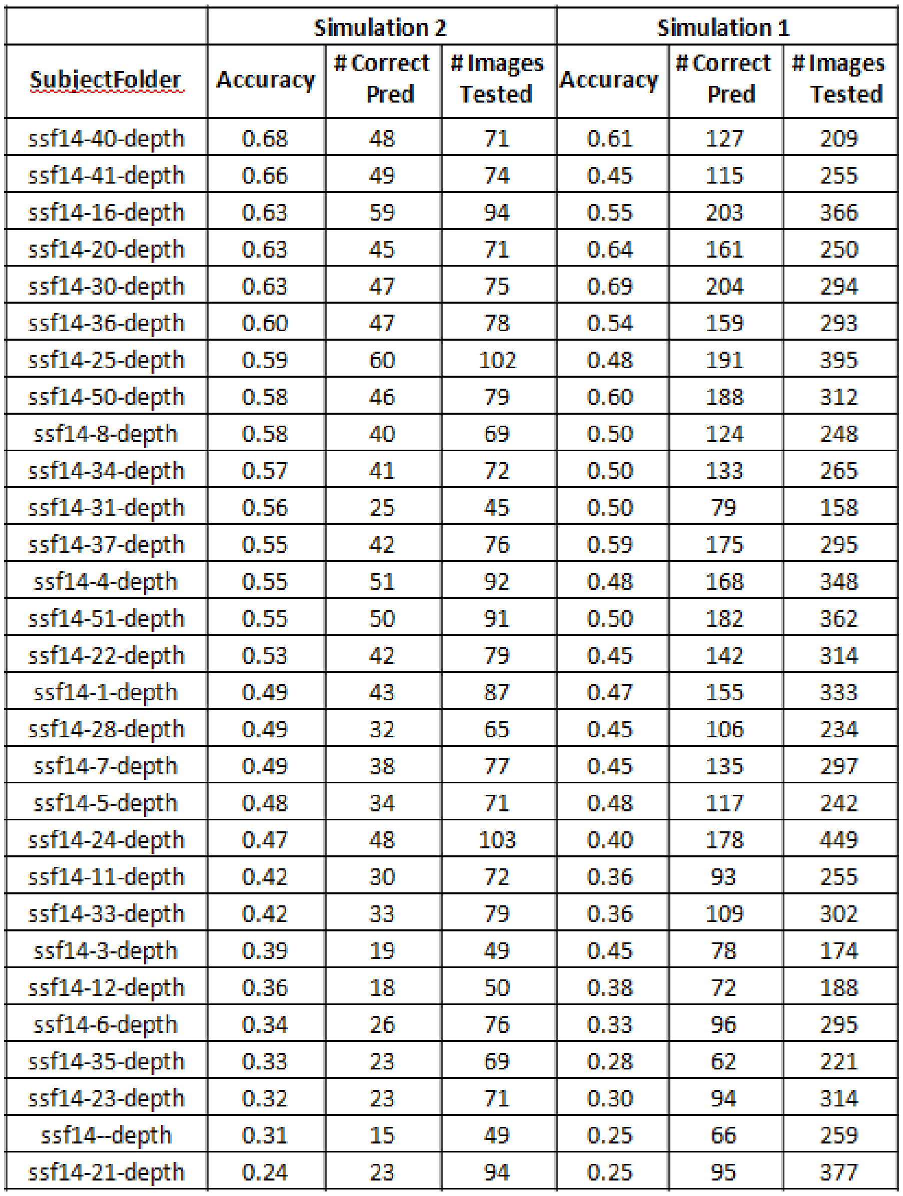
2. Chi Xu, Ashwin Nanjappa, Xiaowei Zhang, Li Cheng. Estimate Hand Poses Efficiently from Single Depth Images. In In International Journal of Computer Vision (IJCV), 2015.

3. 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.

Appendix



	Simulation 2			Simulation 1		
SubjectFolder	Accuracy	# Correct Pred	# Images Tested	Accuracy	# Correct Pred	# Images Tested
ssf14-13-depth	0.95	76	80	0.77	244	316
ssf14-45-depth	0.92	96	104	0.83	321	388
ssf14-42-depth	0.91	70	77	0.78	234	299
ssf14-46-depth	0.90	78	87	0.76	255	337
ssf14-19-depth	0.89	67	75	0.80	231	290
ssf14-15-depth	0.88	77	88	0.73	268	367
ssf14-52-depth	0.88	75	85	0.82	276	336
ssf14-18-depth	0.87	71	82	0.70	221	314
ssf14-14-depth	0.86	88	102	0.75	329	436
ssf14-48-depth	0.84	87	104	0.78	351	452
ssf14-49-depth	0.84	76	90	0.76	266	352
ssf14-43-depth	0.83	59	71	0.73	179	245
ssf14-26-depth	0.82	53	65	0.66	150	228
ssf14-47-depth	0.82	58	71	0.80	202	253
ssf14-44-depth	0.81	60	74	0.69	192	279
ssf14-17-depth	0.80	60	75	0.74	210	285
ssf14-39-depth	0.80	49	61	0.68	133	196
ssf14-10-depth	0.75	56	75	0.61	193	316
ssf14-9-depth	0.70	52	74	0.73	209	287
ssf14-38-depth	0.68	49	72	0.67	186	276



	Simulation 2			Simulation 1		
SubjectFolder	Accuracy	# Correct Pred	# Images Tested	Accuracy	# Correct Pred	# Images Tested
ssf14-40-depth	0.68	48	71	0.61	127	209
ssf14-41-depth	0.66	49	74	0.45	115	255
ssf14-16-depth	0.63	59	94	0.55	203	366
ssf14-20-depth	0.63	45	71	0.64	161	250
ssf14-30-depth	0.63	47	75	0.69	204	294
ssf14-36-depth	0.60	47	78	0.54	159	293
ssf14-25-depth	0.59	60	102	0.48	191	395
ssf14-50-depth	0.58	46	79	0.60	188	312
ssf14-8-depth	0.58	40	69	0.50	124	248
ssf14-34-depth	0.57	41	72	0.50	133	265
ssf14-31-depth	0.56	25	45	0.50	79	158
ssf14-37-depth	0.55	42	76	0.59	175	295
ssf14-4-depth	0.55	51	92	0.48	168	348
ssf14-51-depth	0.55	50	91	0.50	182	362
ssf14-22-depth	0.53	42	79	0.45	142	314
ssf14-1-depth	0.49	43	87	0.47	155	333
ssf14-28-depth	0.49	32	65	0.45	106	234
ssf14-7-depth	0.49	38	77	0.45	135	297
ssf14-5-depth	0.48	34	71	0.48	117	242
ssf14-24-depth	0.47	48	103	0.40	178	449
ssf14-11-depth	0.42	30	72	0.36	93	255
ssf14-33-depth	0.42	33	79	0.36	109	302
ssf14-3-depth	0.39	19	49	0.45	78	174
ssf14-12-depth	0.36	18	50	0.38	72	188
ssf14-6-depth	0.34	26	76	0.33	96	295
ssf14-35-depth	0.33	23	69	0.28	62	221
ssf14-23-depth	0.32	23	71	0.30	94	314
ssf14--depth	0.31	15	49	0.25	66	259
ssf14-21-depth	0.24	23	94	0.25	95	377