Exemplar-Based Human Action Pose Correction and Tagging

Wei Shen
Huazhong Univ. of Sci.&Tech.

Ke Deng
Microsoft Corporation

Xiang Bai
Huazhong Univ. of Sci.&Tech.

Tommer Leyvand
Microsoft Corporation

Baining Guo
Microsoft Research Asia

Zhuowen Tu
Microsoft Research Asia & UCLA

Abstract

The launch of Xbox Kinect has built a very successful computer vision product and made a big impact to the gaming industry; this sheds lights onto a wide variety of potential applications related to action recognition. The accurate estimation of human poses from the depth image is universally a critical step. However, existing pose estimation systems exhibit failures when faced severe occlusion. In this paper, we propose an exemplar-based method to learn to correct the initially estimated poses. We learn an inhomogeneous systematic bias by leveraging the exemplar information within specific human action domain. Our algorithm is illustrated on both joint-based skeleton correction and tag prediction. In the experiments, significant improvement is observed over the contemporary approaches, including what is delivered by the current Kinect system.

1. Introduction

With the development of high-speed depth cameras [1], the computer vision field has experienced a new opportunity of applying a practical imaging modality for building a variety of systems in gaming, human computer interaction, surveillance, and visualization. A depth camera provides depth information as different means to color images captured by the traditional optical cameras. Depth information gives extra robustness to color as it is invariant to lighting and texture changes, although it might not carry very detailed information of the scene.

The human pose estimation/recognition component is a key step in an overall human action understanding system. High speed depth camera with the reliable estimation of the human skeleton joints [22] has recently led to a new consumer electronic product, the Microsoft Xbox Kinect [1].

Even though the depth data provides invariant and informative cues, existing systems (e.g. classification-based approaches for skeleton joint detection [22]) are not all satisfactory due to severe occlusions. Fig. 1 illustrates a pipeline of pose recognition, which includes three important steps: (1) background removal, (2) initial pose estimation, and (3) pose correction. After the background is removed from the depth image, skeleton joints are estimated from the foreground depth information using e.g. [22, 10]. Note that, serious errors exist in the estimated skeleton in Fig. 1(c). The pose correction stage further takes in these initial per-frame estimations (as "noisy" input) and tries to correct the errors and deliver more robust results. In this paper, we focus on the third step, *pose correction*, which also plays a very important role in the overall pose detection/recognition pipeline. To show to what extent the third stage can help, the pose correction stage performs "de-noising" by working on extracted "noisy" skeleton joints only, without looking back at the original depth data.

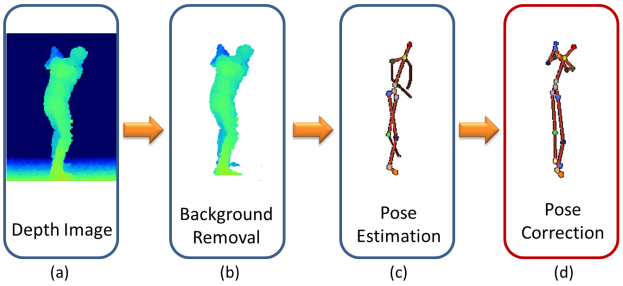

Figure 1. Pose recognition pipeline. From a depth image of single frame, the pipeline includes background removal, initial pose estimation, and pose correction. The skeleton joints marked by color dots in (c) are the ones with high confidence in estimation whereas the ones without color dots are with the low confidence.

To perform pose correction, e.g. obtaining the result shown in Fig. 1(d) from a noisy estimation in Fig. 1(c), two types of information can be leveraged: (1) temporal motion consistency and (2) the systematic bias. Using the temporal information to enforce the motion consistency has been extensively studied in the literature [11, 6] but studying the systematic bias has received relatively less attention. Generally, the biases are no-linear and are associated with complex data manifolds. As illustrated in Fig. 2, the systematic biases do exist, especially in domain specific actions.

For a general estimation problem, its systematic bias might be significant and has an explicit analytic function [2]. In our task of human action pose recognition, each pose

is represented by a number of skeleton joints; each joint is estimated/extracted using local and contextual depth features [22]. The bias estimation problem in our task observes two properties: (1) human action has certain (sometimes strict) regularity especially when some actions, e.g. golf or tennis, are performed, and (2) the bias is not homogeneous in the data manifold. For example, when a person is facing the camera with no occlusion, the initial estimations are accurate; when a person is standing in a side-view with certain hand motion, there is severe occlusion and the initial estimation may not be all correct, as illustrated in Fig. 1. In this paper, the main contribution is learning the inhomogeneous bias function to perform pose correction and we emphasize the following two points: (1) exemplar-based approach serves a promising direction for pose correction in depth images, (2) learning an inhomogeneous regression function should naturally perform data-partition, abstraction, and robust estimation. With a thorough experimental study, our approach shows encouraging results.

Motion and action recognition from optical cameras has been an active research area in computer vision; typical approaches include 3D feature-based [7], part-based (poselets) [3], and segment-based [16] approaches. Although insights can be gained, these methods are not directly applicable to our problem.

From a different view, bias estimation has been a long standing problem in statistics [2]. Related work in the pose correction task uses physical-model-based approaches [23], Kalman-like filters [12], or exemplar-based approaches but with very specific design, which is hard to adapt to the general task [15]. Here we adopt the random forest regressor, which takes in both the estimated solutions and their estimation uncertainties. We show significant improvement over other regressors such as nearest neighborhood [9], Gaussian process regression [20], support vector regression [21], and logistic regression [17]. Our approach is real-time and can be directly applied to the Kinect system.

2. Data

In this section, we introduce the data used for our pose correction problem. The recently launched Kinect camera [1] is able to give 640×480 image at 30 frames per second with depth resolution of a few centimeters. Employing the Kinect camera, we are able to generate a large number of realistic depth images of human poses. The human skeleton estimated from the depth image by the current Kinect system [22] is the direct input for our approach, which is called *ST* (Skeleton esTimation) in the rest of this paper. As shown in Fig. 3(a), there are 20 body joints in a skeleton, including hips, spine, shoulders, head, elbows, wrists, hands, knees, ankles, and feet.

As suggested by the recent work [22, 10], the ground truth positions of the body joints can be captured by motion capture (mocap) system. We obtain a set of mocap of human actions as the ground truth of the estimated skeletal joints. The mocap data is also called *GT* (Ground Truth) in the rest of this paper. In our experiments, we limit the rotation of the user to $\pm 120^\circ$ in both training and testing data. Fig. 3(b) shows several pairs of *ST* and *GT*.

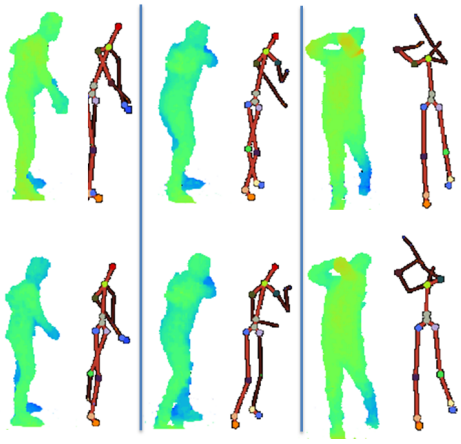

Figure 2. Three groups of poses. In each group, the poses are similar, and the errors of the estimated skeletons are somewhat systematic, e.g. the right forearms of the skeletons in the second group and those in the third group.

3. Pose Correction

3.1. Objectives

We focus on two tasks: joint-based skeleton correction and pose tag prediction. Our inputs are m estimated skeletons $\mathbf{st} = (ST_1, \ldots, ST_m)$ from a video sequence of m depth image frames. Each skeleton estimation ST includes n ($n = 20$ here) joints: $ST = (\hat{\mathbf{x}}_j, c_j; j = 1, \ldots, n)$, where $\hat{\mathbf{x}}_j \in \mathbb{R}^3$ denotes the world coordinates of the jth body joint, as shown in Fig. 3. c_j indicates the confidence for the estimation $\hat{\mathbf{x}}_j$ by the skeleton joint detector, i.e. if joint j has high confidence, $c_j = 1$; Otherwise, $c_j = 0$. As shown in Fig. 1 and Fig. 2, color dots correspond joints of high confidence whereas joints without color dots are with low confidence.

The first task (joint-based skeleton correction) is to predict the "true" position of each joint: $\hat{\mathbf{x}}_j \rightarrow \mathbf{x}_j$ and the "true" skeletons $\mathbf{gt} = (GT_1, \ldots, GT_m)$ where each $GT = (\mathbf{x}_j; j = 1, \ldots, n)$ and $\mathbf{x}_j \in \mathbb{R}^3$. In training, we are given a training set of $\{(\mathbf{st}, \mathbf{gt})_k\}_{k=1}^K$ of K pairs of $\mathbf{st}$ and $\mathbf{gt}$; in testing, we want to predict the "true" $\mathbf{gt}$ from a given input $\mathbf{st}$.

The second task (pose tagging) is to predict the pose tag $\boldsymbol{\Upsilon} = (\Upsilon_1, \ldots, \Upsilon_m)$ from a given $\mathbf{st} = (ST_1, \ldots, ST_m)$. The tag is a real value ranging from 0.0 to 1.0, indicating a specific stage in a particular action, e.g. golf swing. Each type of poses is assigned a tag, as illustrated in Fig. 4. In the somatosensory game, the tag is used to drive the avatar to put on the same pose as the player performs. In training,

Figure 3. Skeleton data. (a) A template skeleton and its joints. The skeleton is a simple directed graph, in which the directions are denoted by the arrows beside the skeleton edges. For denotational simplicity, we do not show the arrows in other figures. (b) Several pairs of input noisy skeletons (upper) and ground truth skeletons (lower).

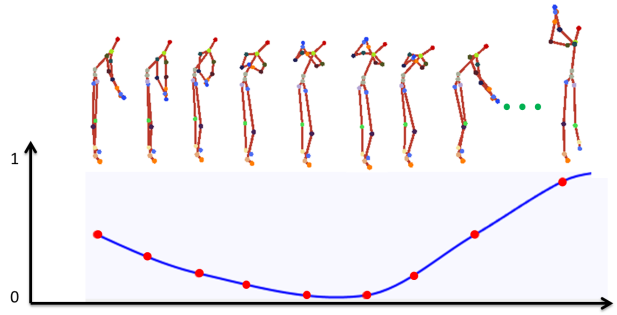

Figure 4. Pose tag. Each type of poses are assigned to a tag, e.g. the tag of "front swing" (the last pose) is about 0.95.

we are given a training set of $\{(\mathbf{st}, \boldsymbol{\Upsilon})_k\}_{k=1}^K$ of K pairs of $\mathbf{st}$ and $\boldsymbol{\Upsilon}$; in testing, we want to predict the tag values $\boldsymbol{\Upsilon}$ from a given input $\mathbf{st}$.

The tag is actually a low dimensional manifold coordinate of the pose. Both of the two tasks are performed to recover the pose from a noisy initial estimation, so we categorize them into the tasks of pose correction.

3.2. Normalized Skeleton Joints Coordinates

From the world coordinates, we want to have an intrinsic and robust representation. Based on the $n = 20$ joints, we show the kinematic skeleton template as displayed in Fig. 3(a), which is a directed graph. Each joint is a node of the graph. Given an $ST = (\hat{\mathbf{x}}_j, c_j; j = 1, \ldots, n)$, we compute its normalized coordinates for our problem, denoted as $H(ST) = (\mathbf{r}_j, c_j; j = 1, \ldots, n)$. Since $\hat{\mathbf{x}}_j$ denotes the world coordinate, we normalize them to the template to remove not only the global translation but also the variation in individual body differences. We use the skeleton joint C. Hip as the reference point, the origin, $\mathbf{r}_1 = (0, 0, 0)$, and map the other joints to the sphere as $\mathbf{r}_j = \frac{\hat{\mathbf{x}}_j - \hat{\mathbf{x}}_{j_o}}{\|\hat{\mathbf{x}}_j - \hat{\mathbf{x}}_{j_o}\|_2}$ where joint j_o is the *direct predecessor* of joint j on the skeleton (directed graph).

The design of the transformed coordinates $H(ST)$ is motivated by the kinematic body joint motion. $H(ST)$ observes a certain level of invariance to translation, scaling, and individual body changes. We can actually drop $\mathbf{r}_1$ since it is always on the origin. One could add other features computed on ST to $H(ST)$ to make it more informative but this could be a topic of future research.

3.3. Joint-based Skeleton Correction

3.3.1 Joint offset inference

To perform skeleton correction from a noisy input ST, instead of directly predicting the "true" positions of the body joints, we infer the offsets between the joints in the ST and those in the GT. This has its immediate significance: when a person is facing the camera with no occlusion, ST is actually quite accurate, and thus has nearly zero difference to GT; when a person is playing the game in side view with severe occlusions, there is often a large and inhomogeneous difference between ST and GT. This is essentially a manifold learning problem. Certain clusters of ST on the manifold can directly be mapped to, e.g. very low values, if we would predict the offsets; predicting the direct coordinates of GT however would have to explore all possible ST in the data space.

Now we show how to compute the offsets between the joints in $ST = (\hat{\mathbf{x}}_j, c_j; j = 1, \ldots, n)$ and those in $GT = (\mathbf{x}_j; j = 1, \ldots, n)$, where $\hat{\mathbf{x}}_j, \mathbf{x}_j \in \mathbb{R}^3$ are the world coordinates of joint j in ST and GT, respectively. To ensure the scale invariance of the offsets, skeletons are normalized based on the default lengths of the skeleton edges in the template shown in Fig. 3(a). First, we choose a set of stable joints $J_S = \{$Spine, C. Shoulder, Head, L. Hip, R. Hip, L. Knee, R. Knee, L. Ankle, R. Ankle $\}$. We call them stable joints because other joints in the ST, such as Hand and Wrist, are often occluded by the human body, thus the skeleton edge lengths between them are prone to errors. Given an ST, for each skeleton edge between the stable joints and their direct predecessors, we compute the proportion to the template skeleton edge length: $\lambda(j, j_o) = \|\hat{\mathbf{x}}_j - \hat{\mathbf{x}}_{j_o}\|_2 / \|\mathcal{T}_j - \mathcal{T}_{j_o}\|_2$, where $\mathcal{T}_j$ is the jth joint for the

template $\mathcal{T}$ (shown in Fig. 3), which is fixed in this problem. Then, the scale proportion of the ST is

$$\lambda(ST) = \frac{\sum_{j \in J_S} \lambda(j, j_o) \cdot \delta(\|\lambda(j, j_o) - \frac{\sum \lambda(j, j_o)}{|J_S|}\|_1 \leq th)}{\sum_{j \in J_S} \delta(\|\lambda(j, j_o) - \frac{\sum \lambda(j, j_o)}{|J_S|}\|_1 \leq th)}, \tag{1}$$

where $\delta(\cdot)$ is an indicator function which is a robust measure to exclude the outliers and

$$th = 3\sqrt{\frac{\sum_{j \in J_S} (\lambda(j, j_o) - \frac{\sum \lambda(j, j_o)}{|J_S|})^2}{|J_S|}}. \tag{2}$$

Finally, the offset of a joint j between the pair of $\hat{\mathbf{x}}_j$ and $\mathbf{x}_j$ is computed as

$$\boldsymbol{\Delta}_j = (\mathbf{x}_j - \hat{\mathbf{x}}_j)/\lambda(ST), \tag{3}$$

and $\mathbf{D} = (\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_n)$ for each skeleton of n joints. For the entire sequence of m images, we have $\mathbf{d} = (\mathbf{D}_1, \ldots, \mathbf{D}_m)$. Note that we do not need to explicitly align the pair of ST and GT, since they are obtained from the same depth image.

3.3.2 Learning the regression for joint offsets

In this section, we discuss how to learn a regression function to predict the offset to perform pose correction. We are given a training set of $\mathcal{S} = \{(\mathbf{st}, \mathbf{gt})_k\}_{k=1}^K$ of K pairs of $\mathbf{st}$ and $\mathbf{gt}$ (for denotational simplicity, we let $K = 1$ and thus k can be dropped for an easier problem understanding). Using the normalization step in Sec. 3.2, we obtain $h(\mathbf{st}) = (H(ST_1), \ldots, H(ST_m))$ where each $H(ST) = (\mathbf{r}_j, c_j; j = 1, \ldots, n)$; using the offset computing stage in Eq. 3, we compute the offset, $\mathbf{d} = (\mathbf{D}_1, \ldots, \mathbf{D}_m)$. Thus, our goal is to predict the mapping $h(\mathbf{st}) \rightarrow \mathbf{d}$.

We first learn a function to directly predict the mapping $f : H(ST) \rightarrow \mathbf{D}$ by making the independent assumption of each pose. From this view, we rewrite the training set as $\mathcal{S} = \{(H(ST_i), \mathbf{D}_i)\}_{i=1}^m$.

Random forest [4, 18, 14] includes an ensemble of tree predictors that naturally perform data-partition, abstraction, and robust estimation. For the task of regression, tree predictors take on vector values and the forest votes for the most possible value. Each tree in the forest consists of split nodes and leaf nodes. Each split node stores a feature index with a corresponding threshold τ to decide whether to branch to the left or right sub-tree and each leaf node stores some predictions.

Our objective is to learn a random forest regression function $f : H(ST) \rightarrow \mathbf{D}$. Following the standard greedy decision tree training algorithm [13, 22, 10], each tree in the forest is learned by recursively partitioning the training set $\mathcal{S} = \{(H(ST_i), \mathbf{D}_i)\}_{i=1}^m$ into left $\mathcal{S}_l$ and right $\mathcal{S}_r$ subsets according to the best splitting strategy $\theta^* = \arg\min_\theta \sum_{p \in \{l,r\}} \frac{|\mathcal{S}_p(\theta)|}{|\mathcal{S}|} e(\mathcal{S}_p(\theta))$, where $e(\cdot)$ is an error function standing for the uncertainty of the set and θ is a set of splitting candidates. If the number of training samples corresponding to the node (node size) is larger than a maximal κ, and $\sum_{p \in \{l,r\}} \frac{|\mathcal{S}_p(\theta^*)|}{|\mathcal{S}|} e(\mathcal{S}_p(\theta^*)) < e(\mathcal{S})$ is satisfied, then recurse for the left and right subsets $\mathcal{S}_l(\theta^*)$ and $\mathcal{S}_r(\theta^*)$, respectively.

The selection of the error function $e(\cdot)$ is important for learning an effective regressor. Here, we employ the rooted mean squared differences:

$$e(\mathcal{S}) = \sqrt{\frac{\sum_{i=1}^m \|\mathbf{D}_i - \frac{\sum_{i=1}^m \mathbf{D}_i}{|\mathcal{S}|}\|_2^2}{m}}. \tag{4}$$

In the training stage, once a tree t is learned, a set of training samples $S_t^{lf} = \{\mathbf{D}_i^{lf}\}_{i=1}^{|S_t^{lf}|}$ would fall into a particular leaf node lf. Obviously, it is not effective to store all the samples in S_t^{lf} for each leaf node lf. Instead, we would do an abstraction for the learning purpose. One choice is to store the mean $\bar{\mathbf{D}}^{(lf)} = \sum_i \mathbf{D}_i^{lf} / |S_t^{lf}|$ of the set S_t^{lf}. One could store other abstractions such as the histogram of S_t^{lf} as well. In addition, each tree t would assign a leaf node label $L_t(H(ST_i))$ for a given $H(ST_i)$.

In the testing stage, given a test example $ST = (\hat{\mathbf{x}}_j, c_j; j = 1, \ldots, n)$, for each tree t, it starts at the root, then recursively branches left or right. Finally, it reaches the leaf node $L_t(H(ST))$ in tree t, then the prediction given by tree t is $F_t(H(ST)) = \delta(lf = L_t(H(ST))) \cdot \bar{\mathbf{D}}^{(lf)}$, where $\delta(\cdot)$ is an indicator function. The final output of the forest (T trees) is a probability function:

$$P_{H(ST)}(\mathbf{D}) = \frac{1}{T} \sum_{t=1}^{T} \exp(-\|\frac{\mathbf{D} - F_t(H(ST))}{h_D}\|_2^2), \tag{5}$$

where h_D is a learned bandwidth. The mean can be considered as another output of the learned regression function $f(H(ST)) = E_{P_{H(ST)}}[\mathbf{D}]$ where $E_{P_{H(ST)}}[\cdot]$ indicates the expectation. The corrected skeleton is obtained by (if we would use the $f(H(ST))$ as the output)

$$CT = ST^- + \lambda(ST) \cdot f(H(ST)), \tag{6}$$

where $ST^- = (\hat{\mathbf{x}}_j; j = 1, \ldots, n)$ and the components in CT are $CT = (\mathbf{z}_j; j = 1, \ldots, n)$. In the experiments, we refer to this method (using the $f(H(ST))$ as the output) as RFR.

3.3.3 Regression cascade

In the recent work [8], an algorithm named cascaded pose regression (CPR) is proposed, which iteratively trains a series of regressors to approach the ground truth. Inspired by CPR, we propose a regression cascade (RFRC) here.

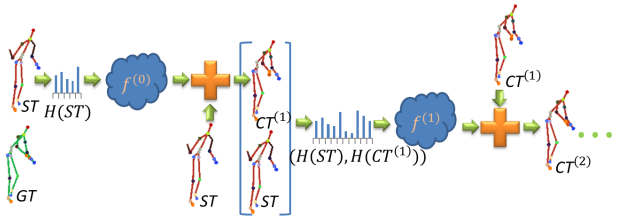

Figure 5. Illustration of the corrected human skeleton updated at each iteration.

As described in Sec. 3.3.2, we learn a regression function $f : H(ST) \rightarrow \mathbf{D}$. Here, we rewrite it as

$$f^{(0)} : H(ST) \rightarrow \mathbf{D}^{(0)}. \tag{7}$$

Then we obtain the corrected skeleton $CT^{(1)}$ by

$$CT^{(1)} = ST^{-} + \lambda(ST) \cdot f^{(0)}(H(ST)). \tag{8}$$

Then we compute the normalized skeleton joint coordinates $H(CT^{(1)})$ and learn the second layer of regression function:

$$f^{(1)} : (H(ST), H(CT^{(1)})) \rightarrow \mathbf{D}^{(1)}, \tag{9}$$

where $\mathbf{D}^{(1)}$ is the offsets between $CT^{(1)}$ and GT computed by the stage in Eq. 3. Then repeat the process mentioned above. The $(i+1)$th layer of regression function is

$$f^{(i)} : (H(ST), H(CT^{(i)})) \rightarrow \mathbf{D}^{(i)}. \tag{10}$$

The output skeleton is

$$CT^{(i+1)} = CT^{(i)} + \lambda(ST) \cdot f^{(i)}(H(ST), H(CT^{(i)})). \tag{11}$$

For consistency, we define $CT^{(0)} = \emptyset$ and obtain

$$CT^{(i+1)} = ST^{-} + \lambda(ST) \cdot \sum_{i} f^{(i)}(H(ST), H(CT^{(i)})). \tag{12}$$

Fig. 5 shows an illustration for the process of the regression cascade.

3.3.4 Temporal constraint

Taking the motion consistency into account, we could use the temporal constraint to improve our correction results. Our learned regression function outputs a probability distribution as shown in Eq. (5), which can be used to employ the temporal information. Given the estimated skeleton sequence $\mathbf{st} = (ST_1, \ldots, ST_m)$, our goal is to obtain $h(\mathbf{st}) \rightarrow \mathbf{d}$, where $\mathbf{d} = (\mathbf{D}_1, \ldots, \mathbf{D}_m)$, with the corresponding corrected skeleton sequence $\mathbf{ct} = (CT_1, \ldots, CT_m)$. To meet the real time requirement, our approach follows a causal model, i.e. the current prediction only depends on past/current inputs/outputs. For the ith input estimated skeleton ST_i, its offset is computed as

$$\mathbf{D}_i = \begin{cases} f(H(ST_i)) & \text{if } i = 1 \\ \arg\min_{\mathbf{D} \in \mathbb{R}^{n \times 3}} E(\mathbf{D}|ST_i, ST_{i-1}, \mathbf{D}_{i-1}) & \text{otherwise} \end{cases}, \tag{13}$$

where $E(\cdot)$ is an energy function defined as

$$E(\mathbf{D}|ST_i, ST_{i-1}, \mathbf{D}_{i-1}) = \alpha \cdot (-\log(P_{H(ST_i)}(\mathbf{D}))) + (1-\alpha)\|ST_i^{-} + \lambda(ST_i)\mathbf{D} - (ST_{i-1}^{-} + \lambda(ST_{i-1})\mathbf{D}_{i-1})\|_2^2, \tag{14}$$

where α is a weight factor. We use coordinate descent to solve Eq. 14. Finally, the corrected skeleton CT_i is

$$CT_i = ST_i^{-} + \lambda(ST_i)\mathbf{D}_i. \tag{15}$$

In the experiments, we refer to the random forest regression and cascade methods under temporal constraint as RFRT and RFRCT respectively.

3.4. Pose tag prediction

In this section we discuss how to learn a regression function to predict the tag of a pose. The learning process is the same as what we did for the skeleton correction; so we follow the notions in Sec. 3.3.2 and 3.3.4 except for replacing the offset $\mathbf{D}$ by the tag value Υ. As stated in the previous section: we are given a training set $\{(\mathbf{st}, \boldsymbol{\Upsilon})_k\}_{k=1}^{K}$ of K pairs of $\mathbf{st}$ and $\boldsymbol{\Upsilon}$ (for denotational simplicity, we let $K = 1$ and thus k can be dropped for easier problem understanding). Using the normalization step in Sec. 3.2, we obtain $h(\mathbf{st}) = (H(ST_1), \ldots, H(ST_m))$, where each $H(ST) = (\mathbf{r}_j, c_j; j = 1, \ldots, n)$. Thus our objective is to obtain $h(\mathbf{st}) \rightarrow \boldsymbol{\Upsilon}$, where $\mathbf{st} = (ST_i; i = 1, \ldots, m)$ and $(\boldsymbol{\Upsilon} = \Upsilon_i; i = 1, \ldots, m)$. Similarly, a random forest regression function to directly predict the tag only based on the individual skeleton $f : H(ST) \rightarrow \Upsilon$ is also learned first. Here, each leaf node in tree t also stores the mean tag values of the samples falling into the leaf node. In the testing stage, given a test example ST, the prediction $F_t(H(ST))$ given by tree t is also computed similarly as in Sec. 3.3.2. The final output of the forest (T trees) is a regression in probability function:

$$P_{H(ST)}(\Upsilon) = \frac{1}{T}\sum_{t=1}^{T} \exp(-\|\frac{\Upsilon - F_t(H(ST))}{h_\Upsilon}\|^2), \tag{16}$$

where h_Υ is a learned bandwidth. The mean is also considered as another output of the learned regression function $f(H(ST)) = E_{P_{H(ST)}}[\Upsilon]$.

A tag is a point on the manifold of the coherent motion, therefore the temporal constraint would be much more useful and effective in predicting the tag value. Our approach for tag prediction is also a cause model. Given the estimated skeleton sequence $\mathbf{st} = (ST_1, \ldots, ST_m)$, the goal is to

obtain the tag sequence $(\Upsilon_1, \ldots, \Upsilon_m)$. For the ith input estimated skeleton ST_i, similarly Υ_i is predicted as:

$$\Upsilon_i = \begin{cases} f(H(ST_i)) & \text{if } i = 1 \\ \arg\min_{\Upsilon \in [0,1]} E(\Upsilon | ST_i, \Upsilon_{i-1}) & \text{otherwise} \end{cases}, \quad (17)$$

where

$$E(\Upsilon | ST_i, \Upsilon_{i-1}) = \alpha \cdot (-\log(P_{H(ST_i)}(\Upsilon))) + (1-\alpha)(\Upsilon - \Upsilon_{i-1})^2, \quad (18)$$

where α is the weight factor. The minimization of the above energy can be done similarly as before.

4. Experimental Results

In this section, we show the experimental results and give the comparisons between alternative approaches, including what is delivered in the current Kinect system. In the remainder of this section, unless otherwise specified, we set the parameters for learning random forest as: the number of trees $T = 50$ and leaf node size $\kappa = 5$. The bandwidths h_D and h_Υ and the weight factor α are optimized on a hold-out validation set by grid search (As an indication, this resulted in $h_D = 0.01$m, $h_\Upsilon = 0.03$ and $\alpha = 0.5$). We set the number of the layers of regression cascade as $L = 2$.

4.1. Joint-based skeleton correction

To evaluate our algorithm, we show the performance on a challenging data set. This data set contains a large number of poses, $15,815$ frames in total, coming from 81 swing motion sequences. Some pose examples are shown in Fig. 3. We select 19 sequences containing $3,720$ frames as the training data set. The rest $12,095$ frames are used for testing.

4.1.1 Error Metrics

Given a testing data set $\{(ST_i, GT_i)\}_{i=1}^m$, we obtain the corrected skeletons $\{CT_i; i = 1, \ldots, m\}$. We measure the accuracy of each corrected skeleton $CT = (\mathbf{z}_j; j = 1, \ldots, n)$ by the sum of joint errors (sJE) $GT = (\mathbf{x}_j; j = 1, \ldots, n)$: $\varepsilon = \sum_j \|\mathbf{z}_j - \mathbf{x}_j\|_2$. To quantify the average accuracy on the whole testing data, we report the mean sJE (msJE) across all testing skeletons: $\sum_i \varepsilon_i / m$ (unit: meter).

4.1.2 Comparisons

In this section, we give the comparison between the methods for joint-based skeleton correction.

Current Kinect approach. To illustrate the difficulty of the problem, we compare with the approach in the current Kinect system. The current Kinect system for skeleton correction is complex, which employs several strategies such as temporal constraint and filtration. The main idea of the approach is nearest neighbor search. For a testing ST, The approach searches its nearest neighbor in the estimated skeletons in the training set. Then the ground truth of the nearest neighbor is scaled with respect to ST to be the corrected skeleton of ST. The details of this system is unpublished. We refer to the current approach in Kinect system as K-SYS in the rest of paper. On the whole testing set, K-SYS achieves 2.0716 msJE, while RFR achieves 1.5866. We illustrate some examples of the corrected skeletons obtained by K-SYS and our algorithm in Fig. 6(b).

Regress the absolute joint position. To show the significance of learning the offsets between joints in ST and GT, we also give the result by directly predicting the absolute joint position by random forest regression (RFRA). To learn the absolute position, for each $GT = (\mathbf{x}_j; j = 1, \ldots, n)$ in the training set $\mathcal{S} = \{(ST_i, GT_i)\}_{i=1}^m$, we translate the C. Hip $\mathbf{x}_1$ to $(0, 0, 0)$ to align them. The absolute joint position of each $GT = (\mathbf{x}_j; j = 1, \ldots, n)$ is obtained by $\tilde{\mathbf{x}}_j = (\mathbf{x}_j - \mathbf{x}_1)/\lambda(GT)$. Then a regression function $\tilde{f} : H(ST) \rightarrow (\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n)$ is learned as the process in Sec. 3.3.2. Given a testing $ST = (\hat{\mathbf{x}}_j; j = 1, \ldots, n)$, the joint positions of the corrected skeleton $CT = (\mathbf{z}_j; j = 1, \ldots, n)$ are obtained by $CT = \lambda(ST)\tilde{f}(H(ST))$. Finally, the C. Hip $\mathbf{z}_1$ of the obtained CT is translated to $\hat{\mathbf{x}}_1$ to align the CT with the ST. As shown in Fig. 6(a), RFRA does not perform as well as RFR.

Other regression algorithms. We also apply other regression algorithms to joint-based skeleton correction, such as Gaussian process regressor [20] (GPR) and support vector regressor [21] (SVR). The implementation of GPR was taken from the GPML toolbox [19], which learns the parameters of the mean and covariance functions of Gaussian processes automatically. GPR achieves 1.7498 msJE. Fig. 6(a) shows the apparent advantage of RFR over GPR. We employ the implementation of SVR taken from the package of LIBSVM [5]. We utilize radial basis function (RBF) kernel for SVR. and obtain 1.6084 msJE. The result obtained by SVR is also worse than RFR. Besides, to train the model of SVR, quite a few parameters need to be tuned.

We illustrate the performances of all the methods mentioned above in Fig. 6(a). The baseline is the msJE of the input estimated skeletons, which is 2.3039 msJE. RFRT, RFRC and RFRCT achieve 1.5831, 1.5766 and 1.5757, respectively. The results demonstrate that both performing in the way of cascade and adding temporal constraint can help improve the performance. Generally, our approach decreases 44% and 16% error with and without occlusion respectively.

4.1.3 Parameter Discussion

We investigate the effects of several training parameters on regression accuracy. As shown in Fig. 7(a), the mean sJE

Figure 6. Comparison with several methods on our testing data set. (a) The quantitative results obtained by several methods. The baseline is the msJE of the input testing estimated skeletons. (b) Examples of the human skeletons. In each example we see the GT, the ST, the CT obtained by K-SYS. and the CT obtained by RFRC. The CTs obtained by RFRC are more similar to the GTs.

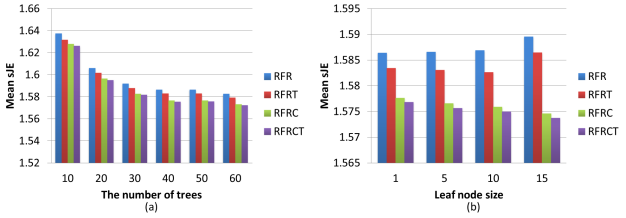

Figure 7. Training parameters *vs.* performance on joint-based skeleton correction. (a) Number of trees. (b) Leaf node size.

decreases as the number of trees increases. We also show the performance by varying the leaf node size. The tree depth is controlled by the leaf node size. The smaller is the leaf node size, the deeper is the tree. As shown in Fig. 7(b), the performance of RFR would decrease when setting larger leaf node size. However, encouragingly, RFRC even obtains better result when setting larger leaf node size.

4.2. Pose tag prediction

To evaluate our method for pose tag prediction, we collect a larger data set totally containing 185 swing motion sequences (mocap data is unnecessary here, so it's convenient to collect more data). We annotate all the sequences and select 37 sequences containing $3,066$ frames as training data; the rest 148 sequences containing $12,846$ frames are used for testing.

4.2.1 Error Metrics

Given the testing data set $\{(ST_i, \Upsilon_i)\}_{i=1}^m$, where Υ_i is the annotated tag of ST_i, we obtain the predicted tags $\{\hat{\Upsilon}_i\}_{i=1}^m$. The tag prediction accuracy on the whole testing data set is quantified by the rooted mean square (RMS) distance: $\sqrt{\Sigma_{i=1}^m(\hat{\Upsilon}_i - \Upsilon_i)^2/m}$.

4.2.2 Comparisons

In this section, we give the comparison between the methods for tag prediction.

Current Kinect approach. The current approach in Kinect system (K-SYS) for tag prediction also contains many details such as imposing temporal constraint and filtering. The main idea of K-SYS is also nearest neighbor search. For a testing ST, K-SYS searchs its nearest neighbor in training set, then the predicted tag of its nearest neighbor is taken as the tag of the ST. K-SYS achieves 0.1376 RMS, which is even better than RFR (0.1406 RMS). This observation indicates the significance of the temporal constraint in tag prediction. Our algorithm RFRT significantly improves R-FR, it achieves 0.1164 RMS. We also compare with K-SYS by varying the size of training data set. We divide our data set into 10 folds with equal sizes, then randomly select N_t folds for training and use the rest for testing. We compute the mean and the standard deviation of the RMS distances obtained by repeating the random selection for 5 times. The results for $N_t = 4, 6, 8$ using 10 trees are shown in Fig. 8(b).

K-nearest neighbors search. To show the advantage of random forest in data abstraction and robust estimation, we compare with K-nearest neighbors search (KNNS). Given a testing sequence $(ST_1, \ldots, ST_m)$, for each $ST_i (i \in (1, \ldots, m))$, K nearest neighbors are searched from the training set, and the tags of the neighbors are obtained: $\boldsymbol{\Upsilon}_K = (\Upsilon_k; k = 1, \ldots, K)$. Then we obtain the probability distribution $P_{H(ST_i)}(\Upsilon) = \frac{1}{K}\sum_{k=1}^K \exp(-\|\frac{\Upsilon - \Upsilon_k)}{h_\Upsilon}\|^2)$. Then considering the temporal consistency, using the method in Sec. 3.4, the optimal value is searched from the distribution $P_{H(ST_i)}(\Upsilon)$ as the predicted tag of ST_i. KNNS only achieves 0.1451 RMS.

Other regression algorithms. We apply GPR and SVR to tag prediction, which achieve 0.1563 RMS and 0.1426 RM-S respectively. The tag is a real value ranging from 0.0 to 1.0, so we also apply logistic regression (LR) [17] to tag prediction. However, it only achieves 0.1806 RMS. Unlike RFR, which benefits from optimization under the temporal constraint, GPR, SVR and LR have no such advantage.

We illustrate the results of tag prediction on our testing data set in Fig. 8(a). Fig. 8(c) shows tag curves of three example sequences. The horizontal and vertical axes of the tag curve are the frame index of the sequence and the tag value, respectively. The curves obtained by RFRT (black) fit the annotated curves (green) best.

Besides the golf swing motion, we also compare RFRT

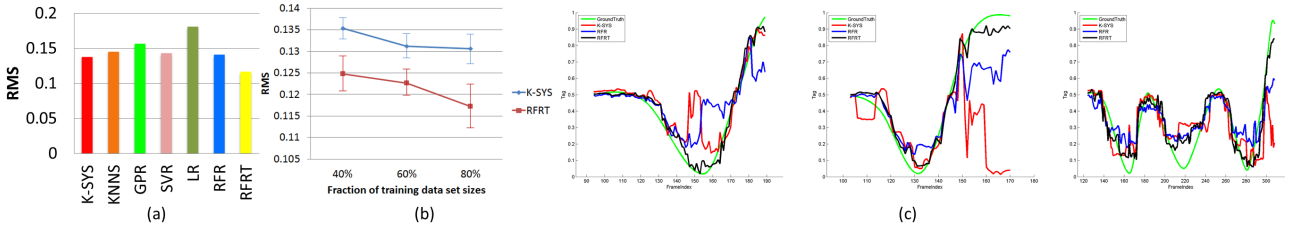

Figure 8. Comparison with several methods on the testing data set. (a) The quantitative results obtained by several methods. (b) Accuracy versus size of training data set (c) The tag curves of three example sequences. In each example, we show the annotated curve (green) and those obtained by K-SYS (red), RFR (blue) and RFRT (black). The yellow curves fit the green best.

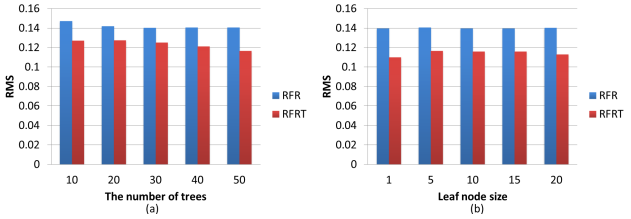

Figure 9. Training parameters *vs.* performance on tag prediction. (a) Number of trees. (b) Leaf node size.

Table 1. The runtime of our approach on Xbox. The runtime is measured by millisecond per frame. For the two tasks, joint-based skeleton correction and tag prediction, we report the runtime of RFR and RFRT, respectively.

Number of trees	10	20	30
Skeleton correction (RFR)	6.6ms	11.8ms	17.2ms
Tag prediction (RFRT)	5.9ms	8.8ms	11.8ms

with K-SYS on a football throwing motion data set. RFRT and K-SYS achieve 0.0791 RMS and 0.0883 RMS respectively. The results for golf swing and football throwing show our approach is applicable to any pose correction.

4.2.3 Parameter Discussion

The effects of the training parameters, including the number of trees and leaf node size, on tag prediction accuracy is shown in Fig. 9. Generally, using more trees and setting smaller leaf node size help improve the performance.

Runtime Our approach is real-time and can be directly embedded into the current Kinect system. We give the runtime of our approach on Xbox in Tab. 1.

5. Conclusion

We have presented a new algorithm for pose correction and tagging from the initially estimated skeletons from Kinect depth images. Our exemplar-based approach serves a promising direction and we highlighted the importance of learning the inhomogeneous systematic bias. Performing cascaded regression and imposing the temporal consistency also improves pose correction. Our experimental results for both pose joints correction and tag prediction show significant improvement over the contemporary systems.

Acknowledgements This work was supported by Microsoft Research Aisa; it also received support from Office of Naval Research Award N000140910099, NSF CAREER award IIS-0844566, National Natural Science Foundation of China #60903096 and #61173120.

References

[1] Microsoft Corp. Kinect for XBOX 360. Redmond WA.

[2] A. Birnbaum. A unified theory of estimation. *The Annals of Mathematical Statistics*, 32(1), 1961.

[3] L. D. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *ICCV*, 2009.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. PAMI*, 25(5):564–575, 2003.

[7] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *ICCV VS-PETS*, 2005.

[8] P. Dollár, P. Welinder, and P. Perona. Cascaded pose regression. In *Proc. CVPR*, 2010.

[9] R. Duda, P. Hart, and D. Stork. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 2000.

[10] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Real-time human pose recognition in parts from a single depth image. In *ICCV*, 2011.

[11] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *Int'l J. of Comp. Vis.*, 29(1):5–28, 1998.

[12] J. Lee and S. Y. Shin. Motion fairing. In *Proceedings of Computer Animation*, pages 136–143, 1996.

[13] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Proc. CVPR*, 2005.

[14] A. Liaw and M. Wiener. Classification and regression by randomforest, 2002.

[15] H. Lou and J. Chai. Example-based human motion denoising. *IEEE Tran. on Visualization and Computer Graphics*, 16(5), 2010.

[16] J. C. Niebles, C.-W. Chen, and L. Fei-Fei. Modeling temporal structure of decomposable motion segments for activity classification. In *ECCV*, 2010.

[17] P. Peduzzi, J. Concato, E. Kemper, T. R. Holford, and A. R. Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *J Clin Epidemiol*, 49(12):1373–1379, 1996.

[18] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1986.

[19] C. E. Rasmussen and H. Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11:3011–3015, 2010. Software available at `http://www.gaussianprocess.org/gpml`.

[20] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[21] B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

[22] 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 a single depth image. In *Proc. CVPR*, 2011.

[23] S. Tak and H.-S. Ko. Physically-based motion retargeting filter. *ACM Transactions on Graphics*, 24(1), 2005.