Human Activity Recognition Approach

Sample Lab Project Report

Name withheld

I. Introduction and objectives

One of the many applications for a smart system is the ability to provide an automated assessment of health. In the current aging population, "the challenges of maintaining mobility and cognitive function make it increasingly difficult to remain living alone therefore forcing many people to seek residence in clinical institutions" [1].

A smart home can be viewed as a possible solution in creating an environment that is capable of monitoring the resident. In a smart home, sensors can be programmed to learn about a resident's normal daily routines which can then be used for performing automated health monitoring and assessment [2]. For example, Pavel et al. [3] suggested that there is a relationship between mobility patterns and cognitive ability. The theory was examined by observing the changes in mobility and found evidence to support the relationship between mobility and cognitive ability.

Lee and Dey [4] also designed an embedded sensing system to determine if the resident gains more awareness about their functional abilities when given information regarding their movements. The ability to perform automated assessment of task quality and cognitive health has greatly improved in performance due to the increased accuracy in identifying the current task at hand [5], [6]. These techniques indicate that specific information can be extracted from a sensor and be used in labelling the activity that is being performed. Some activities such as washing dishes, taking medicine, and using the phone are characterized by interacting with unique objects. A simple solution is to use RFID tags and shimmer sensors to tag these unique objects and use the interaction for activity recognition [7], [8].

The minimum requirement for the creation of the health assessment tool is the ability to detect and recognize human activities. Regarding this point, the main objective of this project is to implement a method for human activity recognition. Traditional activity recognition uses mainly RGB images for analysis. Current methods incorporates different types of information including depth, infrared, and time [9]. The proposed method focuses on using only skeleton point location as a means to performing human activity recognition.

II. Framework

This section describes the proposed approach of using only skeleton points in recognizing a human activity. Fig. 1 illustrates the overall design of the recognition system. The system consists of 4 main components: database, skeleton parser, classifier, and output.

In this experiment, the database chosen must consist of a skeleton frame with the coordinates of the skeleton points. There exists many methods for extracting or computing a skeleton frame from different kinds of data, depth, RGB or 3D, but the Kinect offers a built-in method that automatically provides skeleton information from the hardware itself [10]. In addition, information collected using the Kinect can also provide RGB, depth, and sound information.

The skeleton parser is a necessary component for databases that uses the Kinect and is composed of 2 main elements: the text parser and mapper. The text parser extracts the skeleton information from a encoded text file and the mapper converts the parsed information (in x,y,z camera coordinates) into pixel locations (2D x and y image coordinates).

The classifier is the most fundamental component that exists in any type of recognition systems. The purpose of the classifier is to interpret the data in such a way that it is possible to distinguish the differences between each group/cluster of data.

The output component collects the results generated from the classifier and computes the performance of the system. The system performance can be determined by either measuring the correctness of the recognition algorithm or calculating the error rates of the final results.

A. Databases

The CAD-60 [11] was chosen as the database because this dataset is one of the only databases that uses a Kinect [12] sensor to collect data for human activities. In the CAD-60 [11] dataset there are 12 main activities, 1 baseline activity and 1 random activity. The 12 main activities

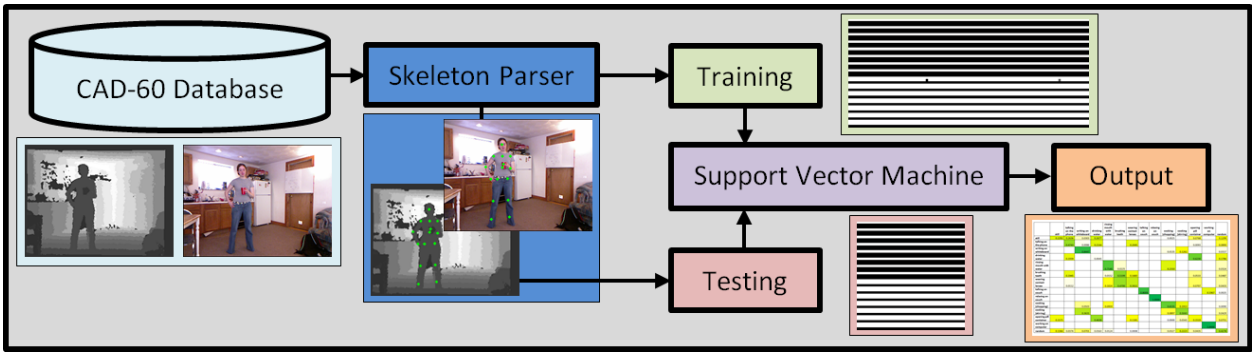


Fig. 1. The overall design of the human activity recognition system.

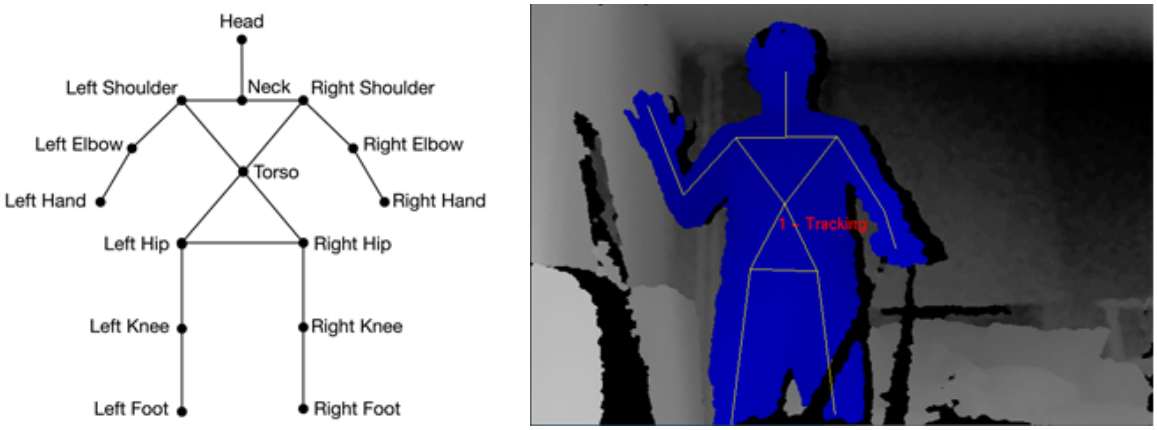
include brushing teeth, stirring, writing, working, talking, wearing contacts, relaxing, opening container, drinking, chopping, talking on chair, and rinsing mouth (shown in Fig. 2). The baseline activity is mainly the neutral position (standing still) and the random activity is the person moving back and forth while stretching.

Fig. 2. The assorted activities in the CAD-60 dataset [11].

B. Skeleton Parser

The Kinect contains a built-in feature that has the capability of detecting and creating a skeleton frame of a person that appears in the scene. The skeleton points of the skeleton frame is provided in a camera coordinate system and can be retrieved through the use of the Kinect SDK [12] or OpenNI [13]. In the CAD-60 [11] dataset, OpenNI [13] was the software used to obtain the skeleton points. One reason OpenNI [13] was chosen for CAD-60 by the authors of [11] because the Kinect SDK [12] was initially not open to public.

Fig. 3(a) shows an image of a skeleton frame containing the 15 skeleton points and Fig. 3(b) illustrates the depth portion of the Kinect [12] sensor with a highlighted individual outlining the skeleton.



(a) (b)

Fig. 3. The skeleton from the Kinect. (a) The obtained skeleton frame from OpenNI [13] containing 15 points; (b) The depth portion of the Kinect with highlighted person including the skeleton.

Fig. 4 outlines the movements of the skeleton points for an individual that is performing the 'drinking water' activity. In this figure, a negative y-axis is used in order to correctly orientate the skeleton points (feet are located at the bottom and head is located at the top). A reflection on the x-axis (negative y-axis) is required to visualize the skeleton points in MatLab because the origin point (0,0) for MatLab is located at the bottom left corner while for imaging softwares (openCV [14], paint) the origin point (0,0) is located at the top left corner. In this individual's 'drinking water' activity, there are a total of 1310 frames and for each frame, there are 45 (15 skeleton points * 3 coordinate values) coordinate points. Observing the movements of the 'drinking water' activity through the 1310 frames will reveal the following patterns of movement:

- Major movement centered around the right arm (right shoulder, right elbow and right hand): a circular motion caused by the repeated task of drinking water,
- Stable/fixed positions of both feet (left foot and right foot),
- Diagonal movement of the left portion (left elbow, left hand and left hip) of the body: When the individual raises the cup to drink, it leads to a deformed skeleton where Kinect attempts to fix by re-adjusting all the skeleton points.

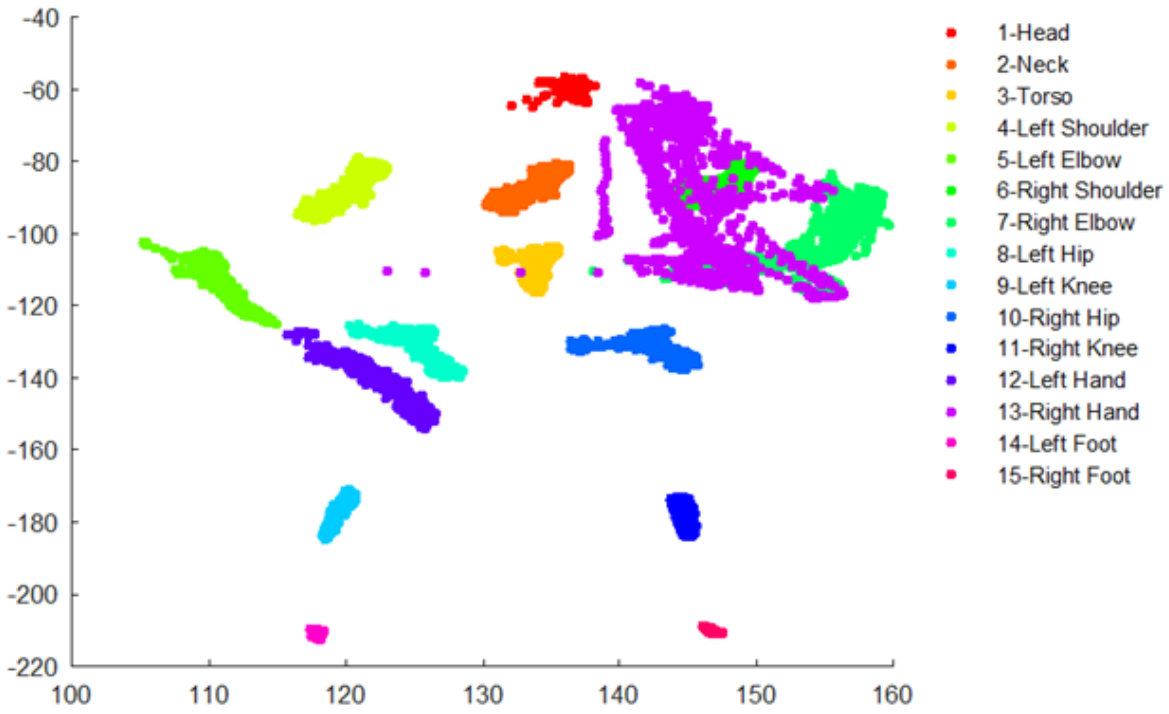


Fig. 4. The movement of the skeleton points of a person performing the 'drinking water' activity.

Note that in this experiment, OpenNI [13] and the CAD-60 [11] dataset will provide the location of the skeleton points but is given in camera space. In camera coordinates, the origin (0,0,0) represents the location of the Kinect, +X (X,0,0) indicate X points to the right of the Kinect, +Y (0,Y,0) specify Y points above the Kinect, and +Z (0,0,Z) mark Z points between the sensor and object. In this system, all values are given in 'points' which loosely translates into distance in millimeters. For example, an object at (-10,20,30) implies the object is 10mm to the left, 20 mm above, and 30 mm away from the Kinect.

In order to illustrate the positioning of the skeleton in a 2D (RGB) image, it is required to convert the camera coordinates into pixel locations. For the CAD-60 [11] dataset the following equations are given to perform the mapping $(x, y, z) \rightarrow (u, v)$:

$$u = 156.8584456124928 + 0.0976862095248 \cdot x - 0.0006444357104 \cdot y + 0.0015715946682 \cdot z \tag{1}$$

$$v = 125.5357201011431 + 0.0002153447766 \cdot x - 0.1184874093530 \cdot y - 0.0022134485957 \cdot z \tag{2}$$

Another method to convert the camera coordinates into pixel locations can be done using the field of view, angles and image resolution. The following parameters are known:

- The images, both RGB and depth, all have a resolution of 320x240;
- When mapping to the image coordinates, the camera coordinate $(0, 0, z)$ will always yield the center of the image (160,120);
- The field of view for the Kinect is given as 21.5 and 28.5 for the vertical and horizontal direction.

Given the parameters, the method of mapping can be achieved as follows: $(x, y, z) \rightarrow (j, k)$:

$$j = 160 + \arctan\left(\frac{x}{z}\right) \cdot \frac{120}{\tan(21.5^\circ)} \tag{3}$$

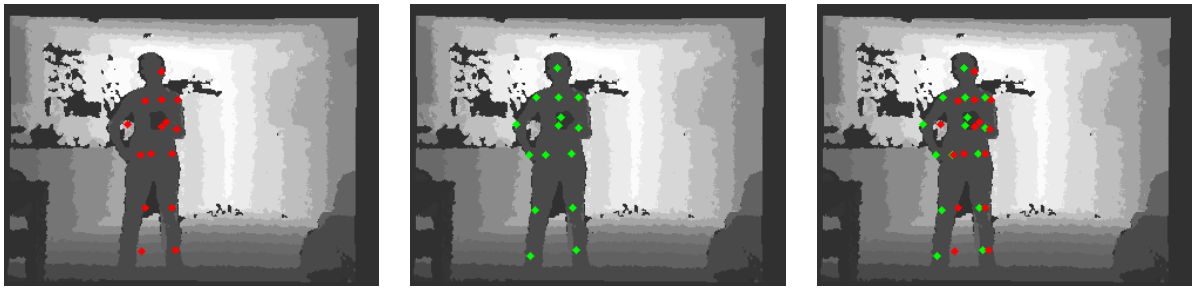
$$k = 120 - \arctan\left(\frac{y}{z}\right) \cdot \frac{160}{\tan(28.5^\circ)} \tag{4}$$

Fig. 5(a) and 6(a) illustrates the mapping equation using Equations 1 and 2. Based on observation, the method of mapping given in the CAD-60 [11] is most likely derived based on the Kinect's field of view. Fig. 5(b) and 6(b) demonstrates the mapping equation using Equations 3 and 4 derived based on the parameters given above.

Both methods of mapping camera coordinates are shown to be capable of generating a skeleton frame that fits with the relative positions of the human body. Howver, contrasting the 2 different methods, the derived method of using angles is visually shown to be a more accurate when the the skeleton is over-layed ontop of the RGB and depth frame (the green dots in Fig. 5 and 6).

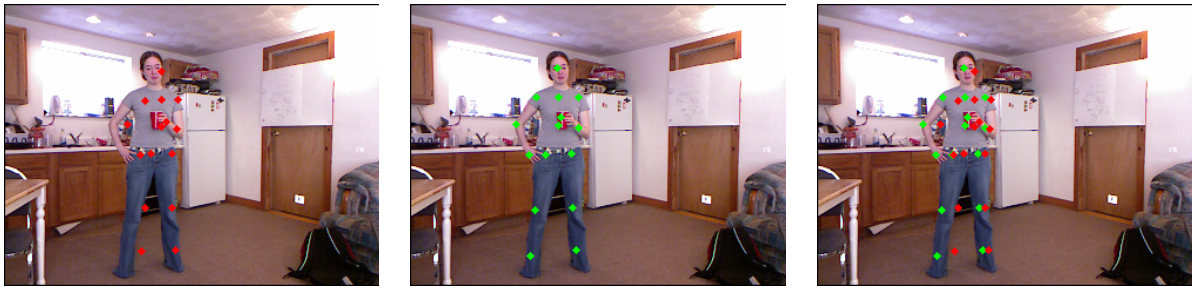
C. Classifier

The training of the activity recognition uses a support vector machine (SVM) classifier. A SVM classifier projects the data onto a higher dimension so that it is possible to divide the data into separable groups [15]. In this experiment, only a linear SVM is used therefore the hyperplane is a straight line that divides the cluster of data (groups) into its own category.



(a) (b) (c)

Fig. 5. The first depth frame of the subject performing the 'drinking water activity'. (a) CAD-60 [11] mapping of skeleton points; (b) the angle-based mapping of skeleton points; (c) both mapping in the same frame.



(a) (b) (c)

Fig. 6. The first RGB frame of the subject performing the 'drinking water activity'. (a) CAD-60 [11] mapping of skeleton points; (b) the angle-based mapping of skeleton points; (c) both mapping in the same frame.

1) Training: The training data is passed to the SVM classifier which is then used to find the approximate hyperplanes that are capable of separating the different human activities. The data used for training is a $Nx45$ matrix where N indicates the total number of frames (number of activities * the number of frames per activity) used for training and 45 is the total number of coordinate points (15 skeleton points * 3 coordinates per point) per frame. Table I shows the number of frames for each activity per subject.

2) Testing: The testing data is also passed to the SVM classifier but instead of using the data to find the hyperplanes, the hyperplanes separates the data into different categories. The separation of data allows for the classification of the human activities.

3) Cross Validation: The statistical analysis is based on the cross validation method which is defined as rotating multiple partitions of the same dataset such that each partition is tested

TABLE I

THE NUMBER OF FRAMES FOR EACH ACTIVITY PER SUBJECT.

Activities	Subject 1	Subject 2	Subject 3	Subject 4	Total
still	482	1029	1119	1102	3732
talking on the phone	830	1525	1288	1308	4951
writing	1637	1792	1597	1792	6818
drinking	778	1587	1310	1529	5204
rinsing	1446	1746	1503	1865	6560
brushing	1675	1351	1783	1580	6389
wearing contacts	1415	835	822	1100	4172
talking on couch	1539	1681	1712	1812	6744
relaxing on couch	1497	1447	1379	1853	6176
chopping	1664	1565	1754	1910	6893
stirring	1349	1346	1467	1835	5997
opening pill container	963	749	621	1012	3345
working on computer	1530	1265	1222	1662	5679
random	1870	1933	1888	1961	7652

individually [16]. In this experiment, the leave-one-out cross validation is adopted therefore the dataset is partitioned into 4 subsets (there are 4 unique subjects). In this method, three subsets are used for training while the last one is used for testing. This process is repeated to a total of 4 times such that each subset has had a chance of being the testing partition. The results of the 4 trials are averaged to generate the statistics and provide the performance measurements of the recognition.

D. Performance Metric

In Table 7, there are 4 different categories: true positive, false negative, false positive, and true negative. The different combinations of each categories can generate a number of different performance metrics [17]. The typical performance measurements are the recall, precision, false acceptance rate (FAR) and false rejection rate (FRR).

$$recall = \frac{TP}{TP + FN}$$

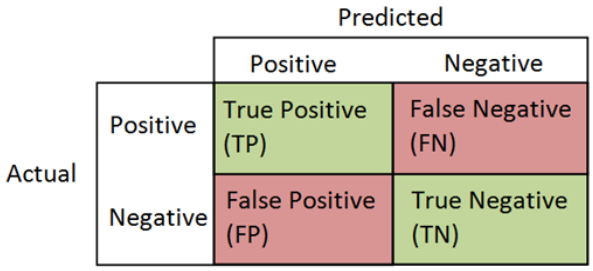


Fig. 7. The 2x2 contingency table for classification.

$$precision = \frac{TP}{TP + FP}$$

$$FAR = \frac{FP}{FP + TN}$$

$$FRR = \frac{FN}{TP + FN}$$

Since all of these metrics are heavily correlated, only one pair of metrics are needed to represent the performance of the system. In this experiment, recall and precision are used for consistency and comparison to [11].

The recall measures the probability of the chosen human activity being correctly identified. For example, given all the 'still' activity, how many of the prediction were correct.

The precision measures the probability of detecting a selected human activity. For example, given a various amount of activities includeing 'still', 'drinking', and etc., what are the chances the algorithm can correctly find the 'still' activity.

III. Experimental Results

Fig. 8 and 9 contains a confusion matrix that illustrates the probabilities of each activity being identified. The sum of the entire row is equal to 1 and the number in each cell represents how often the row activity is identified as the column activity. For example, the 2 unique activities are the 'relaxing on the couch' and 'working on the computer' because they are correctly recognized

	still	talking on the phone	writing on whiteboard	drinking water	rinsing mouth with water	brushing teeth	wearing contact lenses
still	0.2292	0.2636	0.0303	0.2677			
talking on the phone		0.4743	0.0086	0.2144			0.2045
writing on whiteboard			0.8447				
drinking water		0.1939		0.0045			
rinsing mouth with water					0.7140	0.0225	
brushing teeth		0.1945			0.0152	0.5199	0.1683
wearing contact lenses		0.0112			0.1024	0.4700	0.3022
talking on couch							
relaxing on couch							
cooking (chopping)			0.0503		0.0959		
cooking (stirring)			0.3631				
opening pill container	0.1571			0.4636			0.1181
working on computer							
random	0.1366	0.0376	0.0701	0.0163	0.0124		0.0008

Fig. 8. The first portion of the confusion matrix for each activity from the CAD-60 dataset [11].

all the time. The worst case is the ‘drinking water’ activity which is mistaken as the ‘opening pill container’ activity half the time.

An interesting observation in Fig. 8 and 9 is about the ‘drinking water’ and ‘opening pill container’ activities. Between these 2 activites, the matrix indicate inverse relationship. The inverse effect causes most of the ‘drinking water’ activity to be mistaken as ‘opening pill container’ and most of the ‘opening pill container’ activity as ‘drinking water’. A possible explantion for this effect is the similarity between the movement of opening a pill container and drinking a cup of water.

By using the leave-one-out cross validation with the performance metrics, precision and recall, it is possible to calculate the overall performance of the system. Table II displays the performance of various classifiers for human activity recognition. Each row represents the activity in question and each column indicates the performance for each classifier.

	talking on couch	relaxing on couch	cooking (chopping)	cooking (stirring)	opening pill container	working on computer	random
still			0.0025		0.0768		0.1299
talking on the phone					0.0093		0.0890
writing on whiteboard			0.0135	0.1262			0.0157
drinking water					0.6230		0.1786
rinsing mouth with water			0.2310				0.0324
brushing teeth					0.0533		0.0487
wearing contact lenses					0.0707		0.0433
talking on couch	0.8009					0.1967	0.0025
relaxing on couch		1.0000					
cooking (chopping)			0.6535	0.1913			0.0090
cooking (stirring)			0.0897	0.5043			0.0429
opening pill container			0.0000	0.0343	0.1519		0.0751
working on computer						1.0000	
random			0.0327	0.2223	0.0435		0.4278

Fig. 9. The second portion of the confusion matrix for each activity from the CAD-60 dataset [11].

Naive classifier is explained as a simple SVM for baseline comparison. The RGB HOG (histogram of oriented gradient) is a feature descriptor that extracts 32 features based on how often the gradient orientations are seen in a selected region of interest (ROI) in a RGB image [18]. The ROI is found to be a selected portion of the image that contains the subject indicated by a bounding box. The RGBD HOG (histogram of oriented gradient) is the same as RGB HOG but the feature descriptor is applied to both the RGB and depth image. The Skel.+skel HOG is a similar method to the RGBD-HOG but the ROI is based on the location of the head, torso, left arm and right arm indicated by the skeleton points. The skele points method is the proposed method of using only the location/coordinates of the skeleton points as the features for recognition.

Observing Table II, each method is shown to behave differently and not one method performs universally better in all activities. The underlined and bolded rates indicate the highest possible recall and precision for each activity. When looking specifically at the final average, the proposed

TABLE II

THE PERFORMANCE OF ACTIVITY CLASSIFICATION USING DIFFERENT METHODS. *THE AVERAGE DOES NOT INCLUDE 'RANDOM' AND 'STILL' ACTIVITIES.

	naive classifier		RGB HOG		RGBD HOG		Skel.+skel HOG		Skele Points	
	prec.	recall	prec.	recall	prec.	recall	prec.	recall	prec.	recall
brushing teeth	0.645	0.205	0.507	0.308	0.734	0.166	**0.885**	0.553	0.534	**0.584**
cooking (chopping)	0.442	0.293	0.580	0.040	**0.945**	0.111	0.248	0.177	0.747	**0.685**
cooking (stirring)	0.333	0.569	0.561	**0.900**	**0.599**	0.742	0.456	0.433	0.267	0.211
drinking water	0.475	0.247	0.000	0.000	0.240	0.060	**0.717**	**0.713**	0.002	0.002
opening pill container	0.863	0.361	0.721	0.342	0.440	0.358	**0.935**	**0.563**	0.236	0.259
relaxing on couch	0.972	0.764	0.000	0.000	**1.000**	0.215	0.313	0.211	**1.000**	**1.000**
rinsing mouth with water	**0.777**	0.493	0.422	0.733	0.491	**0.973**	0.511	0.514	0.686	0.763
talking on couch	0.715	0.354	0.427	0.594	0.532	0.632	0.732	0.437	**0.877**	**1.000**
talking on the phone	**0.707**	0.215	0.058	0.022	0.164	0.071	0.614	**0.483**	0.309	0.376
wearing contact lenses	**0.820**	**0.897**	0.442	0.406	0.525	0.595	0.786	0.883	0.234	0.397
working on computer	0.935	**0.768**	**1.000**	0.119	1.000	0.290	0.834	0.407	**1.000**	0.751
writing on whiteboard	0.471	0.733	0.412	0.251	**0.940**	**0.970**	0.755	0.813	0.644	0.669
random									0.502	0.645
still									0.058	0.250
average	**0.663**	0.417	0.331	0.235	0.493	0.330	0.679	0.555	*0.545	***0.558**

method of using only skeleton points demonstrates the best recall rate.

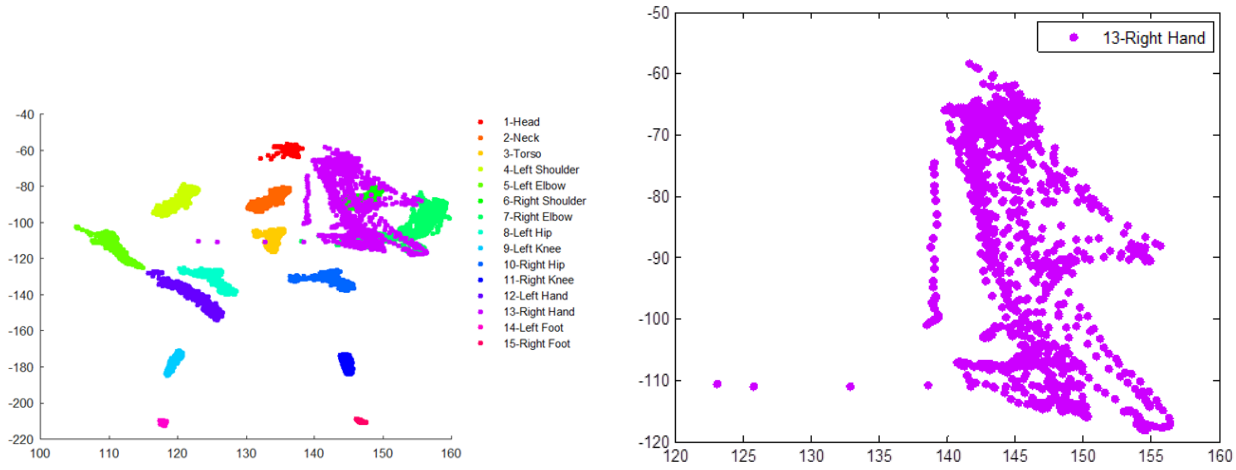
IV. FUTURE WORK

The next step after performing activity recognition is to measure the correctness of the activity, specifically checking if there are abnormal movements during the activity. One possible way of detecting abnormality can be derived based on movement and duration of the activity.

For example, given the skeleton points for when a person is 'drinking water' (Fig. 10(a)), is it possible to detect patterns for a repeated cycle of movements. Since the main movement locations of the 'drinking water' activity is focused on the right arm, the most interesting skeleton point is the right hand. After magnifying the movements of the right hand (shown in Fig. 10(b)), it is possible to remotely see a repeated motion of up and down movements.

Due to the circular movements it is difficult to isolate the cycle duration. A better analysis is to separate the circular movements into x and y direction. Fig. 11 illustrates the movement of

the right hand in the x and y directions. From this plot, the cycle duration can be calculating using the movements of the y direction. A method of approximating the cycle duration is to marke the minimum value for each valley then measure the distance between each minimum.



(a) (b)

Fig. 10. (a) A plot of the skeleton points of a subject performing the 'Drinking Water Activity'. (b) The skeleton point movement for the right hand

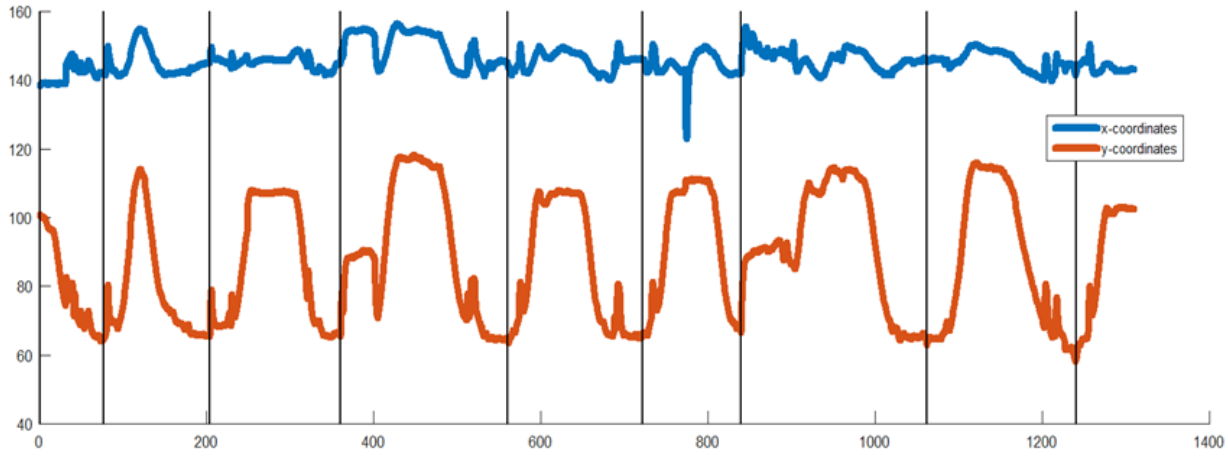


Fig. 11. The x and y-coordinates plots of the right hand movement for the 'Drinking Water' activity.

Table III shows the measured duration in frames for each cycle of the 'Drinking Water' activity.

From this table, cycle 4 and 7 take much longer time to complete. Analysis should be focused on these 2 cycles to find the reason for the extended duration.

TABLE III

THE DURATION OF EACH CYCLE FOR THE 'DRINKING WATER' ACTIVITY.

	1	2	3	4	5	6	7	8	Average
Duration	77	127	156	**200**	161	119	**222**	179	155.125

V. CONCLUSIONS

In conclusion, a feasibility test is performed to check if the method of using only skeleton points for activity recognition is valid. The test shows a precisions of 54.5% and a recall rate of 55.8% which is comparable to the method that uses RGB, depth and skeleton information. Possible benefits of using only skeleton information include:

- unidentifiable features: skeleton points are non-unique therefore very difficult to find the identity associated with the skeleton frame;
- reduced number of features: only 45 coordinate points are required as opposed to 256 features for the skele skel.+skel HOG method.

REFERENCES

[1] A. Arcelus, M. H. Jones, R. Goubran, and F. Knoefel, "Integration of smart home technologies in a health monitoring system for the elderly," in *21st International Conference on Advanced Information Networking and Applications Workshops*, vol. 2, 2007, pp. 820–825.

[2] P. N. Dawadi, D. J. Cook, and M. Schmitter-Edgecombe, "Automated cognitive health assessment using smart home monitoring of complex tasks," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 43, no. 6, pp. 1302–1313, 2013.

[3] M. Pavel, A. Adami, M. Morris, J. Lundell, T. Hayes, H. Jimison, and J. Kaye, "Mobility assessment using event-related responses," in *1st Transdisciplinary Conference on Distributed Diagnosis and Home Healthcare*. IEEE, 2006, pp. 71–74.

[4] M. L. Lee and A. K. Dey, "Embedded assessment of aging adults: a concept validation with stakeholders," in *4th International Conference on Pervasive Computing Technologies for Healthcare*. IEEE, 2010, pp. 1–8.

[5] D. J. Cook, "Learning setting-generalized activity models for smart spaces," *IEEE intelligent systems*, vol. 2010, no. 99, p. 1, 2010.

[6] E. Kim, S. Helal, and D. Cook, "Human activity recognition and pattern discovery," *IEEE Pervasive Computing*, vol. 9, no. 1, pp. 48–53, 2010.

[7] P. Palmes, H. K. Pung, T. Gu, W. Xue, and S. Chen, "Object relevance weight pattern mining for activity recognition and segmentation," *Pervasive and Mobile Computing*, vol. 6, no. 1, pp. 43–57, 2010.

[8] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. Kautz, and D. Hahnel, "Inferring activities from interactions with objects," *IEEE pervasive computing*, vol. 3, no. 4, pp. 50–57, 2004.

[9] H. S. Koppula, R. Gupta, and A. Saxena, "Learning human activities and object affordances from rgb-d videos," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 951–970, 2013.

[10] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," *IEEE MultiMedia*, vol. 19, pp. 4–10, 2012.

[11] J. Sung, C. Ponce, B. Selman, and A. Saxena, "Unstructured human activity detection from rgbd images," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 842–849.

[12] Microsft, "Kinect for Windows Sensor Components and Specifications," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131033.aspx

[13] "OpenNI SDK," http://www.openni.org/openni-sdk, [Online].

[14] "OpenCV 2.4.7.0 documentation," http://docs.opencv.org/index.html, 2013, [Online].

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[16] S. Geisser, *Predictive inference*. CRC press, 1993, vol. 55.

[17] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427 – 437, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306457309000259

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.

APPENDIX

```
#include <iostream>
#include <fstream>
#include <opencv2\opencv.hpp>
#include <string>
#include <math.h>
#include "opencv2\legacy\legacy.hpp"
#include <windows.h>
#include <cstdlib>
#include <iomanip>
using namespace std;
using namespace cv;

//structure for the joint orientation

typedef struct JointOri{
	double m[9]; // 3x3 matrix
	double conf; // confidence value
};

//structure for the joint position
typedef struct JointPos{
	double x; //x coordinate
	double y; //y coordinate
	double z; //z coordinate
	double conf; //confidence value
};

//structure for a frame that contains the skeleton information
typedef struct Frame{
	int joints; //number of joints (15)
	JointOri ori[11]; //joint orientations (11)
	JointPos jointposition[15]; //joint positions (15)
	int num; //frame number
};
//structure for each activity
typedef struct ActivityLabel{
	int actionCode; //the code for each activity, different for each subject
	string actionName; //the name of the activity
	float act;
};
//structure to store all the activities
typedef struct Task{
	Frame *fr; //the frame information for the task
	ActivityLabel action; //activity information for the task
	int frameSize; //number of total frames
};

//structure to store the relation between name of activity and the code for each activity
typedef struct Person{
	ActivityLabel * lbl; //activity label information
	int numTask; //number of task per person
};

//number of subjects
const int numPerson=4;

//store information based on number of subjects in database
Person *per = new Person[numPerson];
```

```
//look at directory to count number of files
int countFiles(string path){

	int fileCounter=-1;
	WIN32_FIND_DATA FindData;
	HANDLE hFind;
	string pathy = string(path.begin(), path.end());
	string stemp = pathy + "*";
	LPCSTR sw = stemp.c_str();

	hFind = FindFirstFile(sw, &FindData);

	while (FindNextFile(hFind, &FindData))
	{
		fileCounter++;
	}
	return fileCounter;
}

//parses each skeleton text file
//obtains the 15 joint positions, 11 orientations for each frame
void loadDatabase(string path, Task * task){

	WIN32_FIND_DATA FindData;
	HANDLE hFind;
	string pathy = string(path.begin(), path.end());
	string stemp = pathy + "*";
	LPCSTR sw = stemp.c_str();

	hFind = FindFirstFile(sw, &FindData);
	int actcounter=0;

	//obtain skeleton information for each activity for each person in the folder
	///each activity is one text file and each person can perform multiple activities
	while (FindNextFile(hFind, &FindData))
	{
		//create the url for the skeleton text file
		string url=FindData.cFileName;
		cout << url << endl;
		if((url.compare("..")<=0))
			continue;

		ostringstream ss3;
		ss3 << path << "\\"<<url;
		string comb3= ss3.str();

		//opens the skeleton file
		ifstream myfile(comb3);
		string line;

		//count the number of lines in skeleton file
		//the line # indicates the number of frames for the activity
		int counter=0;
		while(getline(myfile,line)){
			counter++;
		}
```

```
		myfile.close();
		myfile.open(comb3);
		line="";

		//frame structure to store all the frames
		Frame *fr = new Frame[counter];

		int framecounter=0;

		//parse each lne of the skeleton text file
		while(getline(myfile,line)){

			if(line.compare("END")==0)
				break;

			//obtain the frame number
			int end=line.find(',');
			fr[framecounter].num= stoi(line.substr(0,end));
			line=line.substr(end+1, line.length());
			fr[framecounter].joints=0;

			int temp=fr[framecounter].joints;

			//obtain the skeleton joint information
			while(temp<15){
				int orimc=0;

				//obtain the joint orientation
				if(temp<11){
					while(1){
						end=line.find(',');
						fr[framecounter].ori[temp].m[orimc]=
stod(line.substr(0,end));

						line=line.substr(end+1, line.length());
						orimc++;

						if(orimc>=9)
							break;
					}

					//obtain the joint orientation confidence
					end=line.find(',');

	fr[framecounter].ori[temp].conf=stod(line.substr(0,end));
					line=line.substr(end+1, line.length());
				}

				//obtain x coordinate
				end=line.find(',');
				fr[framecounter].jointposition[temp].x=
stod(line.substr(0,end));
				line=line.substr(end+1, line.length());

				//obtain y coordinate
				end=line.find(',');
				fr[framecounter].jointposition[temp].y=
stod(line.substr(0,end));
```

```
                        line=line.substr(end+1, line.length());

                        //obtain z coordinate
                        end=line.find(',');
                        fr[framecounter].jointposition[temp].z=
stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        //obtain confidence in joint location
                        end=line.find(',');
                        fr[framecounter].jointposition[temp].conf=
stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        temp++;
                }
                fr[framecounter].joints=temp;
                framecounter++;

        }
        myfile.close();
        cout << framecounter << endl;

        //save the skeleton information for this activity
        task[actcounter].fr=fr;
        string tasknum= url.substr(0,url.length()-4);
        task[actcounter].action.actionCode=atoi(tasknum.c_str());
        task[actcounter].frameSize=framecounter;

        for(int x=0; x<numPerson; x++){
                for(int y=0; y<per[x].numTask; y++){
                        if( per[x].lbl[y].actionCode==
task[actcounter].action.actionCode){
                                task[actcounter].action.act=per[x].lbl[y].act;

        task[actcounter].action.actionName=per[x].lbl[y].actionName;
                        }
                }
        }
        actcounter++;
    }
}

//read in information that shows the relationship between activity code and activity name
void loadLabel(string path){

    WIN32_FIND_DATA FindData;
    HANDLE hFind;
    string pathy = string(path.begin(), path.end());
    string stemp = pathy + "*";
    LPCSTR sw = stemp.c_str();

    hFind = FindFirstFile(sw, &FindData);

    while (FindNextFile(hFind, &FindData))
    {
        string url=FindData.cFileName;
        cout << url << endl;
```

```
		if((url.compare("..")<=0))
			continue;

		ostringstream ss3;
		ss3 << path << "\\"<<url;
		string comb3= ss3.str();
		ifstream myfile(comb3);

		string line;
		int counter=0; //number of activities
		while(getline(myfile,line)){
			if(line.compare("END")==0)
				break;
			counter++;
		}

		myfile.close();
		myfile.open(comb3);
		line="";
		int cc=0;
		ActivityLabel *actlbl = new ActivityLabel[counter];
		while(getline(myfile,line)){

			if(line.compare("END")==0)
				break;

			int end=line.find(',');
			actlbl[cc].actionCode= stoi(line.substr(0,end));

			line=line.substr(end+1, line.length());

			end=line.find(',');
			actlbl[cc].actionName= line.substr(0,end);
			actlbl[cc].act=cc;
			cc++;

		}

		int person=atoi(url.substr(url.length()-5,1).c_str());
		if(person==1){
			per[0].lbl=actlbl;
			per[0].numTask=counter;
		}
		else if(person==2){
			per[1].lbl=actlbl;
			per[1].numTask=counter;
		}
		else if(person==3){
			per[2].lbl=actlbl;
			per[2].numTask=counter;
		}
		else if(person==4){
			per[3].lbl=actlbl;
			per[3].numTask=counter;
		}
	}
}
```

```
//temporary function to show the movement of the skeleton points in each frame
void tfunc(){
	string
url="C:\\Users\\Kenneth\\Documents\\skeleton\\activity\\skele\\train\\0511121954";

	ostringstream ss3;
	int count=1;

	ss3 << url<< ".txt";
	string comb3= ss3.str();

	ifstream myfile(comb3);

	string line;

	int counter=0;
	while(getline(myfile,line)){
		counter++;
	}

	myfile.close();
	myfile.open(comb3);
	line="";

	Frame *fr = new Frame[counter];

	int framecounter=0;
	while(getline(myfile,line)){
		if(line.compare("END")==0)
			break;

		int end=line.find(',');
		fr[framecounter].num= stoi(line.substr(0,end));
		line=line.substr(end+1, line.length());
		fr[framecounter].joints=0;

		int temp=fr[framecounter].joints;

		while(temp<15){
			int orimc=0;

			if(temp<11){
				while(1){
					end=line.find(',');
					fr[framecounter].ori[temp].m[orimc]=
stod(line.substr(0,end));
					line=line.substr(end+1, line.length());
					orimc++;

					if(orimc>=9)
						break;
				}

				end=line.find(',');
				fr[framecounter].ori[temp].conf=stod(line.substr(0,end));
				line=line.substr(end+1, line.length());
			}
```

```
                        end=line.find(',');
                        fr[framecounter].jointposition[temp].x= stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        end=line.find(',');
                        fr[framecounter].jointposition[temp].y= stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        end=line.find(',');
                        fr[framecounter].jointposition[temp].z= stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        end=line.find(',');
                        fr[framecounter].jointposition[temp].conf= stod(line.substr(0,end));
                        line=line.substr(end+1, line.length());

                        temp++;
                }
                fr[framecounter].joints=temp;

                framecounter++;
        }

framecounter=0;

Mat cimg=imread("test1.png",1);
imshow("abc",cimg);

int cpt=0;
        while(1)
        {
                double cl[3];
                cl[0]=rand()%255;
                cl[1]=rand()%255;
                cl[2]=rand()%255;
                for(int i=0; i<15; i++){
                        double
alpha=fr[framecounter].jointposition[i].x/fr[framecounter].jointposition[i].z;
                        alpha=156.8584456124928 + 0.0976862095248 *
fr[framecounter].jointposition[i].x - 0.0006444357104 *
fr[framecounter].jointposition[i].y + 0.0015715946682 *
fr[framecounter].jointposition[i].z;
                        double
beta=fr[framecounter].jointposition[i].y/fr[framecounter].jointposition[i].z;
                        beta = 125.5357201011431 + 0.0002153447766 *
fr[framecounter].jointposition[i].x - 0.1184874093530 *
fr[framecounter].jointposition[i].y - 0.0022134485957 *
fr[framecounter].jointposition[i].z;
                        circle(cimg, Point(alpha,beta),1,Scalar(cl[0], cl[1], cl[2]),3);
                        fd << i << " " << alpha << " " << beta <<" ";
                }

                int key = cvWaitKey(10);
                imshow("abc",cimg);

                if(key==' ')
                        break;
```

```
			framecounter++;
			if(framecounter>=counter)
				break;
		}
}
int main(){
		//tfunc();

		//training information
		Task *task;
		//testing information
		Task *testTask;

		//number of files in training folder
		int fileCounter;
		//number of files in testing folder
		int testCounter;

		//obtain activity name and code information
		loadLabel("C:\\Users\\Kenneth\\Documents\\skeleton\\activity\\label\\");

		//obtain training information
		fileCounter=countFiles("C:\\Users\\Kenneth\\Documents\\skeleton\\activity\\skele\\
train\\");
		task=new Task[fileCounter];
		loadDatabase("C:\\Users\\Kenneth\\Documents\\skeleton\\activity\\skele\\train\\",t
ask);

		//obtain testing information
		testCounter=countFiles("C:\\Kenneth\\Documents\\Dropbox\\skeleton\\activity\\skele
\\test\\");
		testTask=new Task[testCounter];
		loadDatabase("C:\\Users\\Kenneth\\Documents\\skeleton\\activity\\skele\\test\\",te
stTask);

		//start svm training
		cout << "Start svm training" << endl;
	    int rowsSize = 0;
	    int trainingArea = 3*15;

		//find the number of frames in training database
		for(int numtask=0; numtask< fileCounter ; numtask++){
			rowsSize+=task[numtask].frameSize;
		}

		//create N*45 matrix for training
		//N is the total number of frames (number of activities * number of frames per
activity)
		//45 coordinate points (15*x,y,z coordinate)
		Mat trainingMat = Mat::zeros(rowsSize, trainingArea, CV_32FC1);

		//populate information in training matrix using the x,y,z joint positions
		int counter1=0;
		int counter2=0;
		for(int numtask=0; numtask< fileCounter ; numtask++){
			for(int numtaskperframe=0; numtaskperframe <  task[numtask].frameSize;
numtaskperframe++){
				for(int skelpos=0; skelpos<15; skelpos++){
```

```
                        trainingMat.at<float>(counter1, counter2*3)=
task[numtask].fr[numtaskperframe].jointposition[skelpos].x;
                        trainingMat.at<float>(counter1, counter2*3+1)=
task[numtask].fr[numtaskperframe].jointposition[skelpos].y;
                        trainingMat.at<float>(counter1, counter2*3+2)=
task[numtask].fr[numtaskperframe].jointposition[skelpos].z;
                        counter2++;
                    }
                    counter1++;
                    counter2=0;
             }
      }

      //label the training matrix where each row is the name of the activity
    Mat matLabels = Mat::zeros(rowsSize,1,CV_32FC1);
      counter1=0;
      for(int numtask=0; numtask< fileCounter ; numtask++){
            for(int numtaskperframe=0; numtaskperframe <  task[numtask].frameSize;
numtaskperframe++){
                    matLabels.at<float>(counter1,0) = task[numtask].action.act;
                    counter1++;
             }
      }

      //svm parameters
      CvSVMParams params;
    params.svm_type    = CvSVM::C_SVC;
    params.kernel_type = CvSVM::LINEAR;

      params.degree = 0; // for poly
      params.gamma = 20; // for poly/rbf/sigmoid
      params.coef0 = 0; // for poly/sigmoid

      params.C = c; // for CV_SVM_C_SVC, CV_SVM_EPS_SVR and CV_SVM_NU_SVR
      params.nu = 0.5; // for CV_SVM_NU_SVC, CV_SVM_ONE_CLASS, and CV_SVM_NU_SVR
      params.p = 0.0; // for CV_SVM_EPS_SVR

      params.class_weights = NULL; // for CV_SVM_C_SVC
      params.term_crit.type = CV_TERMCRIT_ITER +CV_TERMCRIT_EPS;
      params.term_crit.max_iter = 1000;
      params.term_crit.epsilon = 1e-6;

      // Train the SVM
      CvSVM SVM;
      SVM.train(trainingMat, matLabels, Mat(), Mat(), params);

      //create testing matrix
      Mat testingMat = Mat::zeros(testSize, trainingArea, CV_32FC1);

      //populate testing matrix
      counter1=0;
      counter2=0;
      for(int numtask=0; numtask< testCounter ; numtask++){
            for(int numtaskperframe=0; numtaskperframe <  testTask[numtask].frameSize;
numtaskperframe++){
                    for(int skelpos=0; skelpos<15; skelpos++){
                        testingMat.at<float>(counter1, counter2*3)=
testTask[numtask].fr[numtaskperframe].jointposition[skelpos].x;
```

```
                        testingMat.at<float>(counter1, counter2*3+1)=
testTask[numtask].fr[numtaskperframe].jointposition[skelpos].y;
                        testingMat.at<float>(counter1, counter2*3+2)=
testTask[numtask].fr[numtaskperframe].jointposition[skelpos].z;
                        counter2++;
                    }
                    counter1++;
                    counter2=0;
                }
        }

        //create ground truth labels for testing matrix
        Mat testLabels = Mat::zeros(testSize,1,CV_32FC1);
        counter1=0;
        for(int numtask=0; numtask< testCounter ; numtask++){
                for(int numtaskperframe=0; numtaskperframe <  testTask[numtask].frameSize;
numtaskperframe++){
                    testLabels.at<float>(counter1,0) = testTask[numtask].action.act;
                    counter1++;
                }
        }

        //perform the prediction of test results
        Mat resultMat= Mat::zeros(testSize, 1, CV_32FC1);
        SVM.predict(testingMat, resultMat);

        //calculate the performance of prediction
        int acc=0;
        ofstream f("output.txt");
        ofstream g("outputnum.txt");
        for(int i=0; i< testSize; i++){
                g << fixed<< setprecision(0) <<resultMat.at<float>(i,0) << " "<<
testLabels.at<float>(i,0) << endl;
                string cmp1, cmp2;
                int flaggy=0;
                for(int k=0; k<numPerson; k++){
                        for(int j=0; j< per[k].numTask; j++){
                                float t1= resultMat.at<float>(i,0);
                                float t2= float(per[k].lbl[j].act);
                                if(t1==t2){
                                        cmp1=per[k].lbl[j].actionName;
                                        flaggy=1;
                                        break;
                                }
                        }
                        if(flaggy==1)
                                break;
                }

                flaggy=0;
                for(int k=0; k<numPerson; k++){
                        for(int j=0; j< per[k].numTask; j++){
                                float t1= testLabels.at<float>(i,0);
                                float t2= float(per[k].lbl[j].act);
                                if(t1==t2){
                                        cmp2=per[k].lbl[j].actionName;
                                        flaggy=1;
                                        break;
```

```
                        }
                    }
                    if(flaggy==1)
                        break;
                }
                f << cmp1<< " " << cmp2 << endl;
                if(cmp1=="" || cmp2=="")
                    continue;
                if(resultMat.at<float>(i,0) == testLabels.at<float>(i,0)){
                    acc++;
                }
            }
            f.close();
            g.close();

            //output performance to console
            cout << "Correct Guess: "  << acc << endl;
            cout << "Total Count: "  << testSize << endl;
            cout << "accuracy: " <<setprecision(9) << (double)acc/(double)testSize << endl;

            return 0;
}
```