Gesture Recognition with Leap Motion Controller: Image Analysis and Computer Vision II Project Report

Niharika Balachandra
University of Illinois at Chicago
Email: nbalac2@uic.edu
UIN: 668084671

Sharath Kalkur
University of Illinois at Chicago
Email: skalku2@uic.edu
UIN: 655691957

***Abstract*—This paper investigates gesture recognition techniques using Leap Motion Controller to collect 3 dimensional hand features such as coordinates, position of the hand, etc. We then use this raw data to train a Long Short Term Memory Recurrent Neural Network that is able to recognize basic hand gestures.**

***Index Terms*—Hand gesture recognition, Leap Motion Controller, LSTM-RNN, Recurrent neural networks, computer vision, health care, hidden Markov models, learning (artificial intelligence), 3D movements, computer assisted-interface, computer-vision-assisted contact less methodology, gesture sequence classification, isolated gesture classification, isolated gesture recognition, supervised learning, Computer vision, Feature extraction, Sensors, Training, Finger tracking, gesture recognition, humancomputer interface, humancomputer interaction, palm tracking**

I. Introduction

This paper investigates gesture recognition techniques using Leap Motion Controller. We came across a wide range of applications for gesture recognition that prompted us to develop a gesture recognition model. These include:

a) Home Automation: With the advent of home automation systems like Amazon Echo and Google home, hand gesture recognition could be the next step to enabling home automation.

b) Rehabilitation: To facilitate home-based palm and finger rehabilitation in the absence of experts, Leap motion controller can be interfaced with a computing device to record parameters describing 3-D movements of the palm of a user undergoing rehabilitation.

c) Assisted Living: Hand Gesture Recognition can be used as a signaling mechanism to ensure the elderly receive better care by alerting caregivers of specific patient needs communicated via gestures captured using cameras.

d) Health Care: Hand Gesture Recognition could be used to read specific movements so a sterile environment can be maintained without touching a screen, keyboard, or mouse while reading digital medical images during a medical procedure.

e) Automotive Sector: Hand Gesture Recognition can be used in self driving cars to drive using gestures. It could also be applied in navigation systems to control the course the self driving car takes to reach a particular destination

f) Gaming and Consumer Electronics Sector: Hand Gesture Recognition finds wide application in the gaming industry and is widely used with technology such as the Microsoft Xbox or Microsoft Kinect. Smart televisions today are capable of gesture recognition too paving way for more innovation in this field

g) Human-Computer Interaction (HCI): Hand Gesture Recognition can be used in a plethora of Human-Computer Interaction applications. From performing simple tasks such as unlocking smart phones, to controlling devices, and in virtual reality applications etc.

A. Gesture Recognition Methods

Paper [18] discusses various approaches to interpret a gesture based on the type on the input data. Most of these techniques rely on key pointers represented in a 3D coordinate system. Based on the relative motion of these, the gesture can be detected with a high accuracy.

In order to interpret movements of the body, one has to classify them according to common properties and the message the movements may express. For example, in sign language each gesture represents a word or phrase.

Paper [18] discusses two different approaches in gesture recognition:

1) 3D Model Based Gesture Recognition: The foremost method makes use of 3D information of key elements of the body parts in order to obtain several important parameters, like palm position or joint angles

The 3D model approach can use volumetric or skeletal models, or even a combination of the two. Volumetric approaches have been heavily used in computer animation industry and for computer vision purposes. The models are generally created from complicated 3D surfaces, like NURBS or polygon meshes.

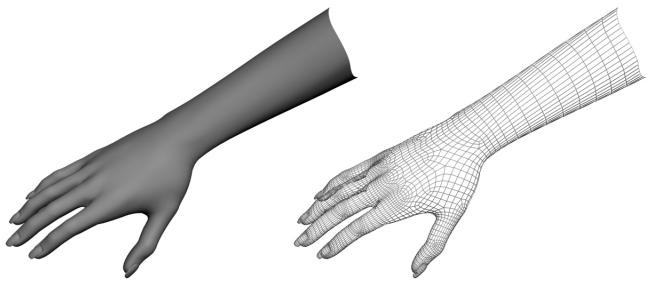


Fig. 1: A real hand (left) is interpreted as a collection of vertices and lines in the 3D mesh version (right), and the software uses their relative position and interaction in order to infer the gesture.

The drawback of this method is that is very computational intensive, and systems for real time analysis are still to be developed. For the moment, a more interesting approach would be to map simple primitive objects to the person's most important body parts (for example cylinders for the arms and neck, sphere for the head) and analyze the way these interact with each other.

Furthermore, some abstract structures like superquadrics and generalized cylinders may be even more suitable for approximating the body parts. The exciting thing about this approach is that the parameters for these objects are quite simple. In order to better model the relation between these, we make use of constraints and hierarchies between our objects.

a) Skeletal-Based Gesture Recognition: Instead of using intensive processing of the 3D models and dealing with a lot of parameters, one can just use a simplified version of joint angle parameters along with segment lengths. This is known as a skeletal representation of the body, where a virtual skeleton of the person is computed and parts of the body are mapped to certain segments. The analysis here is done using the position and orientation of these segments and the relation between each one of them(for example the angle between the joints and the relative position or orientation) Advantages of using skeletal models:

- Algorithms are faster because only key parameters are analyzed.
- Pattern matching against a template database is possible
- Using key points allows the detection program to focus on the significant parts of the body

2) Appearance-Based Gesture Recognition: These models do not use a spatial representation of the body anymore, because they derive the parameters directly from the images or videos using a template database. Some are based on the deformable 2D templates of the human parts of the body, particularly hands. Deformable templates are sets of points on the outline of an object, used as interpolation nodes for the object's outline approximation.

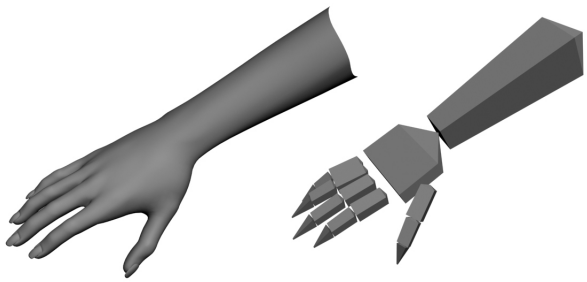


Fig. 2: The skeletal version (right) is effectively modeling the hand (left). This has fewer parameters than the volumetric version and it's easier to compute, making it suitable for real-time gesture analysis systems.

One of the simplest interpolation function is linear, which performs an average shape from point sets, point variability parameters and external deformators. These template-based models are mostly used for hand-tracking, but could also be of use for simple gesture classification.

A second approach in gesture detecting using appearance-based models uses image sequences as gesture templates. Parameters for this method are either the images themselves, or certain features derived from these. Most of the time, only one (monoscopic) or two (stereoscopic) views are used.

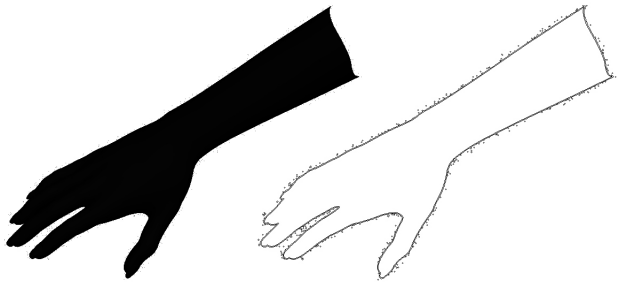


Fig. 3: These binary silhouette(left) or contour(right) images represent typical input for appearance-based algorithms. They are compared with different hand templates and if they match, the correspondent gesture is inferred.

The Leap Motion Controller makes use of the Skeletal-based gesture recognition technique to calculate the necessary features of the hand performing a gesture.

B. Dynamic Time Series Analysis on Continuous Data

Our data set consists of gestures each captured as a 60 frame video. Our gesture recognition model is therefore, analyzing continuous time series data.

It is however, important to realize that comparing time series data is much more complex than comparing non-temporal data, because even if sequences have similar shapes, they are not aligned in a one dimensional axis. [16] We therefore rely on the Leap Motion Controller to extract essential features that allow us to model the gesture under consideration in a 3 dimensional space.

II. Data Collection and Wrangling

A. Role of Leap Motion Controller

The Leap Motion Controller is a sophisticated piece of hardware which is used as a Camera Device using simple, but fast USB Interface. The device is specifically built to detect and track hands and yield a plethora of advanced data about various features of the same. It is comprised of two, monochromatic IR cameras with 3 IR LEDs for illumination in low light. The Leap is capable to capture and extract frame information and transmit it to the device at a maximum of 115 FPS due to the presence of the Cypress EZ-USB [2] chip.

The Leap Motion Controller works alongside the Leap SDK for extracting the features. Few of the features given out by the Leap Motion are the distinction between the right and left hands, the 3D coordinates of their palms, the yaw, pitch, and roll of the two hands, extended fingers count, frame data, etc. The main feature of the Leap exploited along with others in this work is its ability to store a history of 60 frames in its buffer and ability to extract information from each of these frames.

The main feature of the Leap exploited along with others in this work is its ability to store 60 frame history in its buffer and ability to extract

Fig. 4: Leap Motion Controller: 2 Monochromatic IR cameras with 3 IR LEDs (Source: developer.leapmotion.com)

information from each of these frames.

B. Features Selected

In this work, eleven features were chosen to distinguish between the two gestures. The gestures involved swiping right or left using either of the respective hands. The palm is facing the Leap Motion with one finger, the index finger extended out is moved from right to left for swiping right and vice-versa. To distinguish between them, the features used were as follows:

1) The palm's 3D coordinates: The model must be able to classify the gestures irrespective of the plane(s) in which they are performed. It facilitates way to capture the motion of the palm.
2) The dx, dy, and dz of the palm position from the initial position: This helps in capturing the difference of motion of either of the three coordinates. Given the two gestures, the dx varies as negative or positive respectively. Other gestures, if implemented might vary profoundly in dy, or perhaps dz, or stay the same position (with minimal, negligible motion due to the stabilization filter errors)
3) Pitch of the palm: Pitch is the angle between the negative z-axis and the projection of the vector onto the y-z plane. In other words, pitch represents rotation around the x-axis. The model is trained by varying the pitch slightly for every gesture and the model must still recognize it as a valid one.
4) Yaw of the hand: Yaw is the angle between the negative z-axis and the projection of the vector onto the x-z plane. In other words, yaw represents rotation around the y-axis.
5) Roll of the palm surface: Roll is the angle between the y-axis and the projection of the vector onto the x-y plane. In other words, roll represents rotation around the z-axis. The roll is negative angle for the right hand and vice versa.
6) Flag of Left or Right Hand: A logical value to determine if the hand detected is a right (1) or left (0) hand.

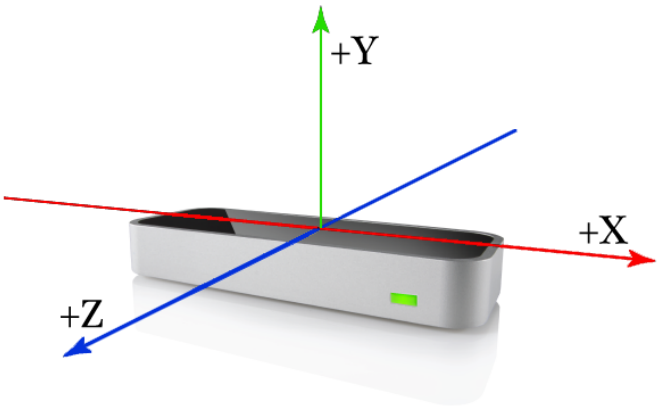


Fig. 5: Leap Motion Coordinates (Source: developer.leapmotion.com)

These 11 features are captured in every frame to distinguish between the gestures. The data from the 60 frames, each comprising of the information of the 11 features, is stored in a data frame to be recorded as one valid time series sample for the model.

C. Obtaining Uncorrelated High Variance Data

To prevent overfitting the model, the gestures are recorded using various modifications to the hand positions. The variations performed were:

- The gesture is swiped in a high radius of curvature bow rather than a straight line.
- The gesture is given both positive and negative pitch angles.
- The gesture is performed at different starting points in the 3D plane

- The gesture speed is varied
- Either the entire arm is moved or just the wrist is while performing the gesture
- Record gestures in low illumination

D. Merging and Time Reversal of Recorded Data

The data acquired from the Leap has a history of 60 frames. The data structure of the frame, frame[0] is the current frame and frame[59] is the 60th frame back in time. Hence, to obtain a valid swipe gesture, the starting point (frame[59]) must be made the first element in the data frame and the end point of the gesture (frame[0]) is made the last element of the data frame. Finally, this data frame of size is stored to a CSV file for training the model. In accompaniment to the data sample acquired above, a label classifying the gesture is also stored in the last column of the CSV file. The training data set is built by recording several gestures for the 2 classes of the gestures to obtain N samples of size . Currently, the obtained number of samples is N=652.

III. MODEL

The time series data acquired is to classify the two gestures correctly. The appropriate models to be considered are Hidden Markov Models [4] (HMM) and Recurrent Neural Networks [11](RNNs).

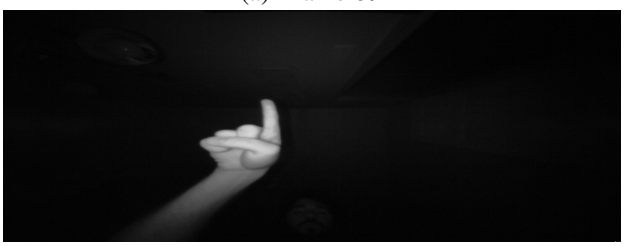
A. Hidden Markov Models

If the model were to be implemented in a HMM chain, the structure would have the observed variable at a given frame, t as the label of the gesture and the hidden variables were the 11 features of the same frame, t. Due to the Markov Chain structure, the probability $P(label|features)$ has the hidden variables at a given frame only depends on the hidden variables of the previous frame, $t-1$. The chain would be of length 60.

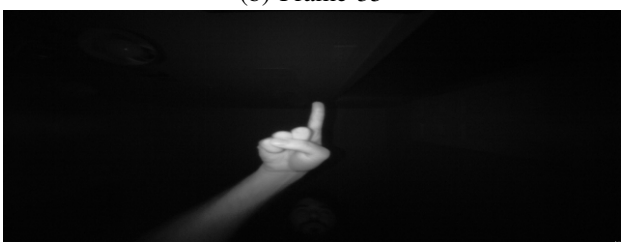
The issues with training the HMM model are:

- The data points must be made probabilistic.
- The conditional probability distribution does not extend beyond one frame
- Training the model is cumbersome. The Markov Chain Monte Carlo [7] (MCMC) is one prescribed method for time series data, but is computationally heavy due to its randomness.

(a) Frame 59



(b) Frame 35



(c) Frame 15

(d) Frame 0

Fig. 6: Images as seen by the Leap Motion-(a), (b), (c), (d) showing frames 59, 35, 15,0 respectively

B. Recurrent Neural Networks

Recurrent Neural Networks are used to train and classify time series data as well. It possesses memory of the input samples through time and assigning/tuning the weights to the neurons. The RNNs back-propagate error through time upon being fed every time-stamp of the input sample. RNNs does not require data to be probabilistic, is easier to train by simple SGD [13] algorithm, and have memory of multiple frames back in time.

However, due to activation functions used in the hidden layers of the RNN, vanishing or exploding

gradients phenomenon occurs [15]. To overcome this issue, the model used is the Long Short Term Memory (LSTM) Models [6]. The forget, read, and write gates work in tandem to ensure the information is retained in the cell long enough.

The model structured [19] [17]to handle the time series data using LSTMs has its layers comprised of 7 LSTM neurons. Fewer neurons yielded under performance and adding more neurons meant overfitting. The next layer, the output layer is a Fully-Connected Layer with 2 neurons- The two classes. The output layer is activated using the Sigmoid function. Figure 8 shows the final model used to train and predict the data.

C. Environment used to Build, Train, and Predict with Model

To build the above model and train it using the acquired data, Python 2.7 [10] libraries for Keras [5] is used. Keras uses either the powerful Theano [14] or Google TensorFlow [3] backend to build, and train Neural Network models. For this work, Keras uses the TensorFlow backend along with the power of GPU due to the support of CUDA Developer Library of NVIDIA [1] for Neural Networks (CUDNN).

GPU is used to train the model faster due to its capability of handling huge pipelines of mathematical operations. In conjunction with the above backbone, Pandas [9], NumpPy [8] and Scikit-learn [12] libraries were used to handle data frames, array operations, and split the data to training and cross validation data, respectively.

D. Training the Model

The model is trained by feeding each of the time series training data in batches of 5. And this is performed over 100 epochs of training. The data set is randomly split in the ratio of 80-20 as Training and Cross Validation data respectively. In addition to model structuring, Keras also handles the fitting of the training data to the model and training it as required.

At every run of an epoch, the training accuracy is calculated and echoed. At the end of all the epochs of training, the model is made to predict on the cross-validation data and its accuracy is also echoed. A good training of the model implies the Cross-Entropy loss of the model is very less and the training accuracy is high but not ideal.

To prevent overfitting, the model is trained several times and each time, the loss and the CV accuracy is noted. At the end of every run of training, the model is saved to a file. The model with the least loss and highest cross validation accuracy is chosen for the prediction. In the chosen model, the Loss has the value of 0.067 and the CV Accuracy achieved is 94

E. Testing the Model

Firstly, the chosen model file is loaded. The Leap Motion is connected to the device and gestures are performed to the Leap Motion, the frame information of 60 frames are recorded to a data-frame and is fed to the models input layer in real-time.

Due to the presence of sigmoid layers at the output layer of the model, both the layers give out the confidence of the gesture performed belong to that respective class. The class with the maximum value of the two is the predicted class by the model. The corresponding text and the audio track voicing the gesture direction are fetched from a list and is displayed/played.

F. Performance of the Model

The models performance when tested in real time achieved high True Positive to False Positive ratio. False positives were observed due to missing presence of the output neuron which classifies any other gestures other than the intended two.

Therefore, any other gesture performed to the Leap Motion would classify it between the two. One work around to overcome the False Positives is have a threshold for the confidence of the output neurons to classify the gestures. This is done since the confidence of the False Positives is very less and the threshold filters them out.

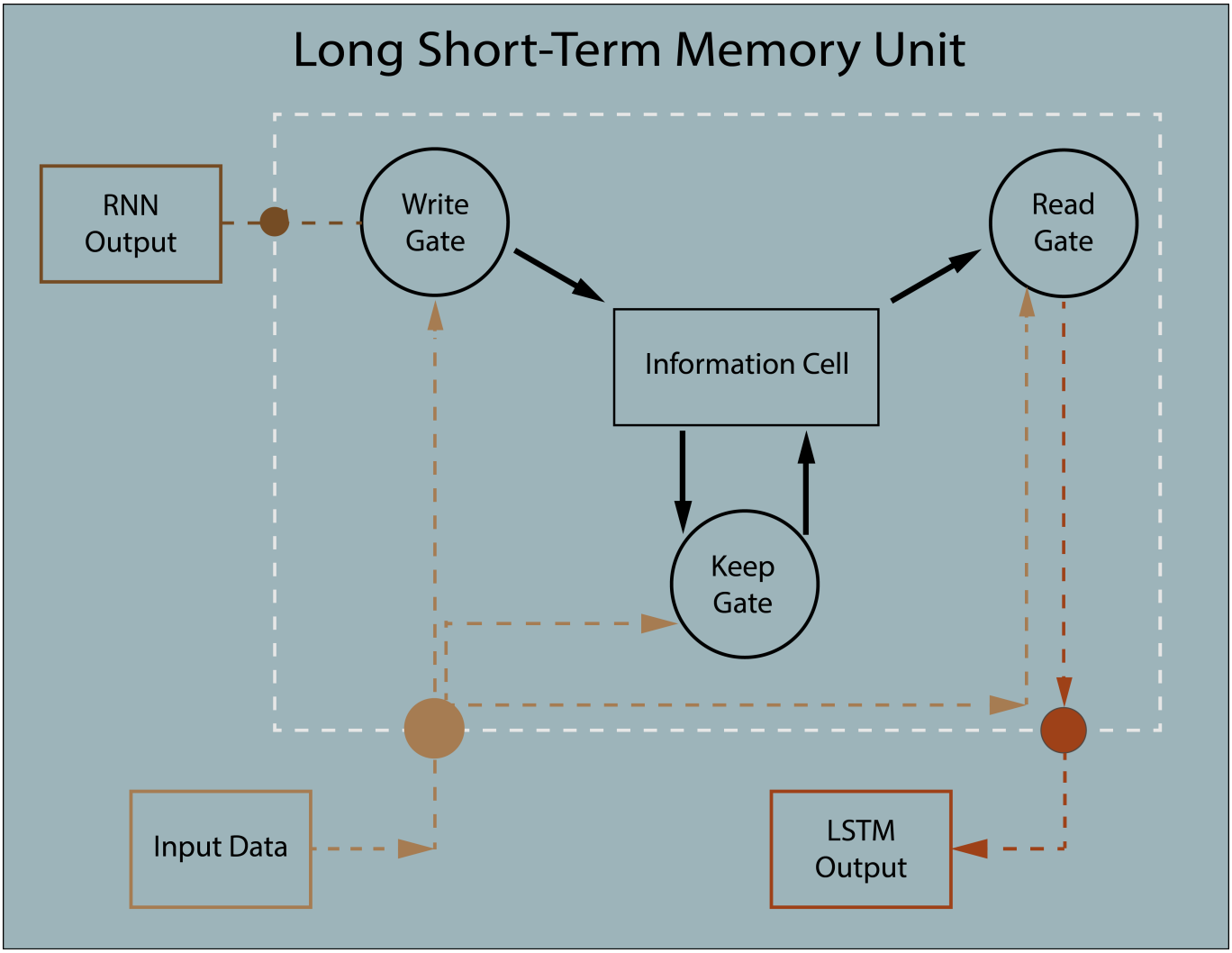


Fig. 7: Single cell of LSTM)

G. Roadblocks

a) Finding the right compatibility: Ubuntu 16.04 (for the right CUDA Support), Python 2.7, Leap SDK 2.3.3, TensorFlow version 1 with GPU

b) Finding the right gesture: We had a hard time selecting a gesture for which we could record consistent data. These are some of the gesture we tried:

- Swipe right and left with palm side wards: the dx feature was erroneous
- Swipe up and down with palm facing down: not much variation in dy feature
- Zoom in and out using pinch strength: no consistency in palm sphere radius feature across frames
- We finally settled for Swipe right and left with palm facing down

c) Finding the optimum hyper parameters for the RNN: After many iterations we settled on 7 LSTM cells

IV. Future Work

Though the model performed to the expectations of the training, there were few shortcomings which needed addressing to make it very robust. Few of them are:

1) Expand the vocabulary of the model to classify more gestures.
2) Include the extra class of free gestures- This eliminates the need of thresholding to filter out False Positives

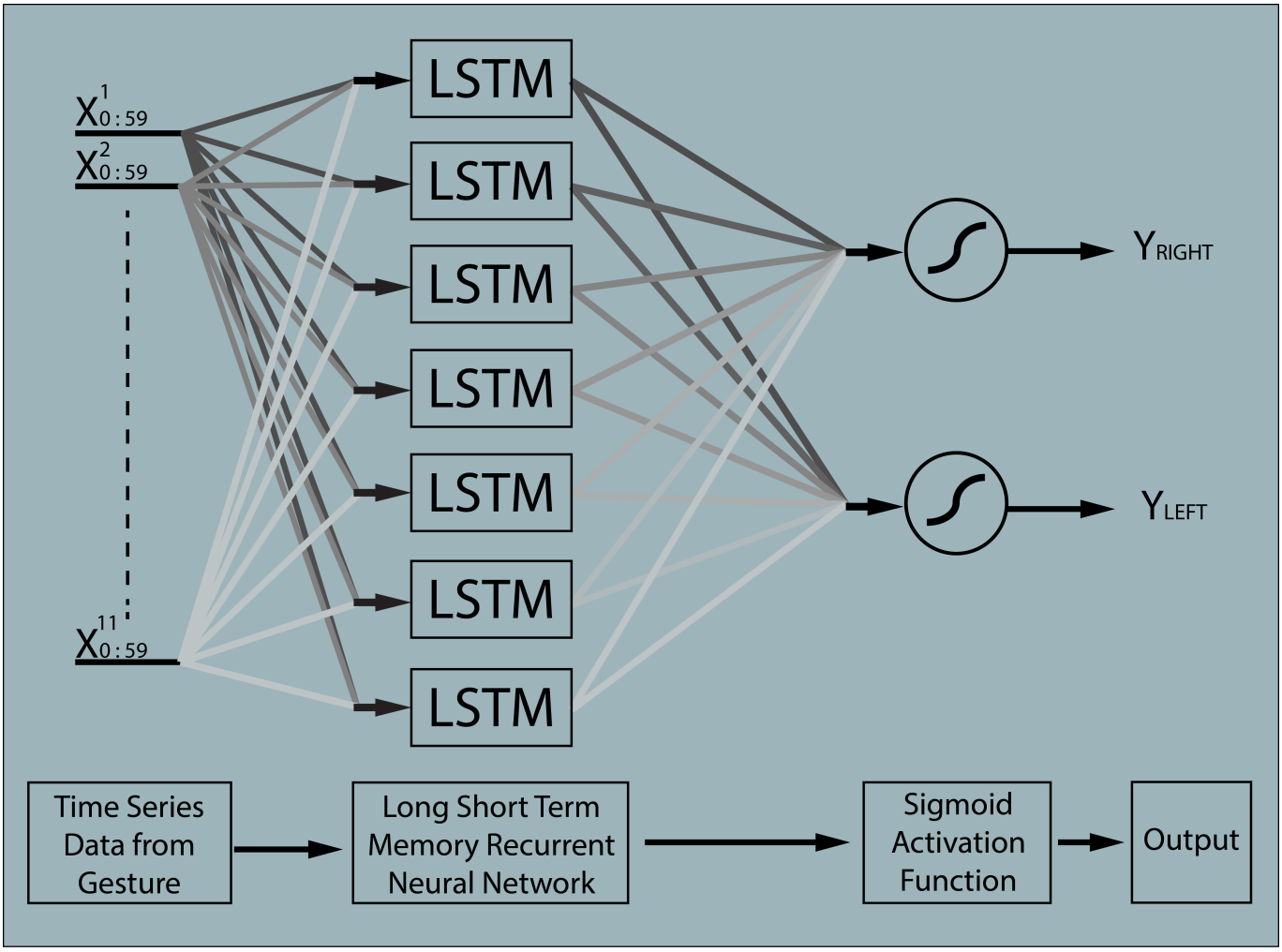


Fig. 8: Chosen Model: Long Short Term Memory Neural Network)

3) Currently, the model predicts Right or Left based on the corresponding hand used to perform the gesture. Ideally, irrespective of the hand, both the gestures must be classified.
4) Record more training samples, with higher variance (using different subjects)

V. Conclusion

In this work, it is shown that using Leap Motion Controller and the power of Deep Learning, a hardware used to detect hands, can be coupled with a Recurrent Neural Network to improvise the detection and further make it distinguish between motion gestures and still hands. Gesture recognition can go a long way if Deep Learning can be implemented on simple hardware as well, rather than sophisticated ones such as the Leap Motion or the Microsoft Kinect.

However, simple cameras fall prey to illumination fluctuations while detecting features. Therefore, it is up to the end user's problem statement to invest in a sophisticated hardware and have the model trained easily with higher accuracy or be economic in choosing a simple camera and have a complicated design process in designing the model.

References

[1] CUDA Developer Library documentation. https://developer.nvidia.com/gpu-accelerated-libraries.
[2] Cypresss EZ-USB chip description. http://www.cypress.com/products/ez-usb-fx2lp. Accessed: 2017-03-30.
[3] Google TensorFlow documentation. https://www.tensorflow.org/.
[4] Hidden Markov model description. https://en.wikipedia.org/wiki/Hidden_Markov_model. Accessed: 2017-03-30.

[5] Keras documentation. https://keras.io/.
[6] Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling description. http://193.6.4.39/~czap/letoltes/IS14/IS2014/PDF/AUTHOR/IS141304.PDF. Accessed: 2017-03-30.
[7] Markov chain Monte Carlo description. https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo. Accessed: 2017-03-30.
[8] Numpy documentation. http://www.numpy.org/.
[9] Pandas documentation. http://pandas-docs.github.io/pandas-docs-travis/.
[10] Python 2.7 documentation. https://docs.python.org/2.7/.
[11] Recurrent neural network description. https://en.wikipedia.org/wiki/Recurrent_neural_network. Accessed: 2017-03-30.
[12] Scikit-learn documentation. http://scikit-learn.org/stable/documentation.html.
[13] Stochastic gradient descent description. https://en.wikipedia.org/wiki/Stochastic_gradient_descent. Accessed: 2017-03-30.
[14] Theano documentation. http://deeplearning.net/software/theano/.
[15] vanishing or exploding gradients phenomenon description. http://neuralnetworksanddeeplearning.com/chap5.html.
[16] A. D. Calin. Gesture recognition on kinect time series data using dynamic time warping and hidden markov models. In *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 264–271, Sept 2016.
[17] V. John, A. Boyali, S. Mita, M. Imanishi, and N. Sanma. Deep learning-based fast hand gesture recognition using representative frames. In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, Nov 2016.
[18] V. I. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, Jul 1997.
[19] S. Shin and W. Sung. Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2274–2277, May 2016.

Appendix: Code

Data Collection

```
import sys, os
sys.path.insert(0, "../lib")
sys.path.insert(0, "../lib/x64")
import Leap
import time
import pandas as pd
import numpy as np
from PIL import Image

leftHand=[]
rightHand=[]
count=502
left_count=238
right_count=259
count_free=0
previous_time=0
frame_count=0
class SimpleListener(Leap.Listener):

    def on_connect(self, controller):
        print "Connected"



    def on_frame(self, controller):
        global count,left_count,right_count,
count_free,previous_time,frame_count

        frame=controller.frame()
        hand=frame.hands
        lFlag=False
        rFlag=False

        if not hand.is_empty:
            for h in hand:
                if h.is_left:
                    leftHand=h
                    lFlag=True
                elif h.is_right:
                    rightHand=h
                    rFlag=True
            if lFlag and rFlag:
                pass

            elif lFlag:
```

```
# Recording Left Gesture

                if (leftHand.palm_normal.roll* Leap.RAD_TO_DEG<=35 and
leftHand.palm_normal.roll* Leap.RAD_TO_DEG>0) and
len(leftHand.fingers.extended())==1:
                    print "Hand Detected for Left"
                    time.sleep(1.5)
                    print "Record now"
                    time.sleep(0.5)
                    frame=controller.frame()
                    lefthand=frame.hands[0]
                    current_time=frame.timestamp
                    time_delta=current_time-previous_time

                    data_left=[[] for _ in range(12)]
                    # Record only if frame rate during gesture is low
                    if time_delta>5000000 and time_delta<1000000000:
                        for i in range(59,-1,-1):
                            data_left[0].append(count)

data_left[1].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[0],4))

data_left[2].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[1],4))

data_left[3].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[2],4))

data_left[4].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[0],4)-
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
0],4))

data_left[5].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[1],4)-
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
1],4))

data_left[6].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[2],4)-
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
2],4))

data_left[7].append(round(controller.frame(i).hand(lefthand.id).palm_n
ormal.roll,4))

data_left[8].append(round(controller.frame(i).hand(lefthand.id).direct
ion.yaw,4))
```

```
data_left[9].append(round(controller.frame(i).hand(lefthand.id).direct
ion.pitch,4))
                            data_left[10].append(int(rFlag))
                            data_left[11].append(1)

                        df_left=pd.DataFrame(data=data_left)
                        df_left=df_left.transpose()

df_left.to_csv("./CSV/%s.csv"%(str(count)),sep=',',index=False,
header=False)
                        count+=1
                        left_count+=1
                        print "Swipe Left Gesture Recorded %s"
%(str(left_count))
                    else:
                        print "Try Again"
                    time.sleep(3)
                    print "Ready"
                    time.sleep(1)
                    previous_time=current_time


# Recording Right Gesture

                if ((rightHand.palm_normal.roll* Leap.RAD_TO_DEG>=-35
and rightHand.palm_normal.roll* Leap.RAD_TO_DEG<0) and
(len(rightHand.fingers.extended())==1)):
                    print "Hand Detected for Swipe Right"
                    time.sleep(1.5)
                    print "Record now"
                    time.sleep(0.5)
                    frame=controller.frame()
                    righthand=frame.hands[0]
                    current_time=frame.timestamp
                    time_delta=current_time-previous_time
                    data_right=[[] for _ in range(12)]
                    # Record only if frame rate during gesture is low
                    if time_delta>5000000 and time_delta<1000000000:
                        for i in range(59,-1,-1):
                            data_right[0].append(count)

data_right[1].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[0],4))

data_right[2].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[1],4))

data_right[3].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[2],4))

data_right[4].append(round(controller.frame(i).hand(righthand.id).stab
```

```
ilized_palm_position[0],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[0],4))

data_right[5].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[1],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[1],4))

data_right[6].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[2],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[2],4))

data_right[7].append(round(controller.frame(i).hand(righthand.id).palm
_normal.roll,4))

data_right[8].append(round(controller.frame(i).hand(righthand.id).dire
ction.yaw,4))

data_right[9].append(round(controller.frame(i).hand(righthand.id).dire
ction.pitch,4))
                            data_right[10].append(int(rFlag))
                            data_right[11].append(0)

                        df_right=pd.DataFrame(data=data_right)
                        df_right=df_right.transpose()

df_right.to_csv("./CSV/%s.csv"%(str(count)),sep=',',index=False,
header=False)
                        count+=1
                        right_count+=1
                        print "Swipe Right Gesture Recorded %s"
%(str(right_count))
                    else:
                        print "Try Again"
                    time.sleep(3)
                    print "Ready"
                    time.sleep(1)
                    previous_time=current_time


        else:
            pass



def main():
    listener = SimpleListener()
    controller = Leap.Controller()
    controller.add_listener(listener)
    # Run infinitely until Enter is pressed
```

```
    print "Hit Enter to quit!"
    try:
        sys.stdin.readline()
    except KeyboardInterrupt:
        pass
    finally:
        controller.remove_listener(listener)

if __name__ == "__main__":
    main()
```

Data Merging

```
import csv
fout=open("./one_file.csv","a")
# first file:
count=0
with open('./CSV/0.csv', 'rb') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        if row[1]==str(0.0) and row [2]==str(0.0) and
row[3]==str(0.0):
            count+=1
        else:
            pass
#Eliminate the samples which had erroneous input recorded
if count==0:
    for line in open("./CSV/0.csv"):
        fout.write(line)

# Rest of the files:
for num in range(1,503):
    count=0
    with open('./CSV/'+str(num)+'.csv', 'rb') as f:
        reader = csv.reader(f, delimiter=',')
        for row in reader:
            if row[1]==str(0.0) and row [2]==str(0.0) and
row[3]==str(0.0):
                count+=1
            else:
                pass
    if count==0:
        f = open("./CSV/"+str(num)+".csv")
        for line in f:
            fout.write(line)
        f.close()
fout.close()
```

Training Recurrent Neural Network

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.models import load_model
from keras.utils import np_utils
from imblearn.over_sampling import SMOTE
import time

# Import dataset
dataset =
pd.read_csv('/home/sharath/LeapDeveloperKit_2.3.1+31549_linux/LeapSDK/
src/one_file.csv',header=None)

#Split to Data samples and labels
x = dataset.iloc[:, 1:11].values
y = dataset.iloc[:, 11].values

#Number of time series to work on is 60
t=60
n_samples=len(dataset)/t


#Reshape input samples to Nx60x11
x = np.reshape(x, (x.shape[0]/t, t, x.shape[1]))
#Set labels to these N samples
indices=[ind for ind in range(len(y)) if ind%t==0]
y=y[indices]

# One-hot encode the classes
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
y_new=label_encoder.fit_transform(y)
y_one_hot=np_utils.to_categorical(y_new)
y=y_one_hot

#Split data to training and testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size =
0.2)

#Construct the model
n_neurons=7
start_time=time.time()
model = Sequential()
model.add(LSTM(n_neurons,
input_shape=(X_train.shape[1],X_train.shape[2])))
model.add(Dense(output_dim=2,init = 'uniform', activation='sigmoid'))
```

```
model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=['accuracy'])

#Train the model
model.fit(X_train, y_train, epochs=100, batch_size=5)

#Evaluation of the Training time
print "%s Minutes of Execution" %str((time.time()-start_time)/60)

#Save the model for prediction
model.save('model_test.h5')
print "Model Saved"

# Evaluate the model on CV Data
scores = model.evaluate(X_test, y_test, verbose=0)
print "%s: %.2f%%" % (model.metrics_names[1], scores[1]*100)
```

Prediction on New Data

```
import numpy as np
import pandas as pd
from keras.models import load_model
from keras.utils import np_utils
import time
import sys
sys.path.insert(0, "../lib")
sys.path.insert(0, "../lib/x64")
import Leap
import tensorflow as tf
import os

leftHand=[]
rightHand=[]
previous_time=0

#Load the model
model=load_model('model_test.h5')
print "Model Loaded"

gesture=["Swipe Right","Swipe Left"]
audio=["aplay right.wav","aplay left.wav"]
graph=tf.get_default_graph()

class SimpleListener(Leap.Listener):

    def on_connect(self, controller):
        print "Connected"

    def on_frame(self, controller):
        global model, graph,previous_time, gesture, audio
```

```
        frame=controller.frame()
        hand=frame.hands
        lFlag=False
        rFlag=False

        if not hand.is_empty:
            for h in hand:
                if h.is_left:
                    leftHand=h
                    lFlag=True
                elif h.is_right:
                    rightHand=h
                    rFlag=True
            if lFlag and rFlag:
                pass

            elif lFlag:

                print "Hand Detected"
                time.sleep(1.5)
                print "Record now"
                time.sleep(0.5)
                frame=controller.frame()
                lefthand=frame.hands[0]
                current_time=frame.timestamp
                time_delta=current_time-previous_time
                data_left=[[] for _ in range(12)]
                if time_delta>4000000 and time_delta<1000000000:
                    for i in range(59,-1,-1):
                        data_left[0].append(0)

data_left[1].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[0],4))

data_left[2].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[1],4))

data_left[3].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[2],4))

data_left[4].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[0],4)-
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
0],4))

data_left[5].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[1],4)-
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
1],4))

data_left[6].append(round(controller.frame(i).hand(lefthand.id).stabil
ized_palm_position[2],4)-
```

```
round(controller.frame(59).hand(lefthand.id).stabilized_palm_position[
2],4))

data_left[7].append(round(controller.frame(i).hand(lefthand.id).palm_n
ormal.roll,4))

data_left[8].append(round(controller.frame(i).hand(lefthand.id).direct
ion.yaw,4))

data_left[9].append(round(controller.frame(i).hand(lefthand.id).direct
ion.pitch,4))
                        data_left[10].append(int(rFlag))
                        data_left[11].append(1)

                    df_left=pd.DataFrame(data=data_left)
                    df_left=df_left.transpose()

                    x=df_left.iloc[:,1:11].values
                    with graph.as_default():

y_pred=model.predict(np.reshape(x,(1,x.shape[0],x.shape[1])))
                    pred=np.argmax(y_pred)
                    # Eliminate the False Postives
                    if (y_pred[0][1] or y_pred[0][0])>=0.3:
                        print gesture[pred]
                        os.system(audio[pred])
                    else:
                        print "try again"
                else:
                    print "Try Again"

                time.sleep(3)
                print "Ready"
                time.sleep(1)
                previous_time=current_time



            elif rFlag:


                print "Hand Detected"
                time.sleep(1.5)
                print "Record now"
                time.sleep(0.5)

                frame=controller.frame()
                righthand=frame.hands[0]
                current_time=frame.timestamp
                time_delta=current_time-previous_time
                data_right=[[] for _ in range(12)]
                if time_delta>5000000 and time_delta<1000000000:
```

```
                    for i in range(59,-1,-1):
                        data_right[0].append(0)

data_right[1].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[0],4))

data_right[2].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[1],4))

data_right[3].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[2],4))

data_right[4].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[0],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[0],4))

data_right[5].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[1],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[1],4))

data_right[6].append(round(controller.frame(i).hand(righthand.id).stab
ilized_palm_position[2],4)-
round(controller.frame(59).hand(righthand.id).stabilized_palm_position
[2],4))

data_right[7].append(round(controller.frame(i).hand(righthand.id).palm
_normal.roll,4))

data_right[8].append(round(controller.frame(i).hand(righthand.id).dire
ction.yaw,4))

data_right[9].append(round(controller.frame(i).hand(righthand.id).dire
ction.pitch,4))
                        data_right[10].append(int(rFlag))
                        data_right[11].append(0)
                    df_right=pd.DataFrame(data=data_right)
                    df_right=df_right.transpose()

                    x = df_right.iloc[:, 1:11].values
                    with graph.as_default():

y_pred=model.predict(np.reshape(x,(1,x.shape[0],x.shape[1])))
                    pred=np.argmax(y_pred)
                    # Eliminate the False Postives
                    if (y_pred[0][0] or y_pred[0][1])>=0.3:
                        print gesture[pred]
                        os.system(audio[pred])
                    else:
                        print "try again"
                else:
```

```
                    print "Try Again"

                time.sleep(3)
                print "Ready"
                time.sleep(1)
                previous_time=current_time

            else:
                pass



def main():
    listener = SimpleListener()
    controller = Leap.Controller()
    controller.add_listener(listener)

    print "Hit Enter to quit!"
    try:
        sys.stdin.readline()
    except KeyboardInterrupt:
        pass
    finally:
        controller.remove_listener(listener)

if __name__ == "__main__":
    main()
```