A Project Report

On

HAND GESTURE RECOGNITION

Submitted in partial fulfillment of the

requirement for the award of the degree of

B. Tech in Computer Science and Engineering

Under The Supervision of
Dr. C Ramesh
Assistant Professor
Department of Computer Science & Engineering

Submitted By:

Faraz Ahmad
18SCSE1120001

Shashank Singh
18SCSE1120010

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
OCT,2021

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Hand Gesture Recognition"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Galgotias University is an authentic record of my own work carried out over a period from August 2021 to October 2021 under the supervision of **(C.Ramesh)** (Assistant Professor (Grade-I), Computer Science).

[illegible]

(Student Signature)

Faraz Ahmad

18021120001

[illegible]

(**Student Signature)**

Shashank Singh

18021120009

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

[illegible]

(Supervisor Signature)

C. Ramesh

Assistant Professor (Grade-I)

CERTIFICATE

The Final Project Viva-Voce examination of **18SCSE1120010 -SHASHANK SINGH**, **18SCSE1120001 – FARAZ AHMED NAGRAMI** has been held on __________________ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Time:

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly indebted to C. Ramesh for their guidance and constant supervision as well as for necessary information regarding the project and for their support in completing the project.

We would like to express our gratitude towards our parents and Galgotias University for their kind co-operation and encouragement, which helped us in completion of this project.

Our thanks and appreciations go to our colleague in developing the project and people who have willingly helped us out with their abilities.

ABSTRACT

Gesture Recognition is the most favored and practicable solution to improve human interaction with computer and has been widely accepted in recent years thanks to its practice in gaming devices such as Xbox, ps4, etc. as well as other devices such as laptops, smartphone, etc. of gestures and particularly the recognition of hand gestures is utilized in various application such as accessibility support, crisis management, medicine etc. This report depicts our fourth-year project "Acknowledgment of gestures", describing the diverse direction and methods that are utilized for hand gesture recognition. Additionally, it portrays many methods utilized for evolution to test the refined software artefact. Since hand gesture recognition linked to two main machine learning and image processing fields, the report further describes different API's and tools can be utilized to execute different approaches and methodologies in such areas.

INTRODUCTION

A gesture is a development of a body part, which can be a hand, head, neck or some other to express an emblematic portrayal of information. It is essentially a method for interfacing in a non-verbal way utilizing our body parts to convey the ideal message. It incorporates different developments, for example, head developments, hand developments or other body parts developments. It is the utilization of hand motions utilized to express the non-verbal correspondence with the PC interface.

Gesture Recognition

Gesture Recognition is characterized as the procedure to recognize the different movements structured by the client and are encouraged to the machine, which can be a PC, Tablet, or some other machine. Gesture Recognition can be examined utilizing two techniques. The first is static motion acknowledgment and the second is dynamic motion acknowledgment. Utilizing Static motion acknowledgment, the predefined signals put away in database can be distinguished while Dynamic motion acknowledgment is progressively founded on down to earth circumstances. With greater reasonableness comes more trouble.

As we probably are aware, the vision-based procedure of hand motion acknowledgment is a significant piece of human computer interaction (HCI). In the most recent decades, the console and mouse assume a significant job in human-PC cooperation. Be that as it may, because of the quick advancement of equipment and programming, new kinds of HCI techniques are required. Specifically, innovations, for example, speech acknowledgment and motion acknowledgment pull in a great deal of consideration in the region of HCI. There are 2 types of gestures: static and dynamic. It is altogether different from customary equipment-based techniques what's more, can achieve human-PC association with the help of signal acknowledgment. Motion acknowledgment decides the expectation of the client through acknowledgment to gestures or movements of the body or body parts. Through the previous decades, numerous scientists have been putting forth attempts to upgrade hand motion acknowledgment strategies. Hand motion acknowledgment is of incredible incentive in numerous apps, for example, acknowledgment of communication via gestures, enlarged reality (computer generated reality), gesture-based communication mediators for the handicapped, and robot control.

Gesture recognition features:

- More accurate
- High security/stability
- Time saving to unlock a device
- Some of the applications of hand gesture recognition are applied in the following sectors: Automotive sector

- Consumer electronics sector Transit sector Gaming sector
- To unlock smartphones Defence Home automation
- Automated sign language translation

- Motion acknowledgment can be led with procedures from PC vision and picture preparing.

Dynamic Gestures

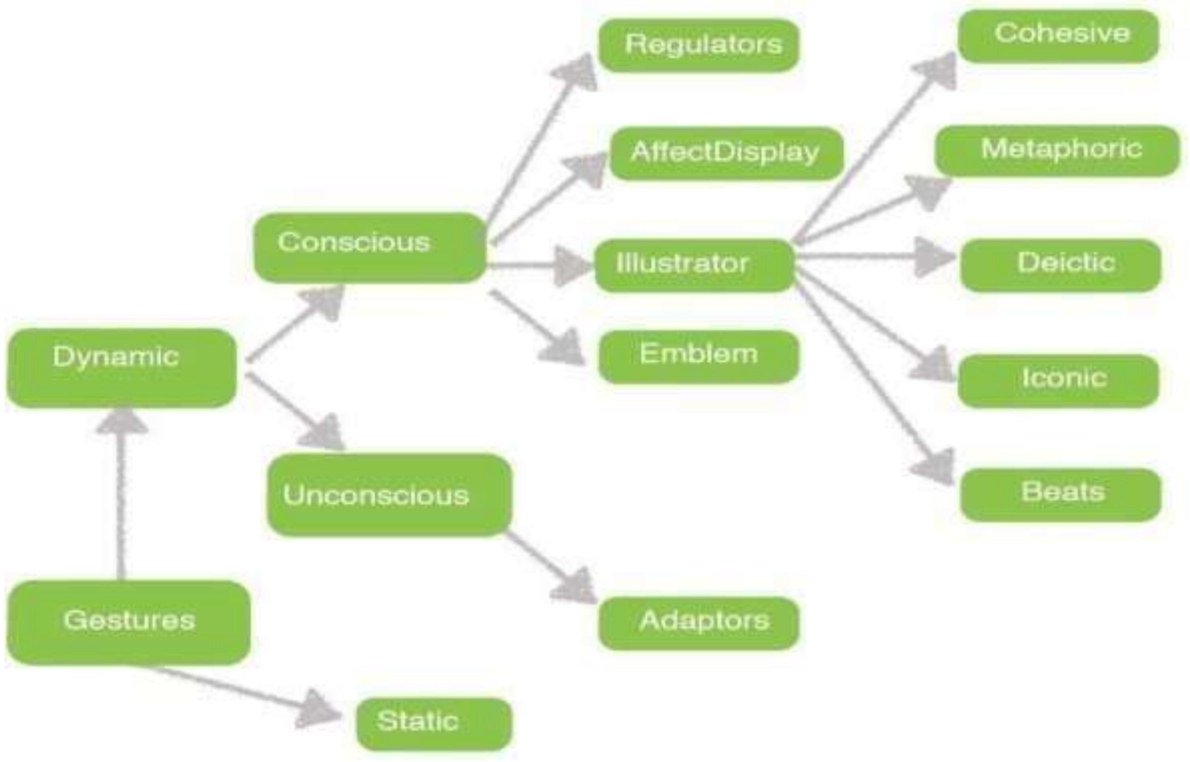


Figure 1: Gestures Classification

LITERATURE REVIEW

The picture of the given hand is sectioned utilizing two distinct strategies; The skin colour-based division by applying HSV colour model sand grouping edge strategies. The altered course examination calculation is received to discover the connection between the unsteadied parameters (variance and covalent) from the information, and is utilized to ascertain the item(hand) incline and pattern by identifying the heading of the hand signal. As we probably are aware, the vision-based procedure of hand motion acknowledgment is a significant piece of human computer interaction (HCI). In the most recent decades, the console and mouse assume a significant job in human-PC cooperation. Be that as it may, because of the quick advancement of equipment and programming, new kinds of HCI techniques are required. Specifically, innovations, for example, speech acknowledgment and motion acknowledgment pull in a great deal of consideration in the region of HCI.

Computer vision and Digital Image Processing

The sense of sight is arguably the most important of man's five senses. It provides a huge amount of information about the world that is rich in detail and delivered at the speed of light. However, human vision is not without its limitations, both physical and psychological. Through digital imaging technology and computers, man has transcended many visual limitations. He can see into far galaxies, the microscopic world, the sub-atomic world, and even “observe” infra-red, x-ray, ultraviolet and other spectra for medical diagnosis, meteorology, surveillance, and military uses, all with great success.

While computers have been central to this success, for the most part man is the sole interpreter of all the digital data. For a long time, the central question has been whether computers can be designed to analyze and acquire information from images autonomously in the same natural way humans can.

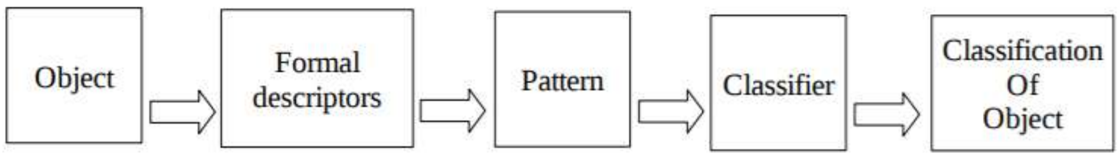
The main difficulty for computer vision as a relatively young discipline is the current lack of a final scientific paradigm or model for human intelligence and human vision itself on which to build a infrastructure for computer or machine learning. The use of images has an obvious drawback.

Humans perceive the world in 3D, but current visual sensors like cameras capture the world in 2D images. The result is the natural loss of a good deal of information in the captured images. Without a proper paradigm to explain the mystery of human vision and perception, the recovery of lost information (reconstruction of the world) from 2D images represents a difficult hurdle for machine vision. However, despite this limitation, computer vision has progressed, riding mainly on the remarkable advancement of decades-old digital image processing techniques, using the science and methods contributed by other disciplines such as optics, neurobiology, psychology, physics, mathematics, electronics, computer science, artificial intelligence and others.

Pattern Recognition and Classifiers

In computer vision a physical object map to a particular segmented region in the image from which object descriptors or features may be derived. A feature is any characteristic of an image, or any region within it, that can be measured. Objects with common features may be grouped into classes, where the combination of features may be considered a pattern. Object recognition may be understood to be the assignment of classes to objects based on their respective patterns.

The general steps in pattern recognition may be summarized in Figure below:



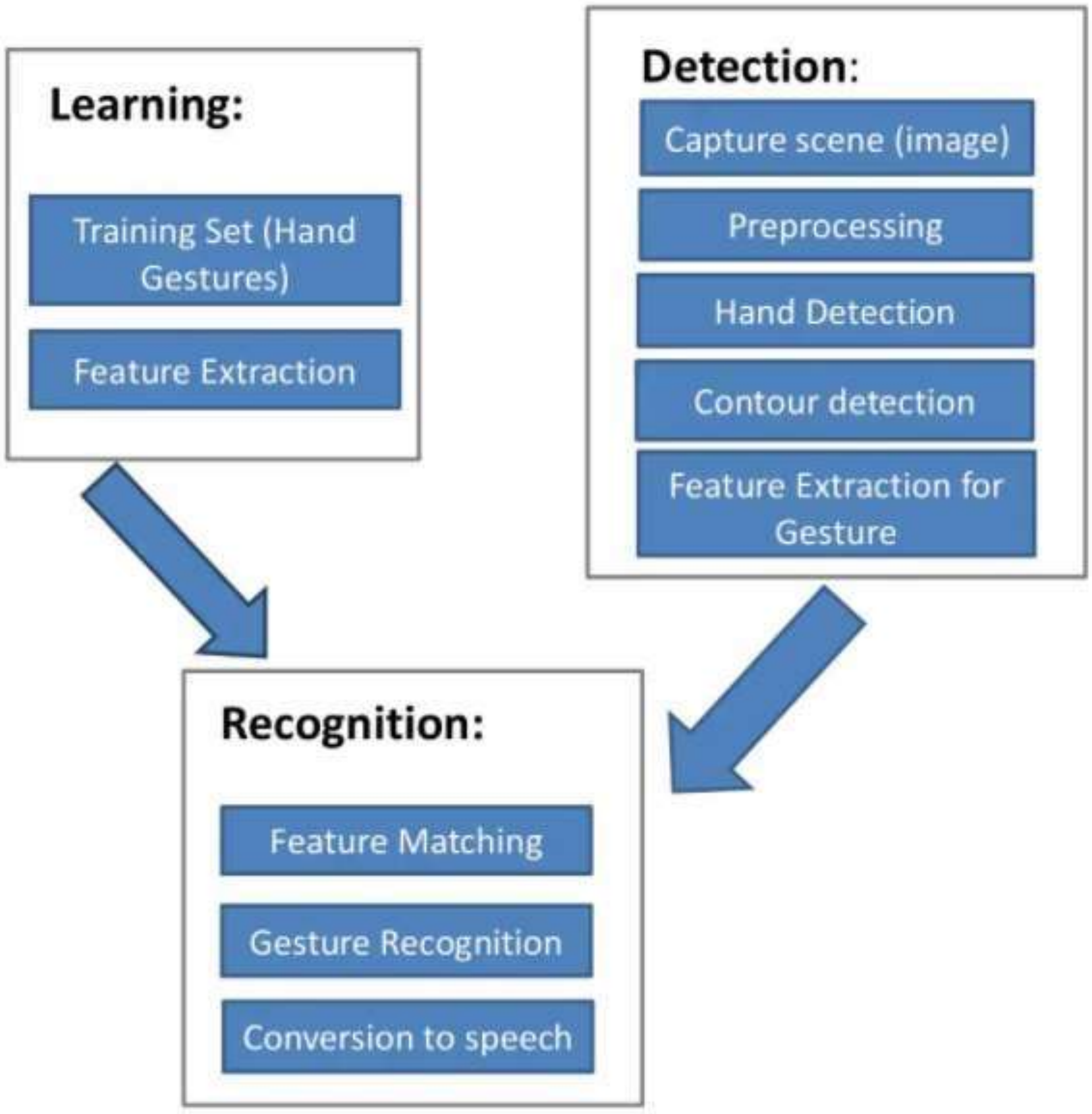
The most important step is the design of the formal descriptors because choices have to be made on which characteristics, quantitative or qualitative, would best suit the target object and in turn determines the success of the classifier.

In statistical pattern recognition, quantitative descriptions called features are used. The set of features constitutes the pattern vector or feature vector, and the set of all possible patterns for the object form the pattern space X (also known as feature space).

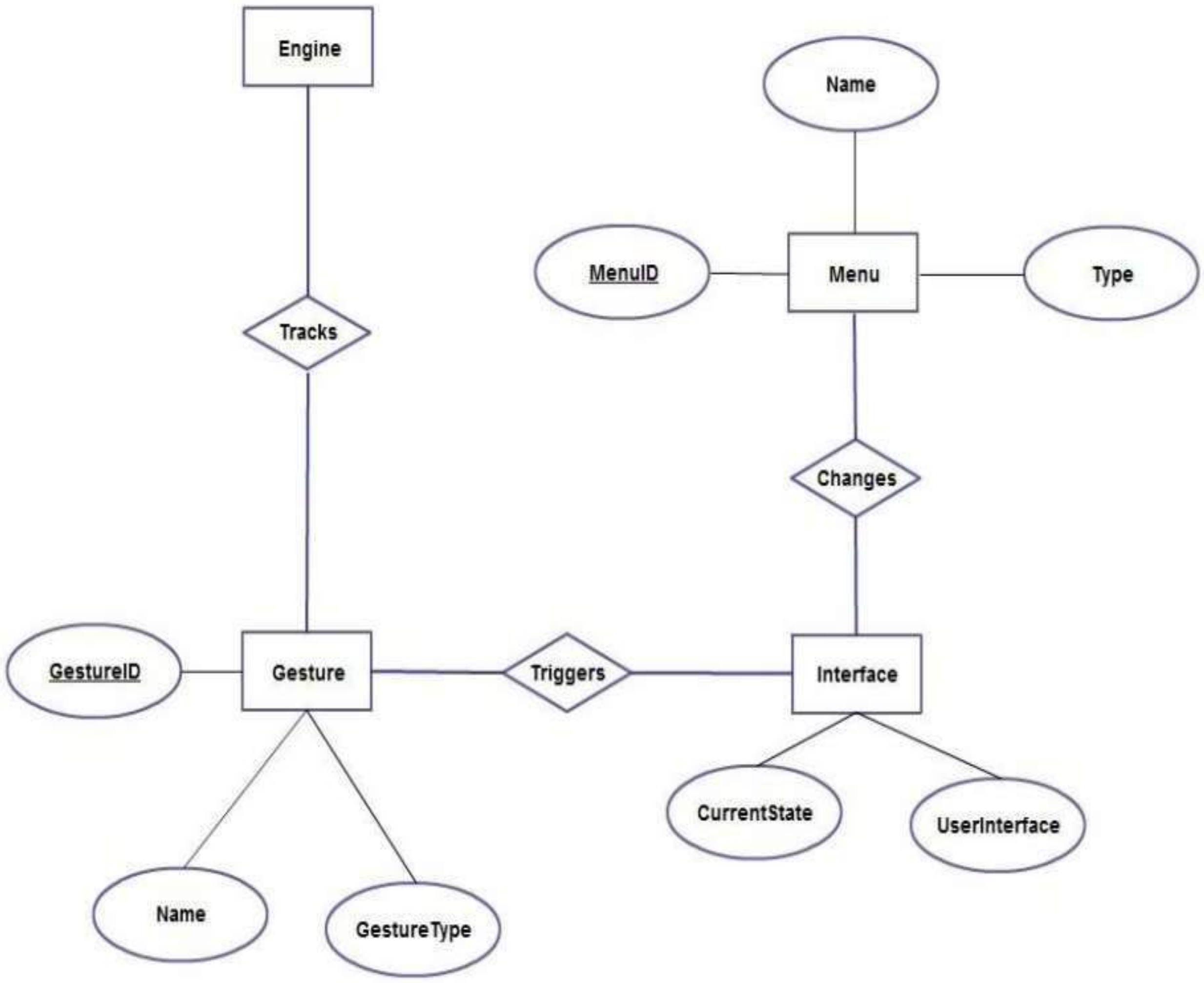
Quantitatively, similar objects in each class will be located near each other in the feature space forming clusters, which may ideally be separated from dissimilar objects by lines or curves called discrimination functions. Determining the most suitable discrimination function or discriminant to use is part of classifier design.

PROJECT DESIGN

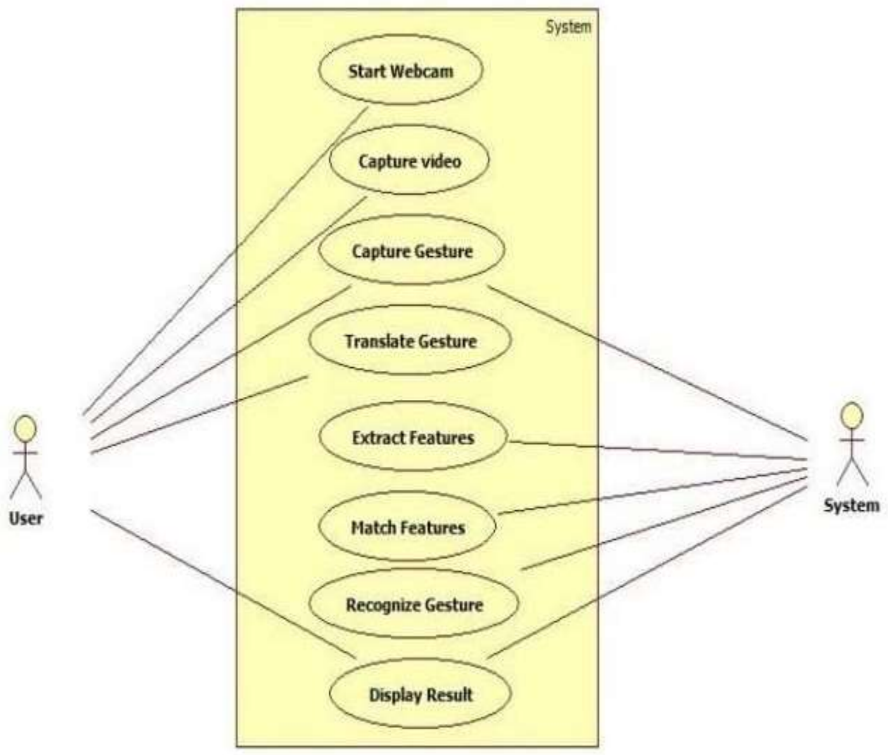
- **Flow Chart**



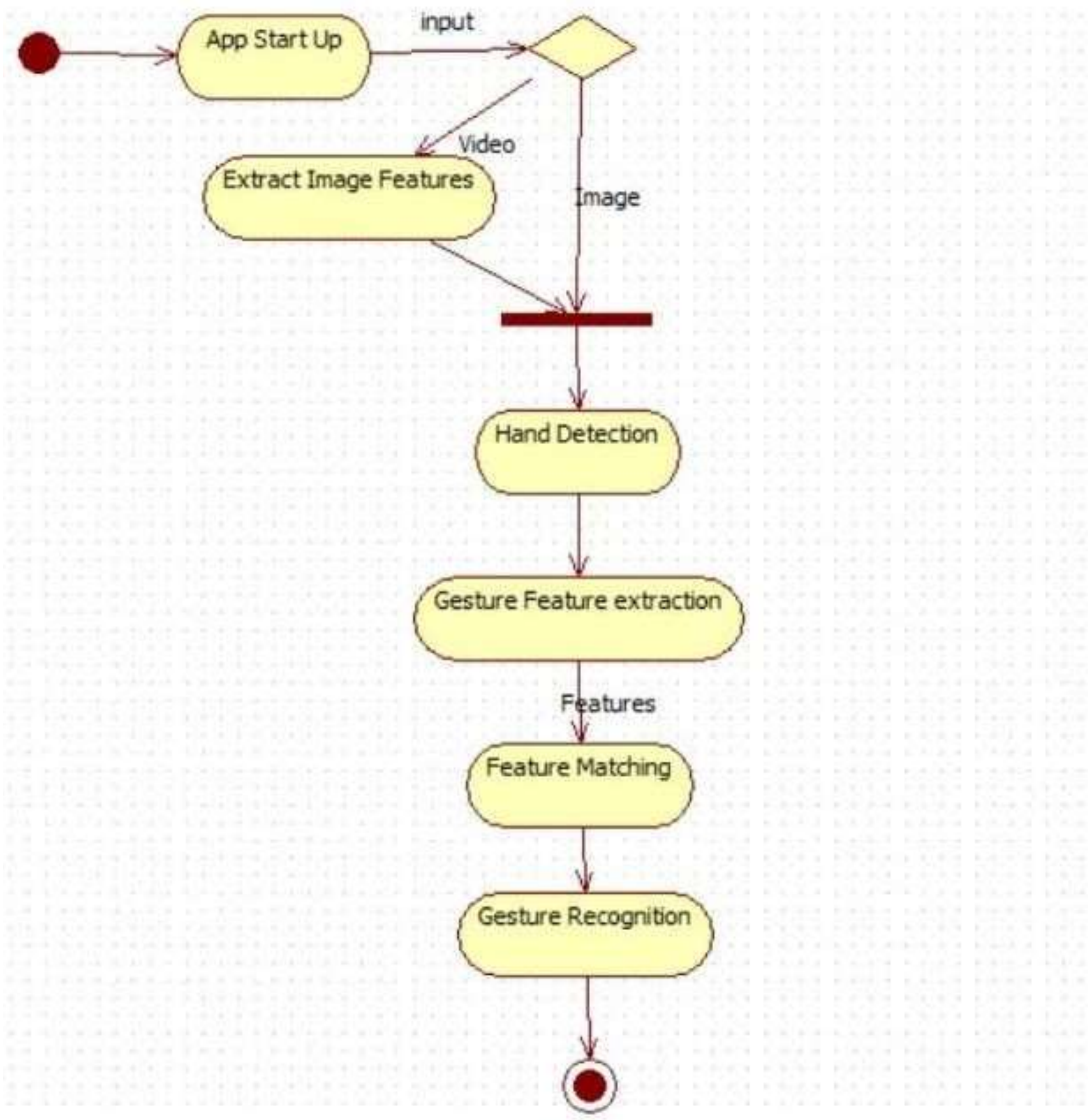
- **Entity relationship diagram**



- **UML Diagram:**

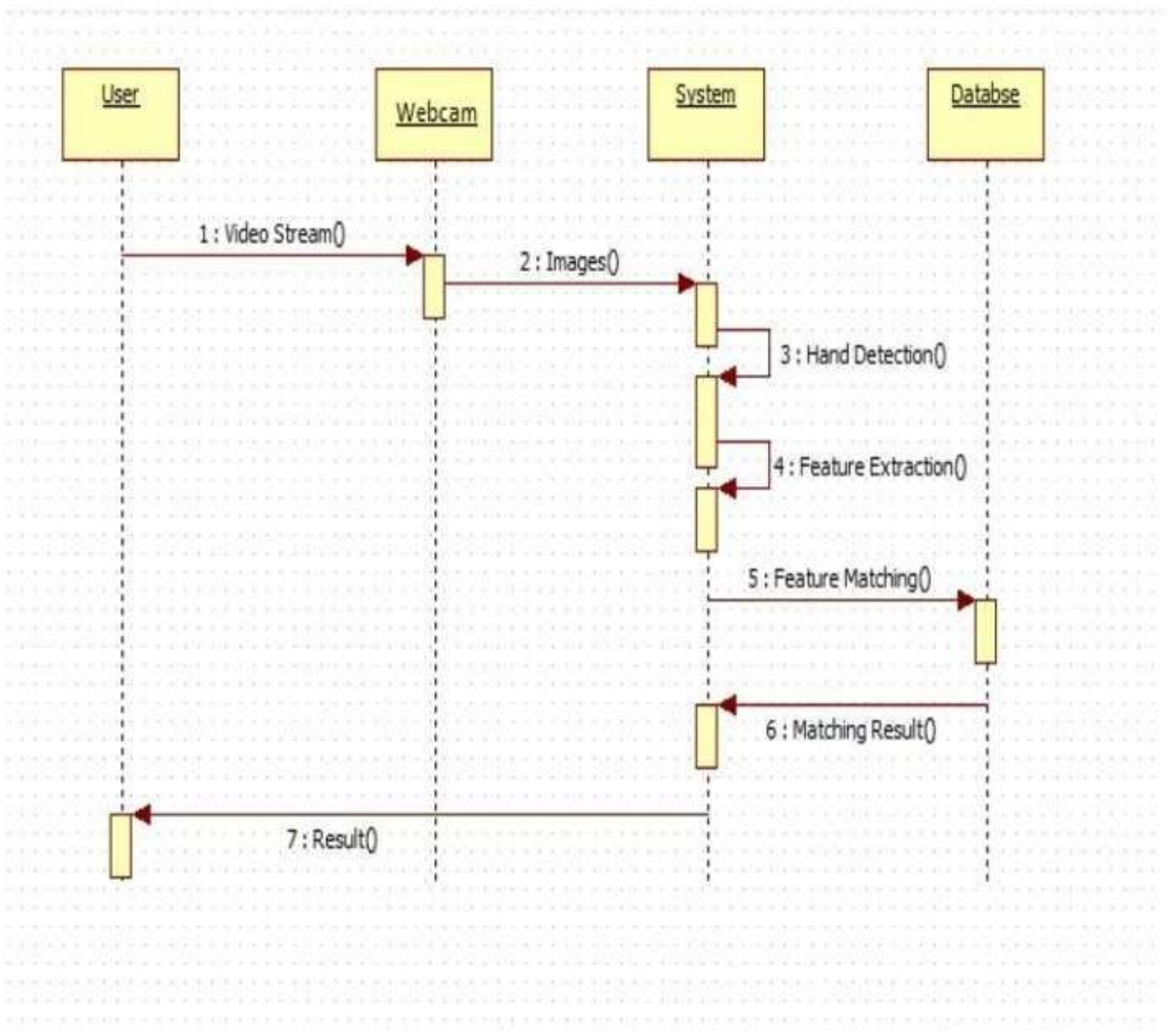

System
Start Webcam
Capture video
Capture Gesture
Translate Gesture
Extract Features
Match Features
Recognize Gesture
Display Result
User
System

- **Flow Diagram:**



Flow Diagram

- **Sequence Diagram:**



Sequence Diagram for proposed system

Implementation of Code:

Collect-data.py:

```
import cv2

import numpy as np

import os

import string

# Create the directory structure

if not os.path.exists("data"):

    os.makedirs("data")

if not os.path.exists("data/train"):

    os.makedirs("data/train")

if not os.path.exists("data/test"):

    os.makedirs("data/test")

for i in range(3):

    if not os.path.exists("data/train/" + str(i)):

        os.makedirs("data/train/"+str(i))

    if not os.path.exists("data/test/" + str(i)):

        os.makedirs("data/test/"+str(i))


for i in string.ascii_uppercase:

    if not os.path.exists("data/train/" + i):

        os.makedirs("data/train/"+i)

    if not os.path.exists("data/test/" + i):

        os.makedirs("data/test/"+i)
```

```
# Train or test
mode = 'train'
directory = 'data/'+mode+'/'
minValue = 70

cap = cv2.VideoCapture(0)
interrupt = -1

while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {
            'zero': len(os.listdir(directory+"/0")),
            'one':  len(os.listdir(directory+"/1")),
            'two': len(os.listdir(directory+"/2")),
    #        'three': len(os.listdir(directory+"/3")),
    #        'four': len(os.listdir(directory+"/4")),
    #        'five': len(os.listdir(directory+"/5")),
    #        'six': len(os.listdir(directory+"/6")),
            'a': len(os.listdir(directory+"/A")),
            'b': len(os.listdir(directory+"/B")),
            'c': len(os.listdir(directory+"/C")),
            'd': len(os.listdir(directory+"/D")),
            'e': len(os.listdir(directory+"/E")),
            'f': len(os.listdir(directory+"/F")),
            'g': len(os.listdir(directory+"/G")),
            'h': len(os.listdir(directory+"/H")),
            'i': len(os.listdir(directory+"/I")),
            'j': len(os.listdir(directory+"/J")),
            'k': len(os.listdir(directory+"/K")),
            'l': len(os.listdir(directory+"/L")),
            'm': len(os.listdir(directory+"/M")),
            'n': len(os.listdir(directory+"/N")),
            'o': len(os.listdir(directory+"/O")),
            'p': len(os.listdir(directory+"/P")),
            'q': len(os.listdir(directory+"/Q")),
            'r': len(os.listdir(directory+"/R")),
            's': len(os.listdir(directory+"/S")),
            't': len(os.listdir(directory+"/T")),
            'u': len(os.listdir(directory+"/U")),
            'v': len(os.listdir(directory+"/V")),
            'w': len(os.listdir(directory+"/W")),
            'x': len(os.listdir(directory+"/X")),
            'y': len(os.listdir(directory+"/Y")),
            'z': len(os.listdir(directory+"/Z"))
            }

    # Printing the count in each set to the screen
    # cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    # cv2.putText(frame, "IMAGE COUNT", (10, ), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "ZERO : "+str(count['zero']), (10, 70), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
```

```
    cv2.putText(frame, "ONE : "+str(count['one']), (10, 80), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "TWO : "+str(count['two']), (10, 90), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "THREE : "+str(count['three']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "FOUR : "+str(count['four']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "FIVE : "+str(count['five']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # cv2.putText(frame, "SIX : "+str(count['six']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "b : "+str(count['b']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "c : "+str(count['c']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "d : "+str(count['d']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "e : "+str(count['e']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "f : "+str(count['f']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "g : "+str(count['g']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "h : "+str(count['h']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "i : "+str(count['i']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "k : "+str(count['k']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "l : "+str(count['l']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "m : "+str(count['m']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "n : "+str(count['n']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "o : "+str(count['o']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "p : "+str(count['p']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "q : "+str(count['q']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "r : "+str(count['r']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "s : "+str(count['s']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
    cv2.putText(frame, "t : "+str(count['t']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "u : "+str(count['u']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "v : "+str(count['v']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "w : "+str(count['w']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "x : "+str(count['x']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "y : "+str(count['y']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    cv2.putText(frame, "z : "+str(count['z']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)

    # Coordinates of the ROI

    x1 = int(0.5*frame.shape[1])

    y1 = 10

    x2 = frame.shape[1]-10

    y2 = int(0.5*frame.shape[1])

    # Drawing the ROI

    # The increment/decrement by 1 is to compensate for the bounding box

    cv2.rectangle(frame, (220-1, 9), (620+1, 419), (255,0,0) ,1)

    # Extracting the ROI

    roi = frame[10:410, 220:520]

#    roi = cv2.resize(roi, (64, 64))


    cv2.imshow("Frame", frame)

    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)


    blur = cv2.GaussianBlur(gray,(5,5),2)

    # #blur = cv2.bilateralFilter(roi,9,75,75)


    th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2
)
```

```
  ret, test_image = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

  #time.sleep(5)

  #cv2.imwrite("/home/rc/Downloads/soe/im1.jpg", roi)

  #test_image = func("/home/rc/Downloads/soe/im1.jpg")




  test_image = cv2.resize(test_image, (300,300))

  cv2.imshow("test", test_image)

 if interrupt & 0xFF == ord('a'):

    cv2.imwrite(directory+'A/'+str(count['a'])+'.jpg', roi)

  if interrupt & 0xFF == ord('b'):

    cv2.imwrite(directory+'B/'+str(count['b'])+'.jpg',  roi)

  if interrupt & 0xFF == ord('c'):

    cv2.imwrite(directory+'C/'+str(count['c'])+'.jpg',  roi)

  if interrupt & 0xFF == ord('d'):

    cv2.imwrite(directory+'D/'+str(count['d'])+'.jpg', roi)

  if interrupt & 0xFF == ord('e'):

    cv2.imwrite(directory+'E/'+str(count['e'])+'.jpg', roi)

  if interrupt & 0xFF == ord('f'):

    cv2.imwrite(directory+'F/'+str(count['f'])+'.jpg', roi)

  if interrupt & 0xFF == ord('g'):

    cv2.imwrite(directory+'G/'+str(count['g'])+'.jpg',  roi)

  if interrupt & 0xFF == ord('h'):

    cv2.imwrite(directory+'H/'+str(count['h'])+'.jpg',  roi)

  if interrupt & 0xFF == ord('i'):

    cv2.imwrite(directory+'I/'+str(count['i'])+'.jpg', roi)
```

```
if interrupt & 0xFF == ord('j'):
    cv2.imwrite(directory+'J/'+str(count['j'])+'.jpg', roi)
if interrupt & 0xFF == ord('k'):
    cv2.imwrite(directory+'K/'+str(count['k'])+'.jpg', roi)
if interrupt & 0xFF == ord('l'):
    cv2.imwrite(directory+'L/'+str(count['l'])+'.jpg', roi)
if interrupt & 0xFF == ord('m'):
    cv2.imwrite(directory+'M/'+str(count['m'])+'.jpg', roi)
if interrupt & 0xFF == ord('n'):
    cv2.imwrite(directory+'N/'+str(count['n'])+'.jpg',  roi)
if interrupt & 0xFF == ord('o'):
    cv2.imwrite(directory+'O/'+str(count['o'])+'.jpg',  roi)
if interrupt & 0xFF == ord('p'):
    cv2.imwrite(directory+'P/'+str(count['p'])+'.jpg',  roi)
if interrupt & 0xFF == ord('q'):
    cv2.imwrite(directory+'Q/'+str(count['q'])+'.jpg',  roi)
if interrupt & 0xFF == ord('r'):
    cv2.imwrite(directory+'R/'+str(count['r'])+'.jpg',  roi)
if interrupt & 0xFF == ord('s'):
    cv2.imwrite(directory+'S/'+str(count['s'])+'.jpg',  roi)
if interrupt & 0xFF == ord('t'):
    cv2.imwrite(directory+'T/'+str(count['t'])+'.jpg',  roi)
if interrupt & 0xFF == ord('u'):
    cv2.imwrite(directory+'U/'+str(count['u'])+'.jpg',  roi)
if interrupt & 0xFF == ord('v'):
    cv2.imwrite(directory+'V/'+str(count['v'])+'.jpg',  roi)
if interrupt & 0xFF == ord('w'):
        cv2.imwrite(directory+'W/'+str(count['w'])+'.jpg', roi)
    if interrupt & 0xFF == ord('x'):
        cv2.imwrite(directory+'X/'+str(count['x'])+'.jpg', roi)
    if interrupt & 0xFF == ord('y'):
        cv2.imwrite(directory+'Y/'+str(count['y'])+'.jpg', roi)
    if interrupt & 0xFF == ord('z'):
        cv2.imwrite(directory+'Z/'+str(count['z'])+'.jpg', roi)

cap.release()
cv2.destroyAllWindows()
```

Image_processing.py:

```
import numpy as np
import cv2
minValue = 70
def func(path):
    frame = cv2.imread(path)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)

    th3 = 
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2
)
    ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    return res
```

preprocessing.py:

```
import numpy as np

import cv2

import os

import csv

from image_processing import func

if not os.path.exists("data2"):

    os.makedirs("data2")

if not os.path.exists("data2/train"):

    os.makedirs("dataluv/train")

if not os.path.exists("data2/test"):

    os.makedirs("data2/test")

path="train"

path1 = "data2"

a=['label']


for i in range(64*64):

    a.append("pixel"+str(i))



#outputLine = a.tolist()



label=0

var = 0

c1 = 0

c2 = 0
```

```
for (dirpath,dirnames,filenames) in os.walk(path):
    for dirname in dirnames:
        print(dirname)
        for(direcpath,direcnames,files) in os.walk(path+"/"+dirname):
                if not os.path.exists(path1+"/train/"+dirname):
                os.makedirs(path1+"/train/"+dirname)
            if not os.path.exists(path1+"/test/"+dirname):
                os.makedirs(path1+"/test/"+dirname)
            # num=0.75*len(files)
            num = 100000000000000000
            i=0
            for file in files:
                var+=1
                actual_path=path+"/"+dirname+"/"+file
                actual_path1=path1+"/"+"train/"+dirname+"/"+file
                actual_path2=path1+"/"+"test/"+dirname+"/"+file
                img = cv2.imread(actual_path, 0)
                bw_image = func(actual_path)
                if i<num:
                    c1 += 1
                    cv2.imwrite(actual_path1 , bw_image)
                else:
                    c2 += 1
                    cv2.imwrite(actual_path2 , bw_image)


                i=i+1


        label=label+1
```

```
print(var)

print(c1)

print(c2)
```

train.py:

```
from keras.models import Sequential

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense , Dropout

import os

os.environ["CUDA_VISIBLE_DEVICES"] = "1"

sz = 128

# Step 1 - Building the CNN


# Initializing the CNN

classifier = Sequential()


# First convolution layer and pooling

classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1), activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolution layer and pooling

classifier.add(Convolution2D(32, (3, 3), activation='relu'))

# input_shape is going to be the pooled feature maps from the previous convolution layer

classifier.add(MaxPooling2D(pool_size=(2, 2)))

#classifier.add(Convolution2D(32, (3, 3), activation='relu'))

# input_shape is going to be the pooled feature maps from the previous convolution layer

#classifier.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# Flattening the layers

classifier.add(Flatten())


# Adding a fully connected layer

classifier.add(Dense(units=128, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=96, activation='relu'))

classifier.add(Dropout(0.40))

classifier.add(Dense(units=64, activation='relu'))

classifier.add(Dense(units=29, activation='softmax')) # softmax for more than 2


# Compiling the CNN

classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']) #
categorical_crossentropy for more than 2



# Step 2 - Preparing the train/test data and training the model

classifier.summary()

# Code copied from - https://keras.io/preprocessing/image/

from keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator(

        rescale=1./255,

        shear_range=0.2,

        zoom_range=0.2,

        horizontal_flip=True)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)


training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size=(sz, sz),
                                                 batch_size=10,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')


test_set = test_datagen.flow_from_directory('data/test',
                                            target_size=(sz , sz),
                                            batch_size=10,
                                            color_mode='grayscale',
                                            class_mode='categorical')
classifier.fit_generator(
        training_set,
        steps_per_epoch=500, # No of images in training set
        epochs=50,
        validation_data=test_set,
        validation_steps=4243)# No of images in test set



# Saving the model
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
print('Model Saved')
classifier.save_weights('model-bw.h5')
```

```
print('Weights saved')
```

app.py:

```
import cv2

import os

import numpy as np

from keras.models import model_from_json

import operator

import time

import sys, os

import matplotlib.pyplot as plt

from hunspell import Hunspell

from string import ascii_uppercase


class Application:

    def init (self):

        self.directory = './model/'

        self.hs = Hunspell()

        self.vs = cv2.VideoCapture(0)

        self.current_image = None

        self.current_image2 = None


        self.json_file = open(self.directory+"model-bw.json", "r")

        self.model_json = self.json_file.read()

        self.json_file.close()

        self.loaded_model = model_from_json(self.model_json)

        self.loaded_model.load_weights(self.directory+"model-bw.h5")
        self.json_file_dru = open(self.directory+"model-bw_dru.json" , "r")
        self.model_json_dru = self.json_file_dru.read()
        self.json_file_dru.close()
        self.loaded_model_dru = model_from_json(self.model_json_dru)
        self.loaded_model_dru.load_weights(self.directory+"model-bw_dru.h5")

        self.json_file_tkdi = open(self.directory+"model-bw_tkdi.json" , "r")
        self.model_json_tkdi = self.json_file_tkdi.read()
        self.json_file_tkdi.close()
        self.loaded_model_tkdi = model_from_json(self.model_json_tkdi)
        self.loaded_model_tkdi.load_weights(self.directory+"model-bw_tkdi.h5")

        self.json_file_smn = open(self.directory+"model-bw_smn.json" , "r")
        self.model_json_smn = self.json_file_smn.read()
        self.json_file_smn.close()
        self.loaded_model_smn = model_from_json(self.model_json_smn)
        self.loaded_model_smn.load_weights(self.directory+"model-bw_smn.h5")

        self.ct = {}
        self.ct['blank'] = 0
        self.blank_flag = 0
        for i in ascii_uppercase:
          self.ct[i] = 0
        print("Model Loaded")
        self.root = tk.Tk()
        self.root.title("Sign language Convertor to Text")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)

        self.root.geometry("900x1100")

        self.root['background']='#97c7f1'

        self.panel = tk.Label(self.root)

        self.panel.place(x = 120, y = 140, width = 655, height = 500)

        self.panel2 = tk.Label(self.root) # initialize image panel

        self.panel2.place(x = 460, y = 145, width = 310, height = 310)


        self.T = tk.Label(self.root)

        self.T.place(x=31,y = 17)

        self.T.config(text = "Sign Language to Text",font=("courier",20,"bold"),background='#90e0e0')

        self.panel3 = tk.Label(self.root) # Current SYmbol

        self.panel3.place(x = 1000,y=200)

        self.T1 = tk.Label(self.root)

        self.T1.place(x = 800,y = 200)

        self.T1.config(text="Character :",font=("Courier",20,"bold"),background='#90e0e0')

        self.panel4 = tk.Label(self.root) # Word

        self.panel4.place(x = 1000,y=250)

        self.T2 = tk.Label(self.root)

        self.T2.place(x = 800,y = 250)

        self.T2.config(text ="Word :",font=("Courier",20,"bold"),background='#90e0e0')

        self.panel5 = tk.Label(self.root) # Sentence

        self.panel5.place(x = 1000,y=300)

        self.T3 = tk.Label(self.root)

        self.T3.place(x = 800,y = 300)

        self.T3.config(text ="Sentence :",font=("Courier",20,"bold"),background='#90e0e0')
        self.T4 = tk.Label(self.root)

        self.T4.place(x = 800,y = 400)

        self.T4.config(text = "Suggestions",font = ("Courier",20,"bold"),background='#90e0e0')


        self.btcall = tk.Button(self.root,command = self.action_call,height = 0,width = 0)

        self.btcall.config(text = "About",font = ("Courier",14))

        self.btcall.place(x = 1100, y = 10)


        self.bt1=tk.Button(self.root, command=self.action1,height = 0,width = 0,bg='#90e0e0')

        self.bt1.place(x = 900,y=460)

        #self.bt1.grid(padx = 10, pady = 10)

        self.bt2=tk.Button(self.root, command=self.action2,height = 0,width = 0,bg='#90e0e0')

        self.bt2.place(x = 1100,y=460)

        #self.panel3.place(x = 10,y=660)

        # self.bt2.grid(row = 4, column = 1, columnspan = 1, padx = 10, pady = 10, sticky = tk.NW)

        self.bt3=tk.Button(self.root, command=self.action3,height = 0,width = 0,bg='#90e0e0')

        self.bt3.place(x = 900,y=530)

        # self.bt3.grid(row = 4, column = 2, columnspan = 1, padx = 10, pady = 10, sticky = tk.NW)

        self.bt4=tk.Button(self.root, command=self.action4,height = 0,width = 0,bg='#90e0e0')

        self.bt4.place(x = 1100,y=530)

        # self.bt4.grid(row = bt1, column = 0, columnspan = 1, padx = 10, pady = 10, sticky = tk.N)

        self.bt5=tk.Button(self.root, command=self.action5,height = 0,width = 0,bg='#90e0e0')

        self.bt5.place(x = 1000,y=600)

        # self.bt5.grid(row = 5, column = 1, columnspan = 1, padx = 10, pady = 10, sticky = tk.N)

        self.str=""

        self.word=""

        self.current_symbol="Empty"
```

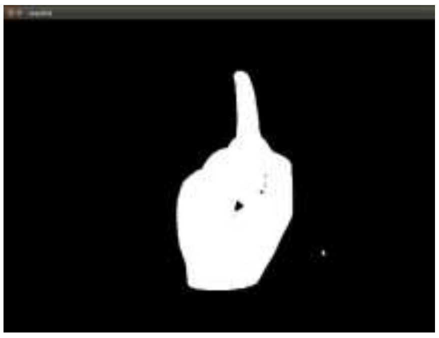
```
        self.video_loop()



    def video_loop(self):
        ok, frame = self.vs.read()
        if ok:
            cv2image = cv2.flip(frame, 1)
            x1 = int(0.5*frame.shape[1])
            y1 = 10
            x2 = frame.shape[1]-10
            y2 = int(0.5*frame.shape[1])
            cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255,0,0) ,1)
            cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
            self.current_image = Image.fromarray(cv2image)
            imgtk = ImageTk.PhotoImage(image=self.current_image)
            self.panel.imgtk = imgtk
            self.panel.config(image=imgtk)
            cv2image = cv2image[y1:y2, x1:x2]
            gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
            blur = cv2.GaussianBlur(gray,(5,5),2)
            th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2
)
            ret, res = cv2.threshold(th3, 70, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
            self.predict(res)
            self.current_image2 = Image.fromarray(res)
            imgtk = ImageTk.PhotoImage(image=self.current_image2)
            self.panel2.imgtk = imgtk
            self.panel2.config(image=imgtk)
            self.panel3.config(text=self.current_symbol,font=("Courier",20))
            self.panel4.config(text=self.word,font=("Courier",20))
            self.panel5.config(text=self.str,font=("Courier",20))
            predicts=self.hs.suggest(self.word)
            if(len(predicts) > 0):
                self.bt1.config(text=predicts[0],font = ("Courier",20))
            else:
                self.bt1.config(text="")
            if(len(predicts) > 1):
                self.bt2.config(text=predicts[1],font = ("Courier",20))
            else:
                self.bt2.config(text="")
            if(len(predicts) > 2):
                self.bt3.config(text=predicts[2],font = ("Courier",20))
            else:
                self.bt3.config(text="")
            if(len(predicts) > 3):
                self.bt4.config(text=predicts[3],font = ("Courier",20))
            else:
                self.bt4.config(text="")
            if(len(predicts) > 4):
                self.bt4.config(text=predicts[4],font = ("Courier",20))
            else:
                self.bt4.config(text="")
        self.root.after(10, self.video_loop)
    def predict(self,test_image):
        test_image = cv2.resize(test_image, (128,128))

        result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))

        result_dru = self.loaded_model_dru.predict(test_image.reshape(1 , 128 , 128 , 1))

        result_tkdi = self.loaded_model_tkdi.predict(test_image.reshape(1 , 128 , 128 , 1))

        result_smn = self.loaded_model_smn.predict(test_image.reshape(1 , 128 , 128 , 1))

        prediction={}

        prediction['blank'] = result[0][0]

        inde = 1

        for i in ascii_uppercase:

            prediction[i] = result[0][inde]

            inde += 1

        #LAYER 1

        prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

        self.current_symbol = prediction[0][0]

        #LAYER 2

        if(self.current_symbol == 'D' or self.current_symbol == 'R' or self.current_symbol == 'U'):

             prediction = {}

             prediction['D'] = result_dru[0][0]

             prediction['R'] = result_dru[0][1]

             prediction['U'] = result_dru[0][2]

             prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

             self.current_symbol = prediction[0][0]


        if(self.current_symbol == 'D' or self.current_symbol == 'I' or self.current_symbol == 'K' or
self.current_symbol == 'T'):

             prediction = {}

             prediction['D'] = result_tkdi[0][0]

             prediction['I'] = result_tkdi[0][1]
```

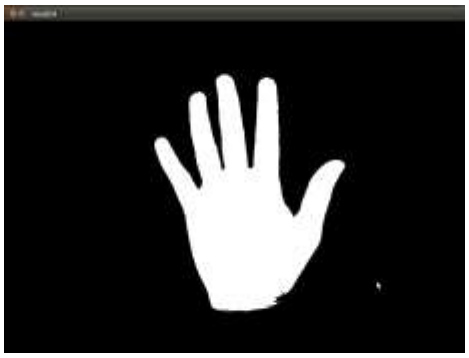
```
    prediction['K'] = result_tkdi[0][2]

    prediction['T'] = result_tkdi[0][3]

    prediction = sorted(prediction.items(), key=operator.itemgetter(1), reverse=True)

    self.current_symbol = prediction[0][0]


if(self.current_symbol == 'M' or self.current_symbol == 'N' or self.current_symbol == 'S'):

    prediction1 = {}

    prediction1['M'] = result_smn[0][0]

    prediction1['N']  = result_smn[0][1]

    prediction1['S'] = result_smn[0][2]

    prediction1 = sorted(prediction1.items(), key=operator.itemgetter(1), reverse=True)

    if(prediction1[0][0] == 'S'):

            self.current_symbol = prediction1[0][0]

    else:

            self.current_symbol = prediction[0][0]

if(self.current_symbol == 'blank'):

   for i in ascii_uppercase:

      self.ct[i] = 0

self.ct[self.current_symbol] += 1

if(self.ct[self.current_symbol] > 60):

   for i in ascii_uppercase:

      if i == self.current_symbol:

         continue

      tmp = self.ct[self.current_symbol] - self.ct[i]

      if tmp < 0:

         tmp *= -1

      if tmp <= 20:
```

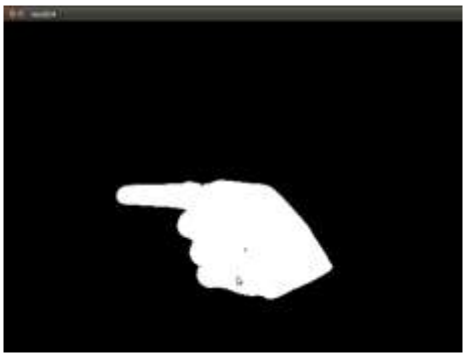
```
                self.ct['blank'] = 0
                for i in ascii_uppercase:
                    self.ct[i] = 0
                return
        self.ct['blank'] = 0
        for i in ascii_uppercase:
            self.ct[i] = 0
        if self.current_symbol == 'blank':
            if self.blank_flag == 0:
                self.blank_flag = 1
                if len(self.str) > 0:
                    self.str += " "
                self.str += self.word
                self.word = ""
        else:
            if(len(self.str) > 16):
                self.str = ""
            self.blank_flag = 0
            self.word += self.current_symbol
def action1(self):
          predicts=self.hs.suggest(self.word)
          if(len(predicts) > 0):
        self.word=""
        self.str+=" "
        self.str+=predicts[0]
def action2(self):
          predicts=self.hs.suggest(self.word)
          if(len(predicts) > 1):
        self.word=""
        self.str+=" "
        self.str+=predicts[1]
def action3(self):
          predicts=self.hs.suggest(self.word)
          if(len(predicts) > 2):
        self.word=""
        self.str+=" "
        self.str+=predicts[2]
def action4(self):
          predicts=self.hs.suggest(self.word)
          if(len(predicts) > 3):
        self.word=""
        self.str+=" "
        self.str+=predicts[3]
def action5(self):
          predicts=self.hs.suggest(self.word)
          if(len(predicts) > 4):
        self.word=""
        self.str+=" "
        self.str+=predicts[4]
def destructor(self):
    print("Closing Application...")
    self.root.destroy()
    self.vs.release()
    cv2.destroyAllWindows()
    def destructor1(self):

        print("Closing Application...")

        self.root1.destroy()


    def action_call(self) :


        self.root1 = tk.Toplevel(self.root)

        self.root1.title("About")

        self.root1.protocol('WM_DELETE_WINDOW', self.destructor1)

        self.root1.geometry("900x900")


        self.tx = tk.Label(self.root1)

        self.tx.place(x = 330,y = 20)

        self.tx.config(text = "CREATED BY", fg="blue", font = ("Courier",30,"bold"))

        self.photo3 = tk.PhotoImage(file='Pictures/luv.png')

        self.w3 = tk.Label(self.root1, image = self.photo3)

        self.w3.place(x = 380, y = 105)

        self.tx3 = tk.Label(self.root1)

        self.tx3.place(x = 380,y = 250)

        self.tx3.config(text = "FARAZ AHMAD\n", font = ("Courier",15,"bold"))


print("Starting Application...")

pba = Application()

pba.root.mainloop()
```

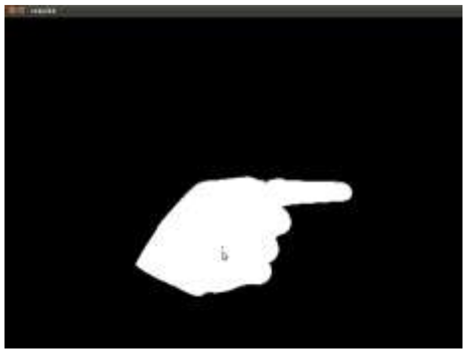
Testing of Program Functions Using Static Images

Figure shows all the chosen gestures with their corresponding control commands for the robot after colour segmentation into skin colour and black.

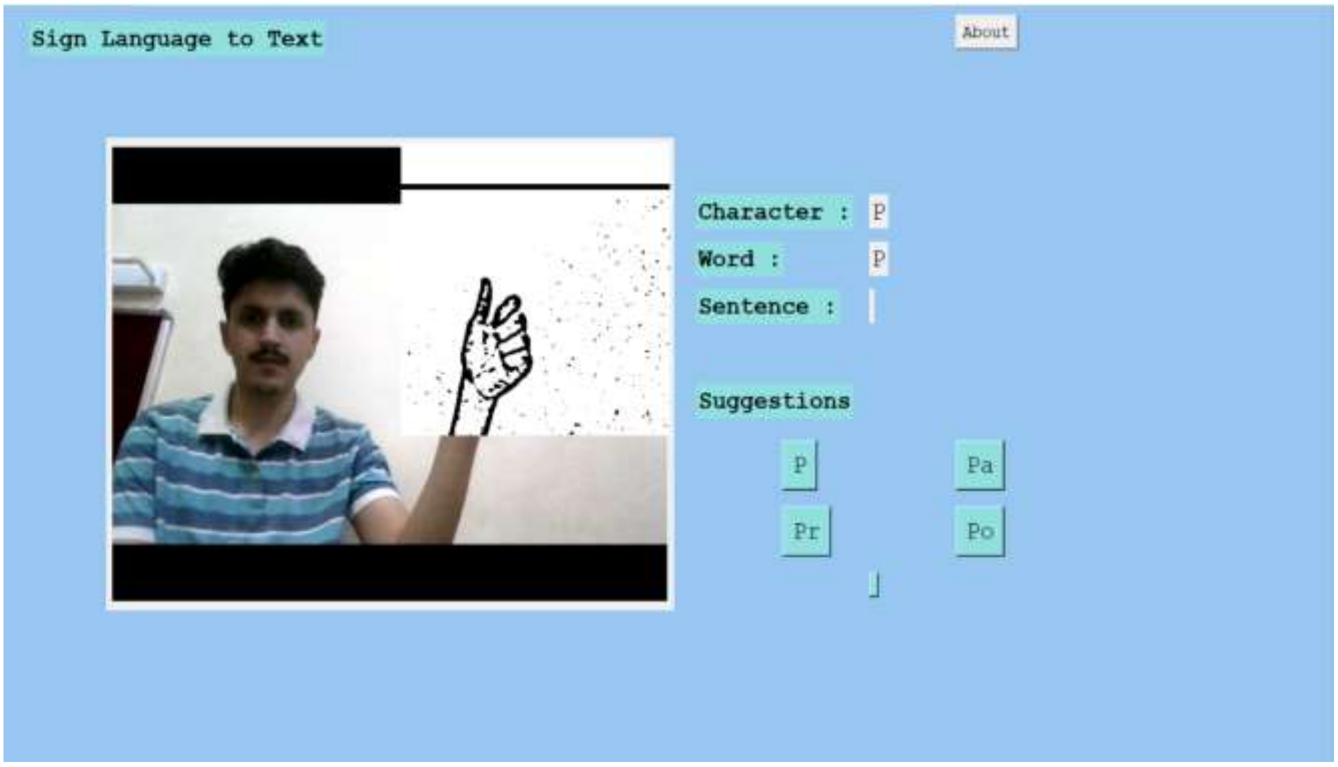


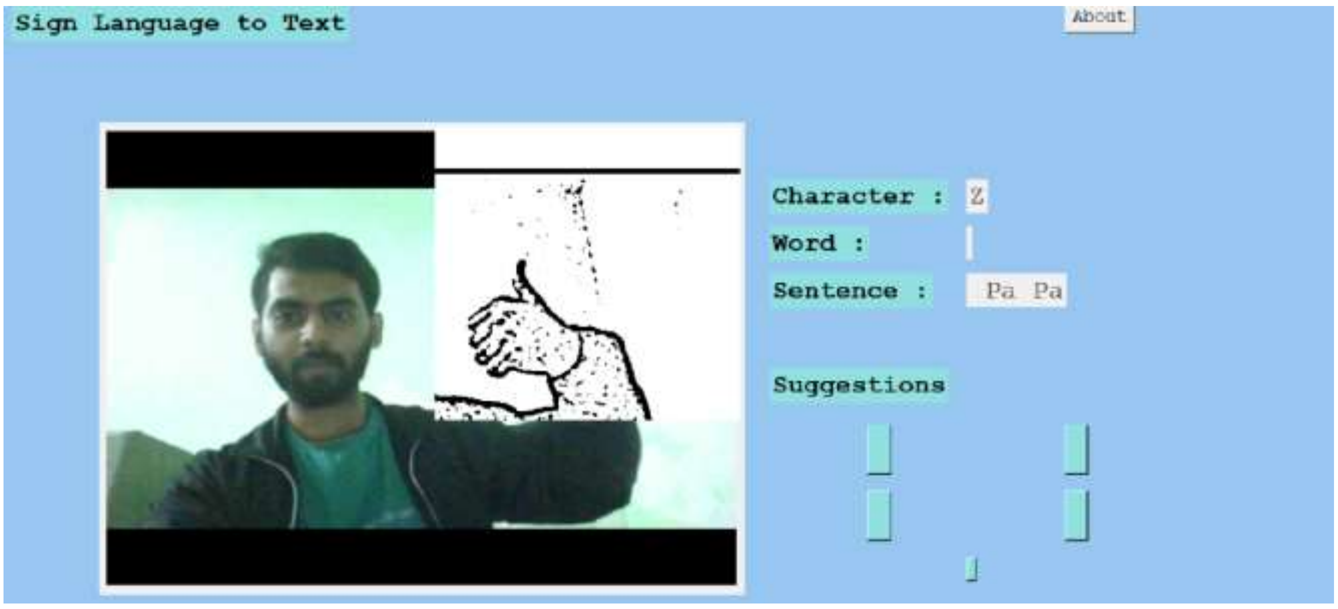






OUTPUT


Sign Language to Text
About
Character : P
Word : P
Sentence :
Suggestions
P
Pa
Pr
Po


Sign Language to Text
About
Character : Z
Word :
Sentence : Pa Pa
Suggestions

Discussion and Conclusion

The results also showed that the gesture recognition application was quite robust for static images.

However, the video version was enormously affected by the amount of illumination, such that is was necessary to check and adjust the HSV values for skin color when starting the program to get the proper output. Sometimes the adjustment was difficult to do because of the lighting conditions and the number of objects in the background.

The application was very susceptible to noise on the video stream. Slight hand movements could affect gesture recognition. Nevertheless, if the hand is steady enough for long enough, the program outputs the correct command.

For integrating the program with the robot in the future, it would be necessarily to consider other output such as speed or velocity as part of the navigational control commands. Based on the results, a computer vision application could detect and recognize simple hand gestures for robot navigation using simple heuristic rules. While the use of moment invariants was not considered suitable because the same gestures could be used pointing in opposite directions, other learning algorithms like AdaBoost could be explored to make the program more robust and less affected by extraneous objects and noise.

FUTURE SCOPE

There are some aspects of projects which can be improved in future.

- Instead of webcam a better and more accurate acquisition device can be used which even used infrared for accuracy e.g., Kinect.
- Mechanism for hand detection is not accurate.
- HU set of invariant moments are very basic descriptors as features of image which will not have good accuracy. A better descriptor can give good results but classification mechanism may change.

REFERENCES

[1] Robust Part-Based Hand Gesture Recognition Using Kinect Sensor Zhou Ren, Junhong Yuan, Member, IEEE, Jingjing Meng, Member, IEEE, and Zheng you Zhang, Fellow, IEEE, 15, AUGUST 2013.

[2] A Fast Gesture Recognition Scheme for Real-Time Human-Machine Interaction Systems. Ching-Hao Lai* Smart Network System Institute for Information Industry Taipei City, Taiwan, 2010.

[3] Real-Time Hand Gesture Recognition System for Daily Information Retrieval from Internet.

IEEE Fourth International Conference on Ubi-Media Computing

[4] Wearable Sensor-Based Hand Gesture and Daily Activity Recognition for Robot_Assisted Living IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART A: SYSTEMS AND HUMANS, VOL. 41, NO. 3, MAY 2011.