Real Time Morse Video to Audio Decoder

Project Report: Digital Signal Processing Lab, Fall 2019

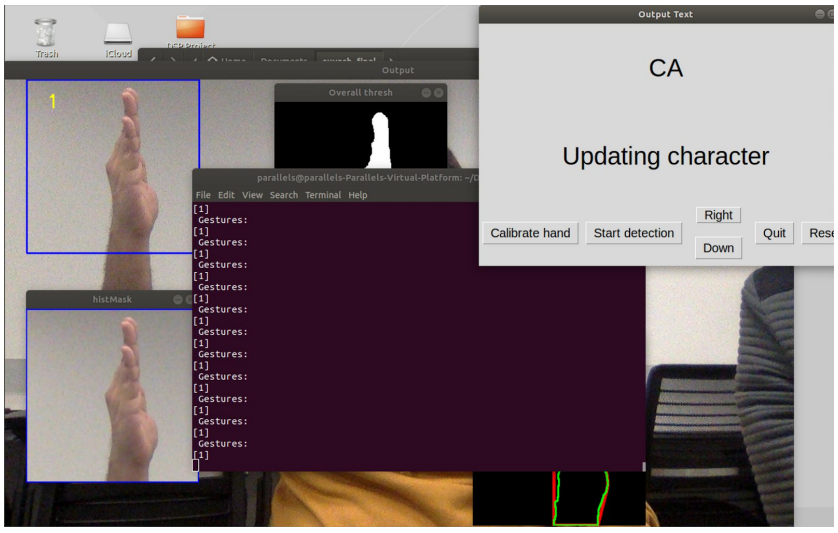
Akhil Wadhwa (aw3509), Suyash Sule (ss11524) and Sagarika Limaye (sl7241)

Final year MS students, Electrical and Computer Engineering, NYU Tandon

Introduction

Morse code is a famous character encoding scheme, popularly used for representing letters and digits in the form of sequences of dots and dashes. Morse telegraphs were extensively used to communicate messages before the advent of radio communication systems. These machines were composed of receivers that made short and long clicking noises corresponding to dots and dashes.

In this project, we make use of the Morse code to decode alphabets that are communicated by the user. The user can provide a visual input of the Morse code using their hand by showing 0 (fist) for a dot or 1 (a finger or side of the palm) for a dash, thus providing a sequence of dots and dashes. This sequence of hand gestures is detected by the program and decoded into the corresponding letters. A 4 or 5 in user input indicates character break to initiate Morse decoding of the most recent sequence of dots and dashes and two 4s or 5s in a row indicate a word break. The program plays audio of the decoded letter after a character break and also says the entire word on a word break.



This project can find application in human computer interaction, robot remote control, visual communication and music creation among others. It can be used for helping blind and mute people communicate their thoughts using simple hand gestures and receive an audio feedback.

List of Libraries

1. OpenCV (cv2): to work with real time video input and image processing techniques
2. Math: for math functions like cos, sin, pi
3. Numpy: for array manipulations, matrix operations
4. Pyautogui: to reduce lag between input and video output
5. Statistics: to find mode of an array of numbers
6. Wave: to read .wav audio files and play audio of the decoded letter
7. Pyaudio: to output the audio frames to the audio stream
8. Pygame: to play an audio file directly
9. gTTS: Google text-to-speech library to play the audio of an entire word
10. Tkinter: to create, modify and handle the GUI

List of Functions:

In-built functions

1. Tk.Tk( ): Tkinter root
2. root.title( ): set title of GUI window
3. root.geometry( ): set dimensions of GUI window
4. Tk.StringVar( ): define string variables in GUI
5. Tk.Label( ): define GUI label
6. Label.pack( ): finalize the position, font, size of GUI element
7. Tk.Button( ): used to implement button widget using Tkinter
8. cv2.VideoCapture(0): open camera at index 0 and start video input
9. capture.read( ): read video frame image
10. cv2.flip(frame): mirror image flipping of the video
11. cv2.putText( ): display text on video frame
12. list.append( ): append element to list
13. .set( ): set variable, parameter value
14. root.update( ): update the values of GUI
15. mode( ): find most frequent element in the list
16. gTTS.save(text, lang, slow): save audio file of text of specified language on local disk
17. pygame.mixer.init( ): initialize pygame reader
18. mixer.music.load(filename): load audio from specified file
19. mixer.music.play( ): play the audio in the loaded file
20. cv2.imshow( ): display video in the video frame

21. root.mainloop( ): is an infinite loop used to run the application and wait for an event to occur
22. cv2.destroyAllWindows( ): close video frames
23. frame.shape( ): dimensions of video frame
24. cv2.cvtColor( ): convert RGB to HSV color representation
25. np.zeros( ), np.ones( )
26. cv2.calcHist( ): creates a histogram of the color image matrix
27. cv2.normalize( ): normalize values in the histogram to [0,1]
28. cv2.rectangle( ): draw rectangle on video frame
29. cv2.threshold( ): convert the image to a 0-and-1 image using a mask
30. max( )
31. tuple( ): convert list/array to tuple
32. cv2. contourArea( ): find the area of the contour
33. cv2.moments( ): to find centroid of the object in image
34. math.sqrt( ), math.acos( ): finds cosine inverse and returns the angle
35. cv2.findContours( ): finds endpoints of contours using a 0-1 threshold image
36. cv2.convexHull( ): finds convex hull endpoints of the contours
37. cv2.convexityDefects( ): finds endpoints of the convexity defects using contour and hull
38. cv2.calcBackProject( ): finds the projection of hand histogram on hand mask
39. cv2.getStructuringElement( ): used to create elliptical/circular shaped kernels
40. cv2.filter2D( ): used to convolve the kernel formed with the image
41. cv2.morphologyEx( ): used to perform advanced morphological transformations such as close, open, dilate as well as gradient and tophat.
42. cv2.merge( ): merges several single-channel arrays into a multi-channel array
43. cv2.bitwise_and( ): as the name suggests, finds the bitwise-and of the 2 input arrays
44. cv2.bilateralFilter( ): applies bilateral filtering to the input image as well as reduces the unwanted noise while keeping edges fairly sharp.
45. cv2.drawContours( ): draws contours of any shape when provided with boundary points
46. cv2.circle( ): used to draw a circle on any image
47. cv2.line( ): used to draw a line segment connecting two points on any image
48. cv2.createBackgroundSubtractorMOG2( ): Gaussian mixture-based background/foreground segmentation. MOG2 has capability to detect shadows
49. Tk.Tk( ).destroy( ): terminates the mainloop and deletes all widgets
50. cv2.release( ): to close the already opened file or camera

User defined functions

1. createHandHistogram( ): creates H-V histogram of calibration hand by converting the RGB hand image to HSV and calculates the histograms of H and V channels
   Input arguments: video frame
   Returns: normalized histogram

2. drawRect( ): draws the 3x3 grid of green rectangles used to calibrate color of hand
   Input arguments: video frame
   Returns: null
3. threshold( ): converts RGB hand mask image to a binary 0-1 threshold image
   Input arguments: hand mask
   Returns: thresholded binary image
4. getextreme( ): finds the topmost point of the hand (usually the tip of middle finger)
   Input arguments: hand contour points
   Returns: extreme top point,
5. getMaxContours( ): finds max of contours
   Input arguments: hand contours
   Returns: max of contours
6. setupFrame( ): defines size and initial position of the ROI (blue box in video frame)
   Input arguments: frame width and height
   Returns: x,y coordinates of bottom left corner of ROI, height and width of ROI
7. getCentroid( ): finds the centroid of the input shape contours
   Input arguments: hand contours
   Returns: x,y coordinates of the centroid
8. calculateAngle( ): calculates the angle between fingers (convexity defect) using cosine rule
   Input arguments: far point, end point and start point
   Returns: angle between the fingers
9. countFingers( ): counts the number of fingers of hand using angle of convexity defects
   Input arguments: contours and convex hull
   Returns: number of detected fingers
10. histMasking( ): finds the hand mask using the hand histogram
    Input arguments: ROI frame, hand histogram
    Returns: hand mask
11. detectHand( ): makes use of all functions to detect hand and count fingers
    Input arguments: video frame, hand histoframe, Bgsubtractor image, x,y coordinates of ROI, height and width of ROI
    Returns: contours, distance between the centroid and top extreme point, len(contours)
12. MorseDetector( ): decodes the gestures to find the corresponding letter, also plays the letter audio and handles erroneous gestures
    Input arguments: array of gestures to be decoded
    Returns: decoded letter
13. calibrate_hand( ): responds to the Calibrate Hand button click on GUI
    Input arguments: null
    Returns: null
14. start_detection( ): responds to the Start Detection button click on GUI
    Input arguments: null
    Returns: null
15. reset_capture( ): resets the background calibration when the GUI Reset button is clicked

Input arguments: null
Returns: null

16. quit_prog( ): terminates all windows and program
    Input arguments: null
    Returns: null
17. turnDown( ): moves the blue box down using GUI
    Input arguments: null
    Returns: null
18. turnUp( ): moves the blue box up using GUI
    Input arguments: null
    Returns: null
19. turnLeft( ): moves the blue box left using GUI
    Input arguments: null
    Returns: null
20. turnRight( ): moves the blue box right using GUI
    Input arguments: null
    Returns: null

Working Details

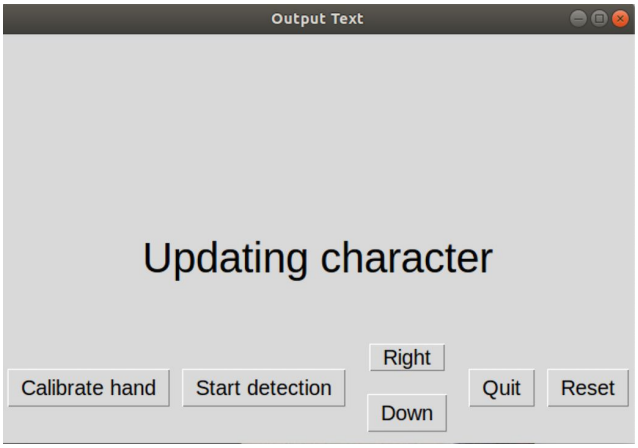
The project primarily makes use of OpenCV for hand gesture recognition from video input. The finger/hand detection module was inspired from Chin Huan's finger detection project [1], [2]. The flow of the code starts from 'main'. The working of the program can be divided into the following sections:

1. Video Capture:

This starts the webcam and opens a window showing the captured video. The region of interest (ROI) is drawn on the video in the form of a blue rectangle, indicating the region of video that will be used for user input/gesture recognition. The program then draws a 3x3 grid of green rectangles which are used to calibrate the color of hand.

2. GUI

The program also opens another window for the graphical user interface (GUI). The GUI window consists buttons to regulate the program controls. The first button is 'Calibrate Hand' to capture the color of the hand after the hand is aligned along the grid of green rectangles. The 'Start Detection' button completes the initial process of calibration by capturing the background in the blue rectangle. This static background can then be used for background subtraction.



The GUI also has a set of up, down, left and right buttons to position the blue box on the video screen. In this way, the user can set the position of the ROI as per their comfort and convenience. The 'Reset' button to reset the background calibration which can be used if the original background changes drastically. The 'Quit' button terminates the program.

The GUI also shows the decoded letters. Before detecting a hand, the GUI says ‘Character Please!’. On detecting a hand, the GUI says ‘Updating Character!’ and if an incorrect gesture is detected, the GUI throws the message ‘Invalid Character!’.

3. HSV Segmentation:

In order to detect the hand, we will be using the HSV (Hue, Saturation and Value) segmentation. Hue determines the colour and Saturation describes the intensity of that color. Finally, Value tells the brightness or lightness. Since the HSV captures gives better information about the brightness for each color, we chose to use HSV instead of RGB. Another point to note is that in order to sample the color of the hand, we used only the Hue and saturation parameters and not Value because the model might misbehave if the brightness of the captured hand is uneven.

As soon one place the hand in the frame and hit "Calibrate hand", a few samples of the hand are taken and a histogram to determine the frequency of each color as it appears in the samples taken is formed. In order to find the probability of each frequency being a part of the hand, we normalize the histogram. Using this normalized histogram, we create a mask which contains the probability of that pixel being a part of the hand.

4. Creating BgSubmask and Threshold:

Consecutively, we perform background subtraction to get a clear image of the hand. After subtraction is complete, we use the following functions to optimize the image:

```
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel, iterations=2)
fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel, iterations=2)
```

where cv2.MORPH_CLOSE closes the holes in the image and cv2.MORPH_OPEN to remove the noise in the background.

Next, we create a background of the original ROI and bitwise-and mask to get the bgSubMask.

Finally, we create a threshold of the mask to get the grayscale image of the hand in the frame.

5. Creating Hand Contours

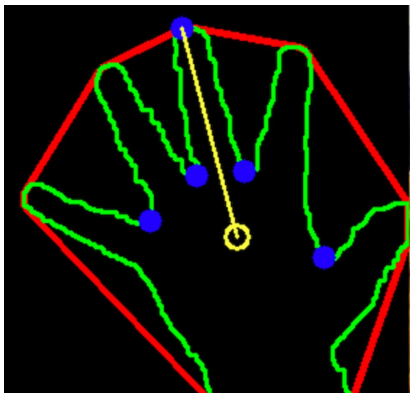
If any object is present in the region of interest the findContours function returns the contours else it is empty.

```
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

We use this to find if the hand is present or not.
Once we place the hand in the ROI we use the contours to get the maximum contours to find the convex hull by

```
maxContour = getMaxContours(contours)
hull = cv2.convexHull(maxContour)
```

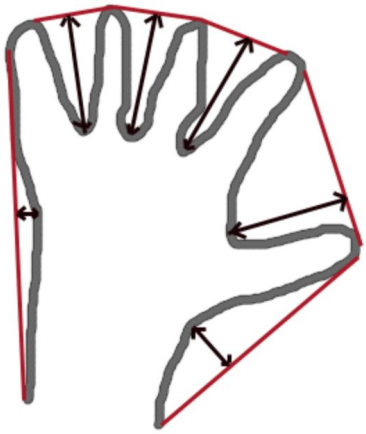
Convex hull:
The convex hull is a convex closure of a set *X* of points in the Euclidean plane or in a Euclidean space. That is given the maximum contours it will return the convex hull of hand as indicated by red in the image below.



We use this convex hull to count the number of fingers

6. Counting Fingers (Gesture Recognition):

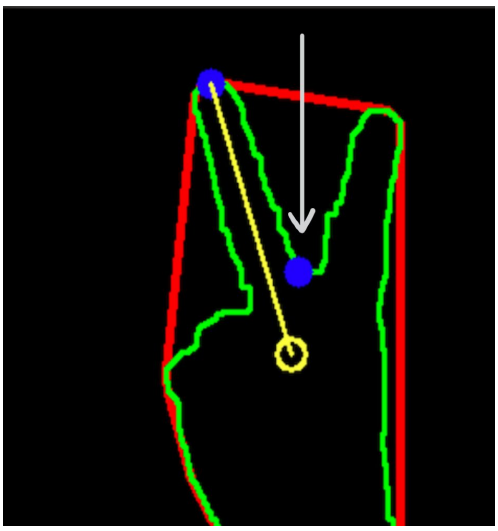
We use the convex hull to find the convexity defects.
Convexity Defects:



Convexity defect is a cavity in an object (blob, contour) segmented out from an image. That means an area that do not belong to the object but located inside of its outer boundary.

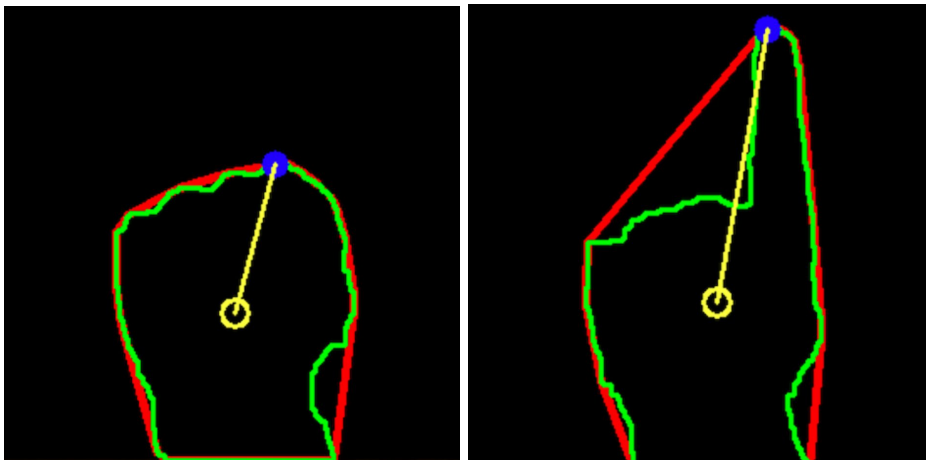
We find the convexity detects as:
defects = cv2.convexityDefects(contour, hull)

To detect the number of fingers we put conditions on the convexity defects. We ignore the defects that have a length of less than 10000 or defects that make an angle of more than 90 degrees.



In the above images we can see that the convexity defect in the first image is indicated by arrow and based on that in the second image we can see that the number of fingers indicated in yellow displays 2.

Detection between 0 and 1



As we can see in the above images for both the number on convexity defects is 0. Therefore, here we use the distance between the centroid (shown in yellow) and the extreme point (shown in blue) as the distinguishing factor to indicate whether it is a zero or one.

centroid and extreme points can be calculated by using functions
centroid = getCentroid(maxContour)
extTop, c = getextreme(contours)

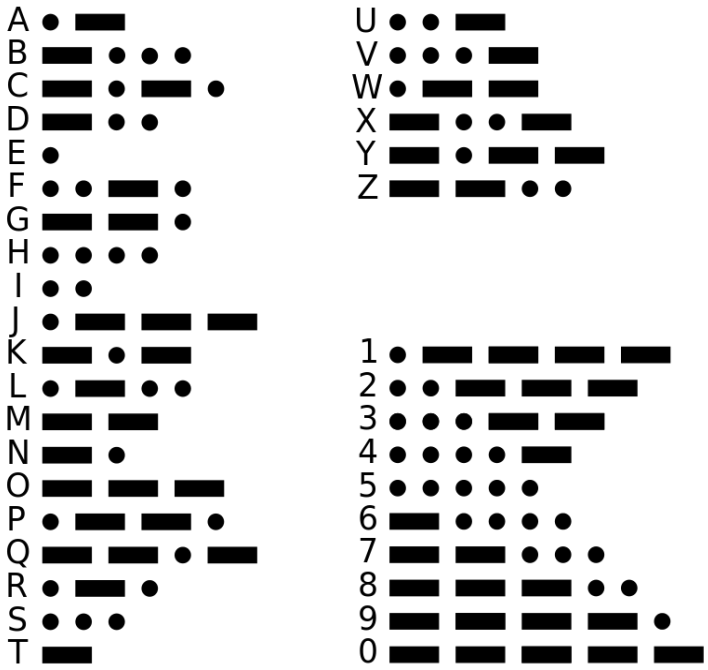
We have calculated the distance between these points and set a threshold to differentiate between zero and one.

7. Gesture Detection:

The program first checks whether a hand is being detected in the video frame or not. When the hand is not being detected, the hand_flag is set to 0. The append_flag counts the number of frames/iterations completed by the program after detecting a hand. This makes the program robust to inaccurate transient detections when the hand is arriving into the ROI. When the append_flag hits 31, i.e. 31 frames of detected hand have been captured, the program finds the gesture that is detected most frequently using mode(). This gesture corresponding to the mode is then decided to be the correctly detected gesture and is stored in the gestures[ ] list. Once the hand moves out of the ROI, the gestures_list is emptied to prepare for the next gesture and append_flag is also reset to 0.

When a 4 or a 5 is detected, it indicates a character break and all the gestures stores in the gestures[ ] list is sent to the Morse decoder and the corresponding letter is obtained. The word_flag is set to 1 when two consecutive 4s or 5s are detected, indicating a word break and the entire word is then converted to an audio file using Google's text-to-speech library. The play_flag indicates whether the audio has already been played or not to ensure that the letter/word audio is played only once.

8. Morse Decoding:



For Morse code detection, we call the function MorseDetector( ) which takes in the sequence of gestures as argument. We first convert the input gesture array into a string which is then looked up in a dictionary which stores the Morse codebook for all the characters and the corresponding audio file of the letter. The Morse code is stored as a key and the corresponding letter and the audio file to be played are stored at the key location. In order to make sure that the program does not terminate in case an invalid character is detected, we implemented the try and except clause, which plays the corresponding audio file when the correct string is detected, or throws the 'Invalid Character' message otherwise. In order to play the audio file, we use Pygame instead of Pyaudio as we faced some version issues in the Ubuntu OS and a mismatch between the audio driver and output stream. However, a combination of Pyaudio and Wave libraries can also be used to read the .wav audio file and play it on the audio stream.

9. Error Detection:

   If the user gives an invalid gesture that does not belong to the Morse code dictionary, instead of termination due to a runtime error, the program will discard the invalid character and ask for another character. We implemented this functionality using the try and except clause.

   Similarly, if there is a noisy detection while detecting a gesture it discards all the recorded gestures and asks for a new gesture. This was also implemented using the try and except clause.

Future Scope

Some of the limitations of this project include the following and working on them can further enhance its performance:

1. Detection of multiple hands: the program is designed to handle gestures from a single hand.
2. Gesture change detection: the program does not react to a change in gesture in the ROI. It instead prepares for the next gesture when the hand moves out of the ROI box. Instantaneous change in gesture detection can help improve user convenience.
3. Morse encoding: a more efficient prefix-free encoding scheme can be designed for faster encoding and decoding.
4. Digit recognition: only letters are part of the codebook dictionary and it can be easily extended to include digits.
5. Reading the entire sentence/paragraph: the program outputs only the individual letters and words to the audio stream. The code can be easily extended to output the sentence and the paragraph using the gTTS library.

References

1. Finger detection project by Chin Huan on Github: https://github.com/ChinHuan/finger-detection
2. Finger detection project summary: https://becominghuman.ai/real-time-finger-detection-1e18fea0d1d4
3. Wikipedia on Morse code: https://en.wikipedia.org/wiki/Morse_code
4. OpenCV documentation: https://docs.opencv.org/master/d9/df8/tutorial_root.html