Assignment 1 Report - Static Hand Recognition

[EIC0104] Computer Vision
Carlos Miguel Ferreira Lucas - up201000717
Diogo Henrique Marques Cruz - up201105483

Faculdade de Engenharia da Universidade do Porto (FEUP),
MIEIC
02/11/2018

1 Abstract

Detection of content in a video is a simple idea, but with a hard correct implementation. In this report we develop an effective method of detecting and tracking hands in uncontrolled webcam feed based on multiple cues including hand shape and skin color. We apply our hand detection results to perform fine-grained human hand recognition. Experimental results show the effectiveness of our approach.

2 Introduction

The motivation for this study is to develop a realtime, low cost, vision based hand gesture recognition system that works precisely on a relatively small restricted gesture space in such an environment that the lighting is relatively stable and the background is not complex.

3 Implementation

Our implementation has two phases:

1) Color calculator where we determine the skin color;
2) Hand video detection where the hand signal is analysed.

3.1 Color Calculator

In order to detect the hand, it is needed to calculate the color of the hand in that ambient. Our solution is to ask the user to place the hand in a square, our region of interest, to take a sample of the skin color. With that sample we extract the minimum, maximum and average of each value of the HSV color space (hue, saturation, and value). The average will be used as the color to search for in the Hand Video Detection phase. The minimum and maximum are used to determine a maximum value of sensitivity the user can allow. A value of 0% sensitivity means that the user wants to identify only the color identified in the average, while a value of 100% sensitivity means the user allows the application to search for a range of colors between the minimum and the maximum identified. By reducing the sensitivity, the user is able to reduce the amount of noise in the image, which can be caused by lighting or by objects with similar colors, and adapt the application to the conditions of his location. Reducing the sensitivity to values close to 0% may cause the application to stop recognizing the user's hand, so searching for a more stable value of sensitivity is recommended.

3.2 Hand Video Detection

In this proposed method, firstly RGB images are captured by the camera, but the RGB color space is not adequate to compare colors, so the color space is changed to HSV. The HSV representation models the way paints of different colors mix together, with the saturation dimension resembling various shades of brightly colored paint, and the value dimension resembling the mixture of those paints with varying amounts of black or white paint.[1] Before finding the bounding box we have applied some of noise elimination steps such as threshold, dilation, erosion, and blur.

1. In threshold, we threshold the image with the skin color values to only get the defined color of the hand;

2. In dilation, we extrapolate the hand to fill dark spots within by increasing the white area;

3. In erosion, because the object was expanded in the dilation phase, we need to shrink it to have a better notion of the contour of the hand;

4. In blur, by convolving the image with a low-pass filter kernel, we remove noises. We use Gaussian blur, as it is highly effective in removing gaussian noise and is not heavy, but the edges are blurred a little bit in this operation[2]

After processing the image, the contours are calculated. Contours can be explained simply as a curve joining all the continuous points, along the boundary, having same color or intensity. The contour with biggest area should be the hand to be detected. Then we do an approximation of the contours by using Ramer-Douglas-Peucker algorithm. That iterative end-point fit algorithm is an algorithm that decimates a curve composed of line segments to a similar curve with fewer points.[3]

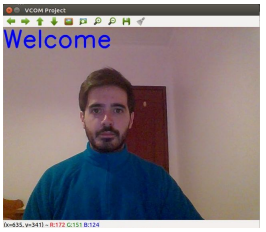
Then we make a convex hull around the hand to detect any convexity defects, that means, all the areas that do not belong to the object but are located inside of its convex hull. The convexity defects in this case are recognized as the space between fingers. The space between fingers is inside the convexity hull, but doesn't match the hand color.

Then, by analysing the number of defects, the area of the convexity defects, the area of the hull, and the ratio between the area of hand and hull we can determinate what signal is the hand doing.

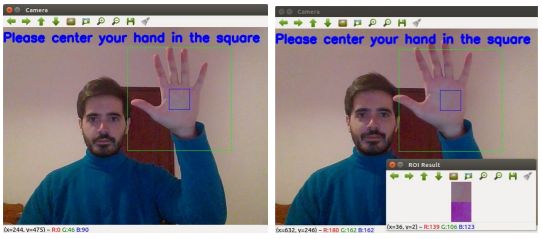
4 Application

This application was built using OpenCV with Python. The application is executed with the command `python hand_detection.py`. There are three arguments that can be passed: `--left` to show the ROI on the left (by default is on the right side), `--shot` to take an image from the webcam instead of its video, `--input=<path>` to analyse a picture instead of the webcam feed.

When the app starts, a window with a "welcome" message is shown:

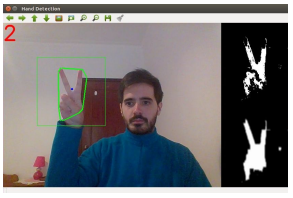


The user can then press "v" to start the color calculator. Here, a blue square is shown inside a green square. The user has to place his hand inside the blue square, in order to calculate the skin color, and press "v" again as shown:

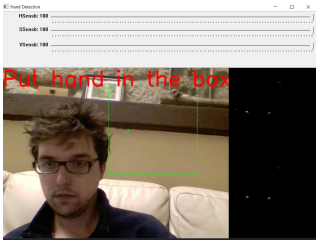


If the ROI doesn't satisfy the user he can press "n" to redo the capture. If any other key is pressed, the app starts the hand video detection.

Where the hand is detected and the signal it is doing is written on the image. The capture in the range of skin color is shown on the top-right side, and on the bottom-right side is shown the image that is being analysed after applying the optimizations mentioned in 3.2.



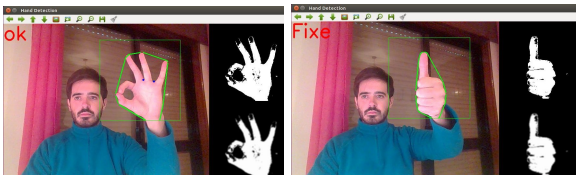
The user can change the sensibility of each value of HSV color range to detect, in a range from 0% to 100%, using 3 trackbars on the top of the screen, as seen in the image below.



5 Results

It can detect a hand doing each number from 0 to 5:

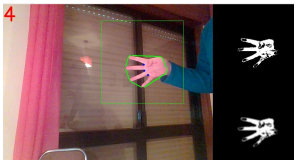
It can also detect some gestures as “ok” and “alright/fixe”:



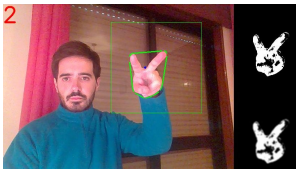
The application can work with a bad image where the hand has a lot of noise and a lot of black holes, which means that the hull doesn’t need to be perfect, it as a high range of bad conditions where it works well. An example is shown:

There can be more than one hand in the image. Only the hand covering the most area within the green square will be analysed:

The hand on image passed to the application can be on an horizontal pose:



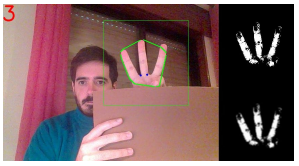
The palm of the hand doesn’t need to be in frontal view:



The background doesn’t need to be constant. It can detect with a lot of similar color on the ROI, and natural light:

The hand doesn’t need to occupy the most of the screen. It can be small as long as the camera detects it:

As long as the fingers are shown, the hand can be partially occulted:



6 Conclusions

The requirements were all met. The app is lightweight, it doesn’t have any heavy algorithm, and as such it is fast to calculate colors and detect the hand signal. OpenCV and Numpy were the only external libraries of Python used, so it is easy to maintain. The app is ready to use straight out of the box.

On the down side, there are some environments where the app doesn’t handle as well as expected. Sometimes is hard to get a good color bound on the environment. If there is too much sunlight, it interferes as it isn’t as regular as artificial light. As the signals are being calculated with areas, if the detection of the color is bad, the signals will have a worse performance. If the environment has colors that are matched with the skin color, there can occur that those areas merge with the hand and it makes impossible for the algorithm to calculate the signal.

7 References

[1]“HSL and HSV,” *Wikipedia*. 12-Oct-2018.
[2]“Gaussian blur,” *Wikipedia*. 15-Aug-2018.
[3]“Ramer–Douglas–Peucker algorithm,” *Wikipedia*. 13-Oct-2018.
[4]“Hand Gesture Recognition for Human-Machine Interaction. | Request PDF,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/221546357_Hand_Gesture_Recognition_for_Human-Machine_Interaction. [Accessed: 01-Nov-2018].
[5]A. Birdal and R. Hassanpour, *Region Based Hand Gesture Recognition*. Václav Skala - UNION Agency, 2008.
[6]M. Panwar and P. S. Mehra, “Hand gesture recognition for human computer interaction,” in *2011 International Conference on Image Information Processing*, 2011, pp. 1–7.
[7]“A novel finger and hand pose estimation technique for real-time hand gesture recognition | Request PDF,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/282487477_A_novel_finger_and_hand_pose_estimation_technique_for_real-time_hand_gesture_recognition. [Accessed: 01-Nov-2018].
[8]S. S. Rautaray and A. Agrawal, “Vision based hand gesture recognition for human computer interaction: a survey,” *Artif. Intell. Rev.*, vol. 43, no. 1, pp. 1–54, Jan. 2015.

8 Annexes

8.1 hand_detection.py

```
#!/usr/bin/env python

import numpy as np
import cv2
import argparse
import color_calculator as cc
import color_detection as cd
import video_detection as vd


def analyse_args():
   """Parses the args"""
   parser = argparse.ArgumentParser()
   parser.add_argument(
       '-i', '--input', default=None, help='Place ROI on
the left')
   parser.add_argument(
       '-l', '--left', action='store_true', help='Place
ROI on the left')
   parser.add_argument(
       '-s',
       '--shot',
       action='store_true',
       help='Not video, just a single shot')
   return parser.parse_args()


def main():
   """Main function of the app"""
   args = analyse_args()
   video_capture = cv2.VideoCapture(0)
   lower_color = np.array([0, 50, 120], dtype=np.uint8)
   upper_color = np.array([180, 150, 250], dtype=np.uint8)

   while True:
       _, frame = video_capture.read()
       frame = cv2.flip(frame, 1)

       cv2.putText(frame, 'Welcome', (0, 50),
cv2.FONT_HERSHEY_SIMPLEX, 2,
                   (255, 0, 0), 3, cv2.LINE_AA)

       cv2.imshow('VCOM Project', frame)
       key = cv2.waitKey(10)
       if key != -1:
           cv2.destroyAllWindows()
           video_capture.release()
           break

   if key == ord('v'):
       try:
           avg_color, max_sensibility =
cc.captureCamera(args.left)
           vd.start(
               avg_color,
               max_sensibility,
               video=not args.shot,
               path=args.input,
               left=args.left)
       except TypeError:
           print 'Did not calculate the color bound.'
   elif key == ord('h'):
       cd.draw_contours(lower_color, upper_color)


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

8.2 color_calculator.py

```
#!/usr/bin/env python

import numpy as np
import cv2


def captureCamera(left=False):
   """
  Creates a color bound based on a ROI
  It analyses the blue square and calculates the maximum,
minimum and average HSV values inside the square.
  Those maximum and minimum values will be used to
determine the maximum sensibility possible, and the
average will be the color bound used to detect the hand.
  Parameters
  ----------
  left : bool, optional
     Set the ROI on the left side of the screen
  """
   cap = cv2.VideoCapture(0)

   outerRectangleXIni = 300
   outerRectangleYIni = 50
   outerRectangleXFin = 550
   outerRectangleYFin = 300
   innerRectangleXIni = 400
   innerRectangleYIni = 150
   innerRectangleXFin = 450
   innerRectangleYFin = 200

   if left:
       move_to_left = 250
       outerRectangleXIni = outerRectangleXIni -
move_to_left
       outerRectangleXFin = outerRectangleXFin -
move_to_left
       innerRectangleXIni = innerRectangleXIni -
move_to_left
       innerRectangleXFin = innerRectangleXFin -
move_to_left

   while True:
       ret, frame = cap.read()
       frame = cv2.flip(frame, 1)
       font = cv2.FONT_HERSHEY_SIMPLEX
       cv2.rectangle(frame, (outerRectangleXIni,
outerRectangleYIni),
                     (outerRectangleXFin,
outerRectangleYFin), (0, 255, 0), 0)
       cv2.rectangle(frame, (innerRectangleXIni,
innerRectangleYIni),
                     (innerRectangleXFin,
innerRectangleYFin), (255, 0, 0), 0)
       cv2.putText(frame, 'Please center your hand in the
square', (0, 35),
                   font, 1, (255, 0, 0), 3, cv2.LINE_AA)
```

```
        cv2.imshow('Camera', frame)

        key = cv2.waitKey(1)
        if key == ord('q'):
            cap.release()
            return None
        elif key != -1:
            roi = frame[innerRectangleYIni +
                        1:innerRectangleYFin,
innerRectangleXIni +
                        1:innerRectangleXFin]
            display_result(roi)
            approved = wait_approval()
            if approved:
                break
            cv2.destroyAllWindows()

    cap.release()
    hsvRoi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

    lower = np.array(
        [hsvRoi[:, :, 0].min(), hsvRoi[:, :, 1].min(),
hsvRoi[:, :, 2].min()])
    upper = np.array(
        [hsvRoi[:, :, 0].max(), hsvRoi[:, :, 1].max(),
hsvRoi[:, :, 2].max()])
    h = hsvRoi[:, :, 0]
    s = hsvRoi[:, :, 1]
    v = hsvRoi[:, :, 2]
    hAverage = np.average(h)
    sAverage = np.average(s)
    vAverage = np.average(v)

    hMaxSensibility = max(abs(lower[0] - hAverage),
abs(upper[0] - hAverage))
    sMaxSensibility = max(abs(lower[1] - sAverage),
abs(upper[1] - sAverage))
    vMaxSensibility = max(abs(lower[2] - vAverage),
abs(upper[2] - vAverage))

    cv2.destroyAllWindows()
    return np.array([[hAverage, sAverage, vAverage],
[hMaxSensibility, sMaxSensibility, vMaxSensibility]])


def display_result(roi):
    """Draws images of the selected ROI"""
    hsvRoi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    roi_result = np.concatenate((roi, hsvRoi))
    cv2.imshow('ROI Result', roi_result)


def wait_approval():
    """Checks if User wants the selected ROI"""
    approval = False
    key = cv2.waitKey(0)
    if key != -1 and key != ord('n'):
        approval = True
    return approval


def main():
    """Main function of the app"""
    captureCamera()


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

8.3 video_detection.py

```
#!/usr/bin/env python

import numpy as np
import cv2
import math
import traceback


def nothing(x):
    pass


def apply_sensibility(avg_color, newHSens, newSSens,
newVSens, maxSensibility):
  """
  Applies sensibility values for each value of HSV, taking
into account the maximum sensibility possible.
  It analyses the parameters and executes the hand
detection accordingly.
  Parameters
  ----------
  avg_color : array
     The average of HSV values to be detected
  newHSens : int
     Percentage of sensibility to apply to Hue
  newSSens : int
     Percentage of sensibility to apply to Saturation
  newVSens : int
     Percentage of sensibility to apply to Value
  maxSensibility : array
     The maximum error margin of HSV values to be detected
  """
  hSens = (newHSens * maxSensibility[0]) / 100
  SSens = (newSSens * maxSensibility[1]) / 100
  VSens = (newVSens * maxSensibility[2]) / 100
  lower_bound_color = np.array([avg_color[0] - hSens,
avg_color[1] - SSens, avg_color[2] - VSens])
  upper_bound_color = np.array([avg_color[0] + hSens,
avg_color[1] + SSens, avg_color[2] + VSens])
  return np.array([lower_bound_color, upper_bound_color])


def start(avg_color,
          max_sensibility,
          video=True,
          path=None,
          left=False):
  """
  Initializes the detection process.
  It analyses the parameters and executes the hand
detection accordingly.
  Parameters
  ----------
  avg_color : array
     The average of HSV values to be detected
  max_sensibility : array
     The maximum error margin of HSV values to be detected
  video : bool, optional
     False if single image
     True if video stream
  path : str, optional
     Path for the image to be analysed
  left : bool, optional
     Set the ROI on the left side of the screen
  """
```

```
  # change this value to better adapt to environment light
(percentage values)
  hSensibility = 100
  sSensibility = 100
  vSensibility = 100

  apply_sensibility(avg_color, hSensibility, sSensibility,
vSensibility, max_sensibility)

  cv2.namedWindow('Hand Detection')
  cv2.createTrackbar('HSensb', 'Hand Detection',
hSensibility, 100, nothing)
  cv2.createTrackbar('SSensb', 'Hand Detection',
sSensibility, 100, nothing)
  cv2.createTrackbar('VSensb', 'Hand Detection',
vSensibility, 100, nothing)

  if path != None:
      frame = cv2.imread(path)
      hand_detection(frame, lower_bound_color,
upper_bound_color, left)
      cv2.waitKey(0)
  else:
      video_capture = cv2.VideoCapture(0)

      while True:
          try:
              _, frame = video_capture.read()
              frame = cv2.flip(frame, 1)
               # get values from trackbar
              newHSens = cv2.getTrackbarPos('HSensb',
'Hand Detection')
              newSSens = cv2.getTrackbarPos('SSensb',
'Hand Detection')
              newVSens = cv2.getTrackbarPos('VSensb',
'Hand Detection')

              # and apply the new sensibility values
              lower_bound_color, upper_bound_color =
apply_sensibility(avg_color, newHSens, newSSens, newVSens,
max_sensibility)
              hand_detection(frame, lower_bound_color,
upper_bound_color, left)

          except Exception as e:
              print e
              pass

          if not video:
              cv2.waitKey(0)
              cv2.destroyAllWindows()
              break

          key = cv2.waitKey(10)
          if key == ord('q'):
              video_capture.release()
              cv2.destroyAllWindows()
              break


def hand_detection(frame, lower_bound_color,
upper_bound_color, left):
    """
  Initializes the detection process.
  It analyses the parameters and executes the hand
detection accordingly.
  Parameters
  ----------
  frame : array-like
     The frame to be analysed
  lower_bound_color : array
     The min of HSV values to be detected
  upper_bound_color : array
     The max of HSV values to be detected
  left : bool, optional
     Set the ROI on the left side of the screen
  """
   kernel = np.ones((3, 3), np.uint8)

   if left:
       roi = frame[100:300, 100:300]
       cv2.rectangle(frame, (100, 100), (300, 300), (0,
255, 0), 0)
   else:
       roi = frame[50:300, 300:550]
       cv2.rectangle(frame, (300, 50), (550, 300), (0,
255, 0), 0)

   hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

   binary_mask = cv2.inRange(hsv, lower_bound_color,
upper_bound_color)
   mask = cv2.dilate(binary_mask, kernel, iterations=3)
   mask = cv2.erode(mask, kernel, iterations=3)
   mask = cv2.GaussianBlur(mask, (5, 5), 90)

   _, contours, hierarchy = cv2.findContours(mask,
cv2.RETR_TREE,

cv2.CHAIN_APPROX_SIMPLE)

   cnt = max(contours, key=lambda x: cv2.contourArea(x))

   l = analyse_defects(cnt, roi)
   analyse_contours(frame, cnt, l + 1)

   show_results(binary_mask, mask, frame)


def analyse_defects(cnt, roi):
   """
  Calculates how many convexity defects are on the image.
  A convexity defect is a area that is inside the
convexity hull but does not belong to the object.
  Those defects in our case represent the division between
fingers.
  Parameters
  ----------
  cnt : array-like
     Contour of max area on the image, in this case, the
contour of the hand
  roi : array-like
     Region of interest where should be drawn the found
convexity defects
  """
   epsilon = 0.0005 * cv2.arcLength(cnt, True)
   approx = cv2.approxPolyDP(cnt, epsilon, True)

   hull = cv2.convexHull(approx, returnPoints=False)
   defects = cv2.convexityDefects(approx, hull)

   l = 0
   for i in range(defects.shape[0]):
       s, e, f, d = defects[i, 0]
```

```
        start = tuple(approx[s][0])
        end = tuple(approx[e][0])
        far = tuple(approx[f][0])
        pt = (100, 180)

        a = math.sqrt((end[0] - start[0])**2 + (end[1] -
start[1])**2)
        b = math.sqrt((far[0] - start[0])**2 + (far[1] -
start[1])**2)
        c = math.sqrt((end[0] - far[0])**2 + (end[1] -
far[1])**2)
        s = (a + b + c) / 2
        ar = math.sqrt(s * (s - a) * (s - b) * (s - c))

        d = (2 * ar) / a

        angle = math.acos((b**2 + c**2 - a**2) / (2 * b *
c)) * 57

        if angle <= 90 and d > 30:
            l += 1
            cv2.circle(roi, far, 3, [255, 0, 0], -1)
        cv2.line(roi, start, end, [0, 255, 0], 2)
    return l


def analyse_contours(frame, cnt, l):
    """
  Writes to the image the signal of the hand.
  The hand signals can be the numbers from 0 to 5, the
'ok' signal, and the 'all right' symbol.
  The signals is first sorted by the number of convexity
defects. Then, if the number of convexity defects is 1, 2,
or 3, the area ratio is to be analysed.
  Parameters
  ----------
  frame : array-like
     The frame to be analysed
  cnt : array-like
     Contour of max area on the image, in this case, the
contour of the hand
  l : int
     Number of convexity defects
  """
    hull = cv2.convexHull(cnt)

    areahull = cv2.contourArea(hull)
    areacnt = cv2.contourArea(cnt)

    arearatio = ((areahull - areacnt) / areacnt) * 100

    font = cv2.FONT_HERSHEY_SIMPLEX
    if l == 1:
        if areacnt < 2000:
            cv2.putText(frame, 'Put hand in the box', (0,
50), font, 2,
                        (0, 0, 255), 3, cv2.LINE_AA)
        else:
            if arearatio < 12:
                cv2.putText(frame, '0', (0, 50), font, 2,
(0, 0, 255), 3,
                            cv2.LINE_AA)
            elif arearatio < 17.5:
                cv2.putText(frame, 'Fixe', (0, 50), font,
2, (0, 0, 255), 3,
                            cv2.LINE_AA)
            else:
                cv2.putText(frame, '1', (0, 50), font, 2,
(0, 0, 255), 3,
                            cv2.LINE_AA)
    elif l == 2:
        cv2.putText(frame, '2', (0, 50), font, 2, (0, 0,
255), 3, cv2.LINE_AA)
    elif l == 3:
        if arearatio < 27:
            cv2.putText(frame, '3', (0, 50), font, 2, (0,
0, 255), 3,
                        cv2.LINE_AA)
        else:
            cv2.putText(frame, 'ok', (0, 50), font, 2, (0,
0, 255), 3,
                        cv2.LINE_AA)
    elif l == 4:
        cv2.putText(frame, '4', (0, 50), font, 2, (0, 0,
255), 3, cv2.LINE_AA)
    elif l == 5:
        cv2.putText(frame, '5', (0, 50), font, 2, (0, 0,
255), 3, cv2.LINE_AA)
    elif l == 6:
        cv2.putText(frame, 'reposition', (0, 50), font, 2,
(0, 0, 255), 3,
                    cv2.LINE_AA)
    else:
        cv2.putText(frame, 'reposition', (10, 50), font, 2,
(0, 0, 255), 3,
                    cv2.LINE_AA)


def show_results(binary_mask, mask, frame):
    """
  Shows the image with the results on it.
  The image is a result of a combination of the image with
the result on it, the original captured ROI, and the ROI
after optimizations.
  Parameters
  ----------
  binary_mask : array-like
     ROI as it is captured
  mask : array-like
     ROI after optimizations
  frame : array-like
     Frame to be displayed
  """
    combine_masks = np.concatenate((binary_mask, mask),
axis=0)
    height, _, _ = frame.shape
    _, width = combine_masks.shape
    masks_result = cv2.resize(combine_masks, dsize=(width,
height))
    masks_result = cv2.cvtColor(masks_result,
cv2.COLOR_GRAY2BGR)
    result_image = np.concatenate((frame, masks_result),
axis=1)
    cv2.imshow('Hand Detection', result_image)


def main():
    """Main function of the app"""
    lower_color = np.array([0, 50, 120], dtype=np.uint8)
    upper_color = np.array([180, 150, 250], dtype=np.uint8)

    start(lower_color, upper_color)


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