Implementation and Report of

Finger Counting Software

Authors:
Md Sakibul Alam
Hossain Mohammad Farhad
Akash Saha
Mahadi Hasan
Md Ripon Bhuiya

Supervisor:
Supta Richard Philip

July 16, 2019

Contents

Abstract

Detecting or recognizing anything from human by computer systems has become a major field of interest. Interacting with computer with face, hand or even sign is now a very popular topic in the world. It is becoming one of the most researched field in computer science. We for practice and leaning purpose done a project that detects a human hand and say the number of fingers are being shown to the camera. It will show the number on the computer screen of the number of fingers. This project can be modified in future to have techniques like communication by hand such as sign language, giving any kind of command to computer by hand etc.

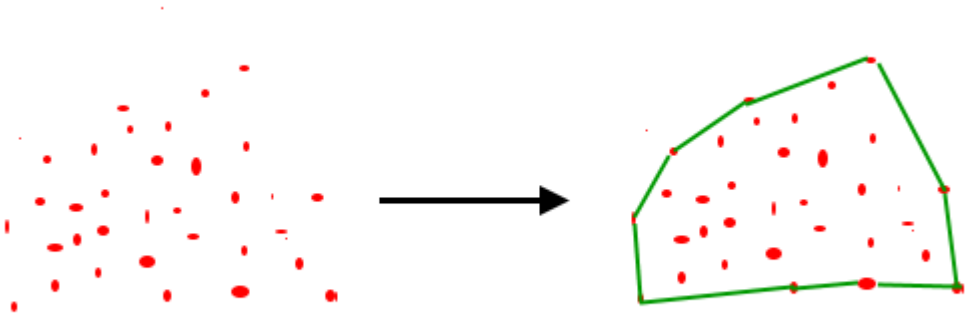
Our project is at beginning level that is the detecting is not yet so perfect. We may need to try more than once to get perfect result. Also the background needs to be fully white otherwise the software fails to detect the fingers.

Chapter 1

1 Introduction

1.1 Problem Statement

A Real time Finger counting software which identifies a person's hand on the spot and say the number of finger is held up. It uses convex hull algorithm for giving the result



1.2 Objectives

Our aim, which we believe we have nearly reached, was to develop a method of finger counting software that will able to say the number of finger is held up. But the software has some complications life the camera quality, background color, the way someone show the hand to the camera. Although the software will work fine if we ignore these problems.

Chapter 2

2 Background Study

2.1 The History of Object Detection

Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.
In the late 1960s, computer vision began at universities that were pioneering artificial intelligence. It was meant to mimic the human visual system, as a stepping stone to endowing robots with intelligent behavior.[11] In 1966, it was believed that this could be achieved through a summer project, by

attaching a camera to a computer and having it "describe what it saw".

What distinguished computer vision from the prevalent field of digital image processing at that time was a desire to extract three-dimensional structure from images with the goal of achieving full scene understanding. Studies in the 1970s formed the early foundations for many of the computer vision algorithms that exist today, including extraction of edges from images, labeling of lines, non-polyhedral and polyhedral modeling, representation of objects as interconnections of smaller structures, optical flow, and motion estimation.

The next decade saw studies based on more rigorous mathematical analysis and quantitative aspects of computer vision. These include the concept of scale-space, the inference of shape from various cues such as shading, texture and focus, and contour models known as snakes. Researchers also realized that many of these mathematical concepts could be treated within the same optimization framework as regularization and Markov random fields.[14] By the 1990s, some of the previous research topics became more active than the others. Research in projective 3-D reconstructions led to better understanding of camera calibration. With the advent of optimization methods for camera calibration, it was realized that a lot of the ideas were already explored in bundle adjustment theory from the field of photogrammetry. This led to methods for sparse 3-D reconstructions of scenes from multiple images. Progress was made on the dense stereo correspondence problem and further multi-view stereo techniques. At the same time, variations of graph cut were used to solve image segmentation. This decade also marked the first time statistical learning techniques were used in practice to recognize faces in images (see Eigenface). Toward the end of the 1990s, a significant change came about with the increased interaction between the fields of computer graphics and computer vision. This included image-based rendering, image morphing, view interpolation, panoramic image stitching and early light-field rendering.

Recent work has seen the resurgence of feature-based methods, used in conjunction with machine learning techniques and complex optimization frameworks.

2.2 Related Fields

Artificial intelligence Areas of artificial intelligence deal with autonomous planning or deliberation for robotical systems to navigate through an environment. A detailed understanding of these environments is required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.

Information engineering Computer vision is often considered to be part of information engineering.

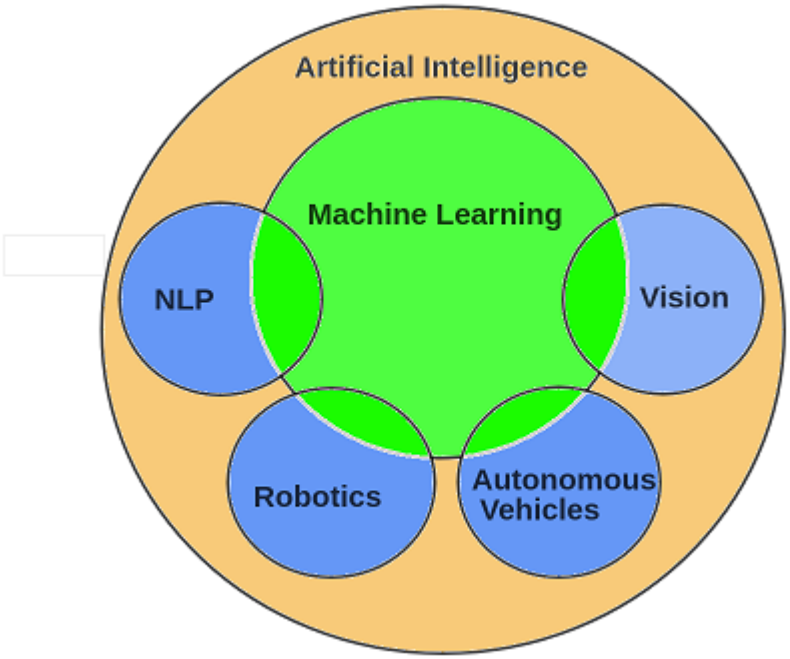
Solid-state physics Solid-state physics is another field that is closely related to computer vision. Most computer vision systems rely on image sensors, which detect electromagnetic radiation, which is typically in the form of either visible or infra-red light. The sensors are designed using quantum physics. The process by which light interacts with surfaces is explained using physics.

Neurobiology A third field which plays an important role is neurobiology, specifically the study of the biological vision system. Over the last century, there has been an extensive study of eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. This has led to a coarse, yet complicated, description of how "real" vision systems operate in order to solve certain vision-related tasks. These results have led to a subfield within computer vision where artificial systems are designed to mimic the processing and behavior of biological systems, at different levels of complexity. Also, some of the learning-based methods developed within computer vision (e.g. neural net and deep learning based image and feature analysis and classification) have their background in biology.

Signal processing Yet another field related to computer vision is signal processing. Many methods for processing of one-variable signals, typically temporal signals, can be extended in a natural way to processing of two-variable signals or multi-variable signals in computer vision. However, because of the specific nature of images there are many methods developed within computer vision which have no counterpart in processing of one-

variable signals. Together with the multi-dimensionality of the signal, this defines a subfield in signal processing as a part of computer vision.

Other fields Beside the above-mentioned views on computer vision, many of the related research topics can also be studied from a purely mathematical point of view. For example, many methods in computer vision are based on statistics, optimization or geometry. Finally, a significant part of the field is devoted to the implementation aspect of computer vision; how existing methods can be realized in various combinations of software and hardware, or how these methods can be modified in order to gain processing speed without losing too much performance. Computer vision is also used in fashion ecommerce, inventory management, patent search, furniture, and the beauty industry.



Chapter 3

3 Methodology

3.1 Requirement

Hands-on knowledge of Numpy and Matplotlib is essential before working on the concepts of OpenCV. Make sure that you have the following packages installed and running before installing OpenCV.

- Python
- Numpy
- Matplotlib

3.2 Algorithm

Convex hull (CH) is basically an important geometrical problem that could be solved computationally. The problem is all about constructing, developing, articulating, circumscribing or encompassing a given set of points in plane by a polygonal capsule called convex polygon. Justifiably, convex hull problem is combinatorial in general and an optimization problem in particular. Here, convexity refers to the property of the polygon that surrounds the given points making a capsule. A convex polygon is a simple polygon without any self-intersection in which any line segment between two points on the edges ever goes outside the polygon. Lucidly stated, a convex polygon houses a convex set which is a region such that every pair of points is within the region and the line that joins such pair is also within the region. With this definition, a cube, rectangle, regular polygon and the like are convex in nature. Any polygon which has hollowness, dent or extended vertices cannot be convex. A convex curve forms the boundary of a convex set.

3.2.1 Finger Counting Software Code With Explanation

Imports

In [1]:
```
import cv2
import numpy as np

# Used for distance calculation later on
from sklearn.metrics import pairwise
```

Global Variables

We will use these as we go along.

In [2]:
```
# This background will be a global variable that we update through a few functions
background = None

# Start with a halfway point between 0 and 1 of accumulated weight
accumulated_weight = 0.5


# Manually set up our ROI for grabbing the hand.
# Feel free to change these. I just chose the top right corner for filming.
roi_top = 20
roi_bottom = 300
roi_right = 300
roi_left = 600
```

Finding Average Background Value

The function calculates the weighted sum of the input image src and the accumulator dst so that dst becomes a running average of a frame sequence:

In [3]:
```
def calc_accum_avg(frame, accumulated_weight):
    '''
    Given a frame and a previous accumulated weight, computed the weighted average of the image passed in.
    '''

    # Grab the background
    global background

    # For first time, create the background from a copy of the frame.
    if background is None:
        background = frame.copy().astype("float")
        return None

    # compute weighted average, accumulate it and update the background
    cv2.accumulateWeighted(frame, background, accumulated_weight)
```

Segment the Hand Region in Frame

In [4]:
```
def segment(frame, threshold=25):
    global background

    # Calculates the Absolute Differentce between the backgroud and the passed in frame
    diff = cv2.absdiff(background.astype("uint8"), frame)

    # Apply a threshold to the image so we can grab the foreground
    # We only need the threshold, so we will throw away the first item in the tuple with an underscore _
    _ , thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Grab the external contours form the image
    # Again, only grabbing what we need here and throwing away the rest
    image, contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # If length of contours list is 0, then we didn't grab any contours!
    if len(contours) == 0:
        return None
    else:
        # Given the way we are using the program, the largest external contour should be the hand (largest by area)
        # This will be our segment
        hand_segment = max(contours, key=cv2.contourArea)

        # Return both the hand segment and the thresholded hand image
        return (thresholded, hand_segment)
```

Counting Fingers with a Convex Hull

We just calculated the external contour of the hand. Now using that segmented hand, let's see how to calculate fingers. Then we can count how many are up!

Example of ConvexHulls:

In [5]:
```
def count_fingers(thresholded, hand_segment):


    # Calculated the convex hull of the hand segment
    conv_hull = cv2.convexHull(hand_segment)

    # Now the convex hull will have at least 4 most outward points, on the top, bottom, left, and right.
    # Let's grab those points by using argmin and argmax. Keep in mind, this would require reading the documentation
    # And understanding the general array shape returned by the conv hull.

    # Find the top, bottom, left , and right.
    # Then make sure they are in tuple format
    top    = tuple(conv_hull[conv_hull[:, :, 1].argmin()][0])
    bottom = tuple(conv_hull[conv_hull[:, :, 1].argmax()][0])
    left   = tuple(conv_hull[conv_hull[:, :, 0].argmin()][0])
    right  = tuple(conv_hull[conv_hull[:, :, 0].argmax()][0])

    # In theory, the center of the hand is half way between the top and bottom and halfway between left and right
    cX = (left[0] + right[0]) // 2
    cY = (top[1] + bottom[1]) // 2
```

```
# find the maximum euclidean distance between the center of the palm
# and the most extreme points of the convex hull

# Calculate the Euclidean Distance between the center of the hand and the left, right, top, and bottom.
distance = pairwise.euclidean_distances([(cX, cY)], Y=[left, right, top, bottom])[0]

# Grab the largest distance
max_distance = distance.max()

# Create a circle with 90% radius of the max euclidean distance
radius = int(0.8 * max_distance)
circumference = (2 * np.pi * radius)

# Not grab an ROI of only that circle
circular_roi = np.zeros(thresholded.shape[:2], dtype="uint8")

# draw the circular ROI
cv2.circle(circular_roi, (cX, cY), radius, 255, 10)


# Using bit-wise AND with the cirle ROI as a mask.
# This then returns the cut out obtained using the mask on the thresholded hand image.
circular_roi = cv2.bitwise_and(thresholded, thresholded, mask=circular_roi)

# Grab contours in circle ROI
image, contours, hierarchy = cv2.findContours(circular_roi.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

# Finger count starts at 0
count = 0
```

NG

```
# loop through the contours to see if we count any more fingers.
for cnt in contours:

    # Bounding box of countour
    (x, y, w, h) = cv2.boundingRect(cnt)

    # Increment count of fingers based on two conditions:

    # 1. Contour region is not the very bottom of hand area (the wrist)
    out_of_wrist = ((cY + (cY * 0.25)) > (y + h))

    # 2. Number of points along the contour does not exceed 25% of the circumference of the circular ROI (otherwise we'r
    limit_points = ((circumference * 0.25) > cnt.shape[0])


    if  out_of_wrist and limit_points:
        count += 1

return count
```

NG

Run Program

In [10]:
```
cam = cv2.VideoCapture(0)

# Intialize a frame count
num_frames = 0

# keep looping, until interrupted
while True:
    # get the current frame
    ret, frame = cam.read()

    # flip the frame so that it is not the mirror view
    frame = cv2.flip(frame, 1)

    # clone the frame
    frame_copy = frame.copy()

    # Grab the ROI from the frame
    roi = frame[roi_top:roi_bottom, roi_right:roi_left]

    # Apply grayscale and blur to ROI
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (7, 7), 0)

    # For the first 30 frames we will calculate the average of the background.
    # We will tell the user while this is happening
    if num_frames < 60:
        calc_accum_avg(gray, accumulated_weight)
        if num_frames <= 59:
            cv2.putText(frame_copy, "WAIT! GETTING BACKGROUND AVG.", (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
            cv2.imshow("Finger Count",frame_copy)

    else:
```

```
    else:
        # now that we have the background, we can segment the hand.

        # segment the hand region
        hand = segment(gray)

        # First check if we were able to actually detect a hand
        if hand is not None:

            # unpack
            thresholded, hand_segment = hand

            # Draw contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (roi_right, roi_top)], -1, (255, 0, 0),1)

            # Count the fingers
            fingers = count_fingers(thresholded, hand_segment)

            # Display count
            cv2.putText(frame_copy, str(fingers), (70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

            # Also display the thresholded image
            cv2.imshow("Thesholded", thresholded)
```

```
    # Draw ROI Rectangle on frame copy
    cv2.rectangle(frame_copy, (roi_left, roi_top), (roi_right, roi_bottom), (0,0,255), 5)

    # increment the number of frames for tracking
    num_frames += 1

    # Display the frame with segmented hand
    cv2.imshow("Finger Count", frame_copy)


    # Close windows with Esc
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

# Release the camera and destroy all the windows
cam.release()
cv2.destroyAllWindows()
```

Chapter 4

4 Conclusion

We have a real time finger counting project.We are using the technologies available in the Open-Computer Vision (OpenCV) library and methodology to implement them using Python.

Our project can only detect the finger. Not so perfect yet So we will try to do that in future like that

5 Bibliography

- Hsu, R. L., Abdel-Mottaleb, M., Jain, A. K. (2002). Face detection in color images. IEEE transactions on pattern analysis and machine intelligence, 24(5), 696-706.

- Chazelle, Bernard (1993), "An optimal convex hull algorithm in any fixed dimension" (PDF), Discrete and Computational Geometry, 10 (1): 377–409, doi:10.1007/BF02573985.

- de Berg, M.; van Kreveld, M.; Overmars, Mark; Schwarzkopf, O. (2000), Computational Geometry: Algorithms and Applications, Springer, pp. 2–8.

- Andrew, A. M. (1979), "Another efficient algorithm for convex hulls in two dimensions", Information Processing Letters, 9 (5): 216–219, doi:10.1016/0020-0190(79)90072-3.

- van der Walt, Stéfan; Colbert, S. Chris; Varoquaux, Gaël (2011). "The NumPy array: a structure for efficient numerical computation". Computing in Science and Engineering. IEEE. arXiv:1102.1523. Bibcode:2011arXiv1102.1523V.

- Bradski, Gary; Kaehler, Adrian (2008). Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc. p. 6.

- Adrian Kaehler; Gary Bradski (14 December 2016). Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media. pp. 26ff. ISBN 978-1-4919-3800-3.

- Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor (1 April 2012). "Realtime Computer Vision with OpenCV". Queue: 40:40–40:56. doi:10.1145/2181796.2206309 (inactive 2019-07-14).