



IMAGE PROCESSING AND UNDERSTANDING COURSE, SEM 2021 -1

HAND GESTURE RECOGNITION (FINGER COUNTER)
(MID TERMS PROJECT)

Submitted by:

MULYA FAJAR NINGSIH ALWI - 001202000101

ANGELINA YULISA WANNEY – 001202000076

Submitted to:

Mr. Risnandar, Ph.D.

Date:

27th October 2021

TABLE OF CONTENTS

TABLE OF CONTENTS	I
CHAPTER 1 - INTRODUCTION.....	1
CHAPTER 2 - METHOD	2
2.1 IMAGE ACQUISITION	2
2.2 IMAGE SEGMENTATION.....	2
2.3 IMAGE COMPRESSION	3
2.4 FINGER RECOGNITION.....	3
2.5 PYTHON LIBRARIES	4
2.5.1 OPENCV	4
2.5.2 MEDIAPIPE	4
2.6 PROJECT CREATION STEPS	4
CHAPTER 3 - EXPERIMENTAL RESULT	8
CHAPTER 4 - CONCLUSION.....	13
REFERENCES.....	14

CHAPTER 1

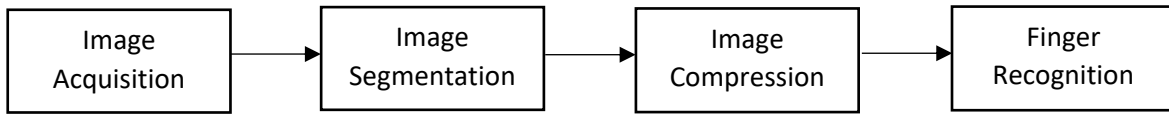
INTRODUCTION

This project or program was conducted with the aim of helping toddlers or young children to be able to recognize numbers, number gestures on fingers, and how to calculate simple calculations using fingers in an exciting way with the help of visualization. This program was developed using the python programming language with the help of additional libraries such as OpenCV and MediaPipe and several other methods.

To use this program, the user must be in front of the computer's webcam. The webcam will be used to recognize the shape and pattern of the user's hand and the hand signer will count the number of fingers recorded on the webcam. The program will display the results of hand patterns and the number of fingers that are recognized on a real-time video frame stream.

CHAPTER 2

METHOD



2.1 IMAGE ACQUISITION

Image acquisition in image processing can be broadly defined as the action of retrieving an image from some sources, usually a hardware-based source, so it can be passed through whatever processes need to occur afterward. Performing image acquisition in image processing is always the first step in the workflow sequence because, without an image, no processing is possible. The image that is acquired is completely unprocessed and is the result of whatever hardware was used to generate it, which can be very important in some fields to have a consistent baseline from which to work. One of the ultimate goals of this process is to have a source of input that operates within such controlled and measured guidelines that the same image can.

In OpenCV, we can use `VideoCapture(0)` to run the camera for retrieving image

For example: `capture = cv.VideoCapture(0)`

2.2 IMAGE SEGMENTATION

In digital image processing, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

In OpenCV it's by default reads images in BGR format. We can use the `cvtColor(src, flag)`

For example: `imageRGB = cv.cvtColor(image,cv.COLOR_BGR2RGB)`

2.3 IMAGE COMPRESSION

Image compression is the technique where the size of an image or the size in bytes of graphics is minimized by keeping the quality of the image to an acceptable level or without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space.

In OpenCV we can use the `resize(src,(fx/width, fy/height))`

For example: `image = cv.resize(image,(750, 550))`

2.4 FINGER RECOGNITION

In this method, we use MediaPipe libraries to help us detect the finger. MediaPipe Hands is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame. We can access two sub-modules from MediaPipe, namely drawing_utils and hands. The drawing_utils module includes some useful helper functions to draw detections and landmarks over images, amongst other functionalities. The hands module contains the Hands class that we will use to perform the detection of hand landmarks on an image. We are doing this as a convenience, to avoid using the full path every time we want to access one of the functionalities of these modules.

Below is an example of how we can access it:

```
mpDraw = mp.solutions.drawing_utils
```

```
mpHands = mp.solutions.hands
```

2.5 PYTHON LIBRARIES

2.5.1 OPENCV

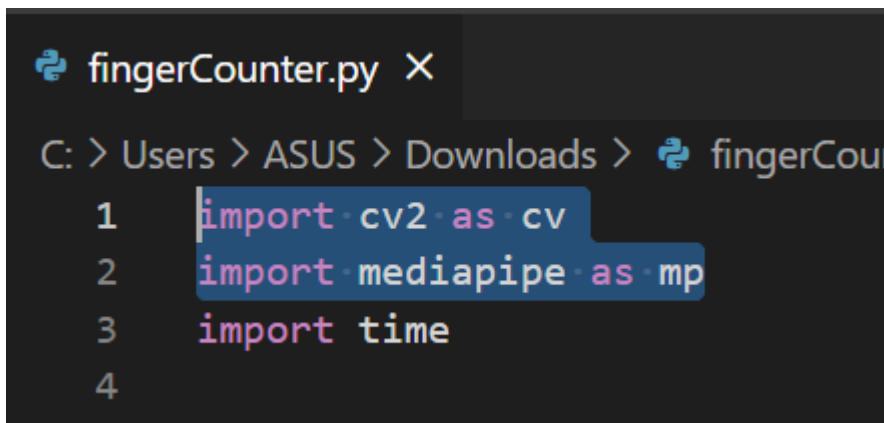
OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

2.5.2 MEDIAPIPE

MediaPipe is used for building multimodal (eg. video, audio, any time series data), cross-platform (i.e Android, iOS, web, edge devices) applied ML (Machine Learning) pipelines. With MediaPipe, a perception pipeline can be built as a graph of modular components, including, for instance, inference models (e.g., TensorFlow, TFLite) and media processing functions. MediaPipe Models are Face Detection, Face Mesh, Iris, Pose, Hands, Hair Segmentation, Object Detection, Selfie Segmentation, etc.

2.6 PROJECT CREATION STEPS

Step 1: Import Libraries



```
fingerCounter.py X
C: > Users > ASUS > Downloads > fingerCou
1 import cv2 as cv
2 import mediapipe as mp
3 import time
4
```

Step 2: Turn on the Webcam Using OpenCV

```
def main():  
    capture = cv.VideoCapture(0)  
  
    # Using web cam
```

Step 3: Define the Parameters

```
def __init__(self, image_mode=False, max_hands=2, model_complexity=1,  
             min_detection_confidence=0.8, min_tracking_confidence=0.5):  
    self.image_mode = image_mode  
    self.max_hands = max_hands  
    self.model_complexity = model_complexity  
    self.min_detection_confidence = min_detection_confidence  
    self.min_tracking_confidence = min_tracking_confidence  
  
    self.mpHands = mp.solutions.hands  
    self.hands = self.mpHands.Hands(self.image_mode, self.max_hands, self.model_complexity,  
                                     self.min_detection_confidence, self.min_tracking_confidence)  
    self.mpDraw = mp.solutions.drawing_utils  
    self.mp_drawing_styles = mp.solutions.drawing_styles
```

Step 4: Do the Image Segmentation Using Color Spaces

```
def detect_landmarks(self, image, draw=True, draw_conn  
                    imageRGB = cv.cvtColor(image, cv.COLOR_BGR2RGB)  
    land_mark_data = []  
    hand_classified_landmarks = [[], []]  
    results = self.hands.process(imageRGB)  
    landmarks = results.multi_hand_landmarks  
    data = None
```

Step 5: Define the Image Size and Resolution

```
while True:
    ret, image = capture.read()
    image = cv.flip(image, 1)
    image = cv.resize(image, (750, 550))
    print(image.shape)

    landmarks, image = hand_detector.detect_landmarks(
        image, draw_default_style=False)
```

Step 6: Detect the Hand Landmarks Using MediaPipe Hands

```
def detect_landmarks(self, image, draw=True, draw_connections=True, draw_default_style=False):
    imageRGB = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    land_mark_data = []
    hand_classified_landmarks = [[], []]
    results = self.hands.process(imageRGB)
    landmarks = results.multi_hand_landmarks
    data = None
    if landmarks:
        for hand_landmarks in landmarks:
            for id, landmark in enumerate(hand_landmarks.landmark):
                h, w, c = image.shape
                px, py = int(landmark.x*w), int(landmark.y*h)
                data = (id, px, py)
                land_mark_data.append(data)
            if draw and not draw_connections:
                self.mpDraw.draw_landmarks(image, hand_landmarks)
            elif draw and draw_connections and not draw_default_style:
                self.mpDraw.draw_landmarks(
                    image, hand_landmarks, self.mpHands.HAND_CONNECTIONS)
            elif draw and draw_connections and draw_default_style:
                self.mpDraw.draw_landmarks(image, hand_landmarks, self.mpHands.HAND_CONNECTIONS, self.mp_drawing_styles.get_default_han
                    ), self.mp_drawing_styles.get_default_hand_connections_style())
        if land_mark_data[0][1] > land_mark_data[4][1]:
            if len(land_mark_data) > 20:
                hand_classified_landmarks[1] = land_mark_data[0:21]
                hand_classified_landmarks[0] = land_mark_data[21::]
            else:
                hand_classified_landmarks[1] = land_mark_data[0:21]
        elif land_mark_data[4][1] > land_mark_data[0][1]:
            if len(land_mark_data) > 20:
```


Step 7: Do the Finger Counter

```
def count_up_fingers(self, data):
    fingers = [[], []]
    if len(data[1]) != 0:
        if (data[1][3][1] > data[1][4][1]):
            fingers[1].append(1)
        else:
            fingers[1].append(0)

        if (data[1][5][2] > data[1][8][2] and data[1][7][2] > data[1][8][2]):
            fingers[1].append(1)
        else:
            fingers[1].append(0)
        if (data[1][9][2] > data[1][12][2] and data[1][11][2] > data[1][12][2]):
            fingers[1].append(1)
        else:
            fingers[1].append(0)
        if (data[1][13][2] > data[1][16][2] and data[1][15][2] > data[1][16][2]):
            fingers[1].append(1)
        else:
            fingers[1].append(0)
        if (data[1][17][2] > data[1][20][2] and data[1][19][2] > data[1][20][2]):
            fingers[1].append(1)
        else:
            fingers[1].append(0)
    if len(data[0]) != 0:
        if (data[0][3][1] < data[0][4][1]):
            fingers[0].append(1)
        else:
            fingers[0].append(0)
```

CHAPTER 3

EXPERIMENTAL RESULT



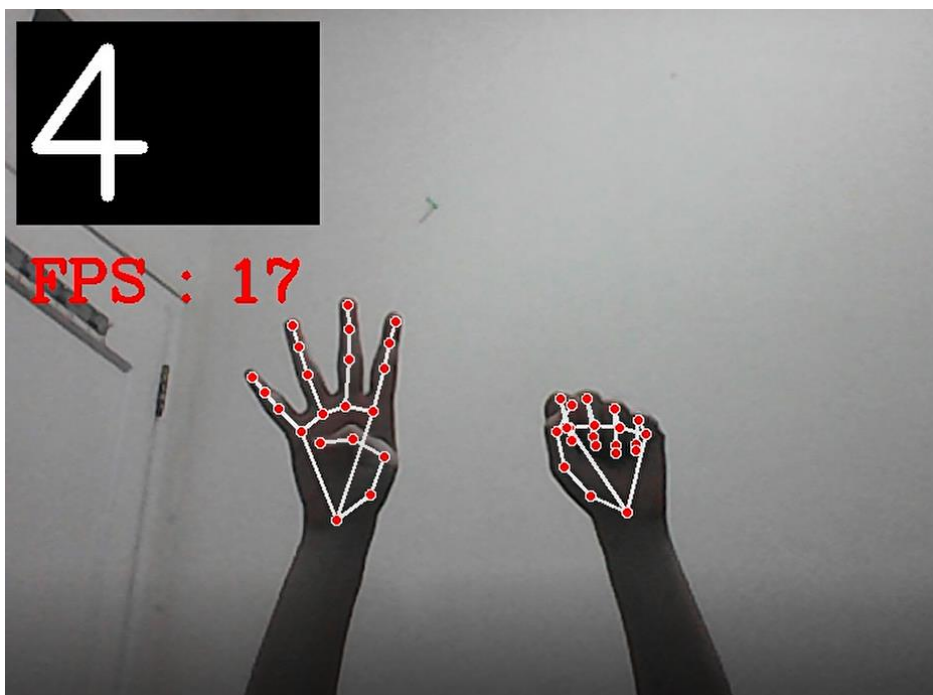
The recognition of the hand gesture for 1 finger.



The recognition of the hand gesture for 2 fingers.



The recognition of the hand gesture for 3 fingers.



The recognition of the hand gesture for 4 fingers.



The recognition of the hand gesture for 5 fingers.



The recognition of the hand gesture for 6 fingers.



The recognition of the hand gesture for 7 fingers.



The recognition of the hand gesture for 8 fingers.



The recognition of the hand gesture for 9 fingers.



The recognition of the hand gesture for 10 fingers.

CHAPTER 4

CONCLUSION

In this project or program, we have implemented various techniques for efficient human-computer interaction. We have also used different techniques or methods for processing the input and output images in our program. We hope that our program can be useful in accordance with our program's goal, which is to be able to help toddlers or young children to be able to recognize numbers, number gestures on fingers, and how to calculate simple calculations using fingers in an exciting way with the help of visualization.

REFERENCES

- Kashyapa, R. (2020, July 29). *Image Acquisition Components*. Retrieved from qualitastech.com: <https://qualitastech.com/image-acquisition-components/>
- MediaPipe. (n.d.). *MediaPipe Hands*. Retrieved from MediaPipe: <https://google.github.io/mediapipe/solutions/hands.html>
- Stone, R. (n.d.). *Image Segmentation Using Color Spaces in OpenCV + Python*. Retrieved from realpython.com: <https://realpython.com/python-opencv-color-spaces/>
- Surabhi N, S. N. (2017). *Image Compression Techniques: A Review*. Retrieved from ijedr.org: <https://www.ijedr.org/papers/IJEDR1701089.pdf>
- Murtaza's Workshop - Robotics and AI. (2021, April 3). *Finger Counter using Hand Tracking Computer Vision | OpenCV Python 2021*. Retrieved from YouTube: https://www.youtube.com/watch?v=p5Z_GGRCI5s&ab_channel=Murtaza%27sWorkshop-RoboticsandAI