MAJOR PROJECT REPORT

"HAND GESTURE RECOGNITION"

University School of Information, Communication and Technology

Guru Gobind Singh Indraprastha University, Delhi

Submitted by: YASH ARYAN (06816403220)

Batch: B.Tech (CSE) 8th Semester

Mentor: Dr. Priyanka Bhutani

TABLE OF CONTENT

Certificate

This is to certify that Yash Aryan (Enrollment No.06816403220) has successfully submitted the Major Project report entitled "Hand Gesture Recognition" in partial fulfillment for the requirement of the award of the degree of Bachelor of Technology in Computer Science Engineering at USICT. The project has been carried out under my supervision and guidance. He has shown dedication, perseverance, and a willingness to learn new technologies throughout the project.

Dr. Priyanka Bhutani

University School of Information, Communication & Technology, USICT_GGSIPU, New Delhi-110078

Date: 13-05-2024

Candidate’s Declaration

I, Yash Aryan (Enrollment No.06816403220), a student of B.Tech (CSE 8th Semester), USICT, Guru Gobind Singh Indraprastha University, hereby declare that the work which is presented in this Major Project Report entitled "Hand Gesture Recognition" is an original and authentic work of mine under the technical guidance of Dr. Priyanka Bhutani, Assistant Professor, USIC&T. I declare that the work in this project has not been submitted in full or in any part for any diploma or degree course of this or any other University to the best of my knowledge and belief. I will be solely responsible myself for any copyright infringement or plagiarism, if any, in the said work, and declare that all necessary due acknowledgement has been made in the content of said work. My supervisor/ guide shall not be held responsible for full or partial violation of copyright or intellectual property rights or any type of plagiarism involved above in the said work.

Name: Yash Aryan

Enrolment Number: 06816403220

Course: B. Tech CSE 8th Semester, University School of Information, Communication & Technology, USICT_GGSIPU, New Delhi-110078

Date: 13-05-2024

Acknowledgement

It gives me immense pleasure to take this opportunity to acknowledge my obligation to my mentor, Dr. Priyanka Bhutani, University School of Information and Communication Technology, GGSIPU, who has not only guided me throughout the project but also made a great effort in making the project a success. I am highly thankful to my guide for her keen interest, valuable guidance, technical acumen, round the clock encouragement, moral support & suggestions in the completion of the project.

Name: Yash Aryan

Enrolment Number: 06816403220

Course: B. Tech CSE 8th Semester, University School of Information, Communication & Technology, USICT_GGSIPU, New Delhi-110078

Date: 13-05-2024

Abstract

This project uses Mediapipe, OpenCV and Tensorflow for recognizing the hand gestures. This project was made using Python. When a user makes a gesture, it detects the hand gesture, recognizes it and displays the frames per second along with the detected gesture to the user.

The objectives of making a hand gesture recognition project using Mediapipe and OpenCV are:

1. Real-time Gesture Recognition: Developing a system capable of accurately recognizing hand gestures in real-time from camera feed.
2. Gesture Classification: Building a model that can classify different hand gestures into predefined categories or commands.
3. Human-Computer Interaction: Enabling natural and intuitive interactions between humans and computers or devices through hand gestures.
4. Accessibility: Creating interfaces that allow users with disabilities or limitations to interact with technology more easily through gestures, without relying solely on traditional input methods like keyboards or mice.

Problem Statement

Hand gesture recognition is a vital component in human-computer interaction, offering a natural and intuitive means of communication. In various domains such as sign language translation, virtual reality, robotics, and gaming, accurate recognition of hand gestures can significantly enhance user experience and accessibility. However, building an efficient hand gesture recognition system poses several challenges:

1. Complexity of Hand Gestures: Hand gestures can vary widely in terms of complexity, shape, and movement patterns, making their recognition a non-trivial task. Capturing the subtle nuances of hand movements and accurately translating them into meaningful commands require sophisticated algorithms.

2. Variability in Lighting and Background: Lighting conditions and background clutter can significantly affect the performance of hand gesture recognition systems. Variations in illumination and diverse backgrounds can obscure hand features, leading to errors in gesture classification.

3. Real-Time Processing: Many applications of hand gesture recognition, such as virtual reality gaming or human-robot interaction, demand real-time processing capabilities. Achieving low-latency recognition while maintaining high accuracy is essential for seamless user interaction.

4. Data Acquisition and Annotation: Acquiring a diverse dataset of hand gestures encompassing different hand shapes, orientations, and movements is crucial for training robust machine learning models. Additionally, annotating these datasets with accurate labels requires considerable effort and expertise.

5. Model Generalization: Ensuring that the trained gesture recognition model generalizes well to unseen data and can accurately classify gestures performed by different individuals is vital for its practical usability across various user demographics.

This project can accurately classify a wide range of hand gestures in real-time, under varying environmental conditions.

Functional Requirements

1. Gesture Detection and Classification:

- The system should accurately detect and classify a predefined set of hand gestures.

2. Robustness to Environmental Conditions:

- The system should be robust to changes in lighting conditions, background clutter, and variations in hand appearance.

3. Gesture Customization and Training:

- It should allow for customization of gestures, enabling users to define and train the system for new gestures.

4. Cross-platform Compatibility:

- It should support various operating systems (e.g., Windows, macOS, Linux, Android, iOS).

5. Training and Model Updates:

- It should support the training of machine learning models with new data to improve recognition accuracy over time.

Non-functional Requirements

1. Accuracy and Precision:

- The system should achieve high accuracy and precision in gesture recognition, minimizing false positives and false negatives.
- It should accurately distinguish between similar gestures and provide reliable recognition results.

2. Response Time:

- The system should have low response times, ensuring quick feedback to users after performing a gesture.

3. Portability:

- It should be compatible with a wide range of hardware configurations and operating systems.

4. Maintainability:

- The system should be designed with maintainability in mind, allowing for easy updates, bug fixes, and enhancements.
- Codebase should be well-structured, documented, and modular to facilitate future development and maintenance efforts.

Design and Implementation

- **Tech Stack Used**
  - Mediapipe
  - Tensorflow
  - OpenCV

- **Hardware and Software Interfaces**
  - **Hardware**
    - Fast internet enabled mobile or computer device
  - **Software**

Software Used	**Description**
Mediapipe	An open source, cross-platform, customizable ML solution for live and streaming media.
Tensorflow	TensorFlow is an open source software library for high performance numerical computation.
OpenCV	OpenCV is a cross-platform library using which we can develop real-time computer vision applications.

Components

1. Data Acquisition:

- This component involves capturing input data, typically in the form of images or video frames containing hand gestures.

2. Preprocessing:

- Preprocessing steps include resizing, cropping, and normalizing input images to ensure consistency and enhance model performance.

3. Hand Detection and Tracking:

- MediaPipe offers a hand detection solution that can locate and track hand landmarks in real-time. MediaPipe's hand tracking module detects the presence of hands in the input frames and track their movements.

4. Hand Landmark Detection:

- MediaPipe provides a pre-trained hand landmark model that can accurately identify key landmarks.

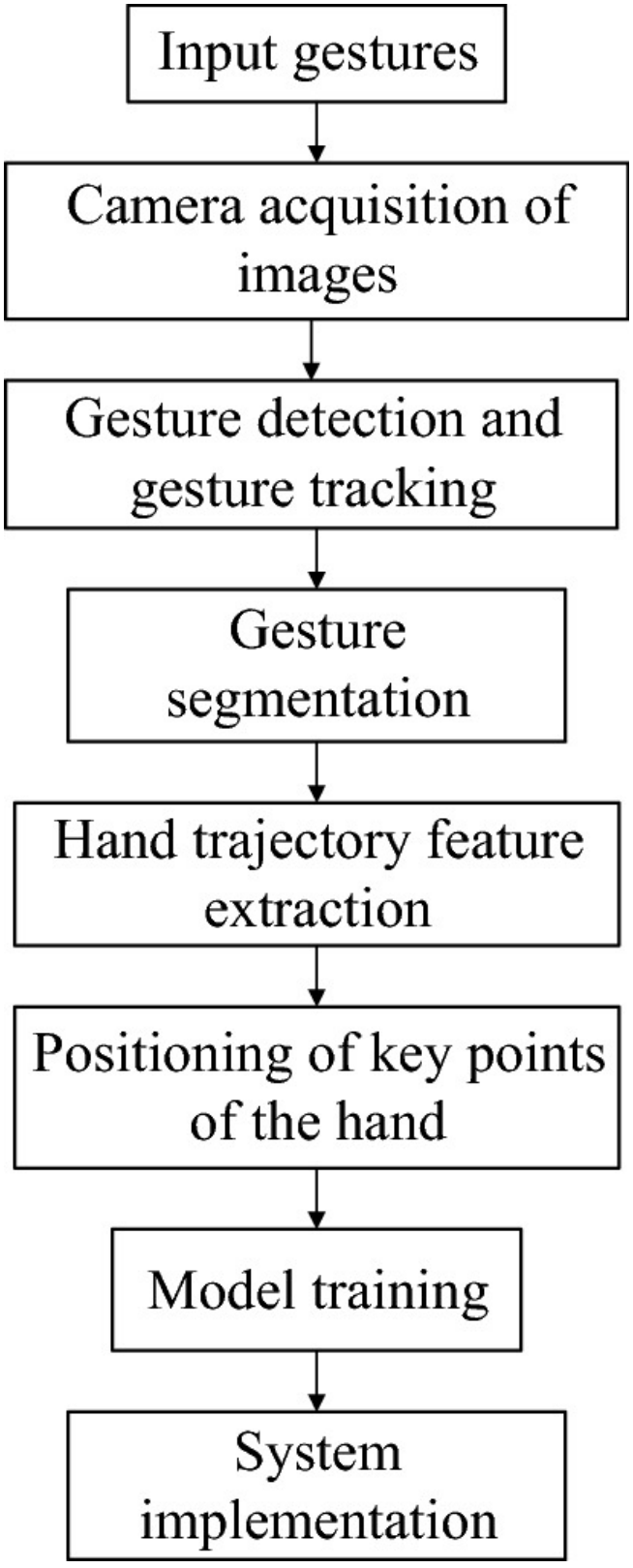
5. Feature Extraction:

- Extracting relevant features from the detected hand landmarks. This may involve computing distances between landmark points, angles between fingers, or other geometric properties that represent the hand gesture.

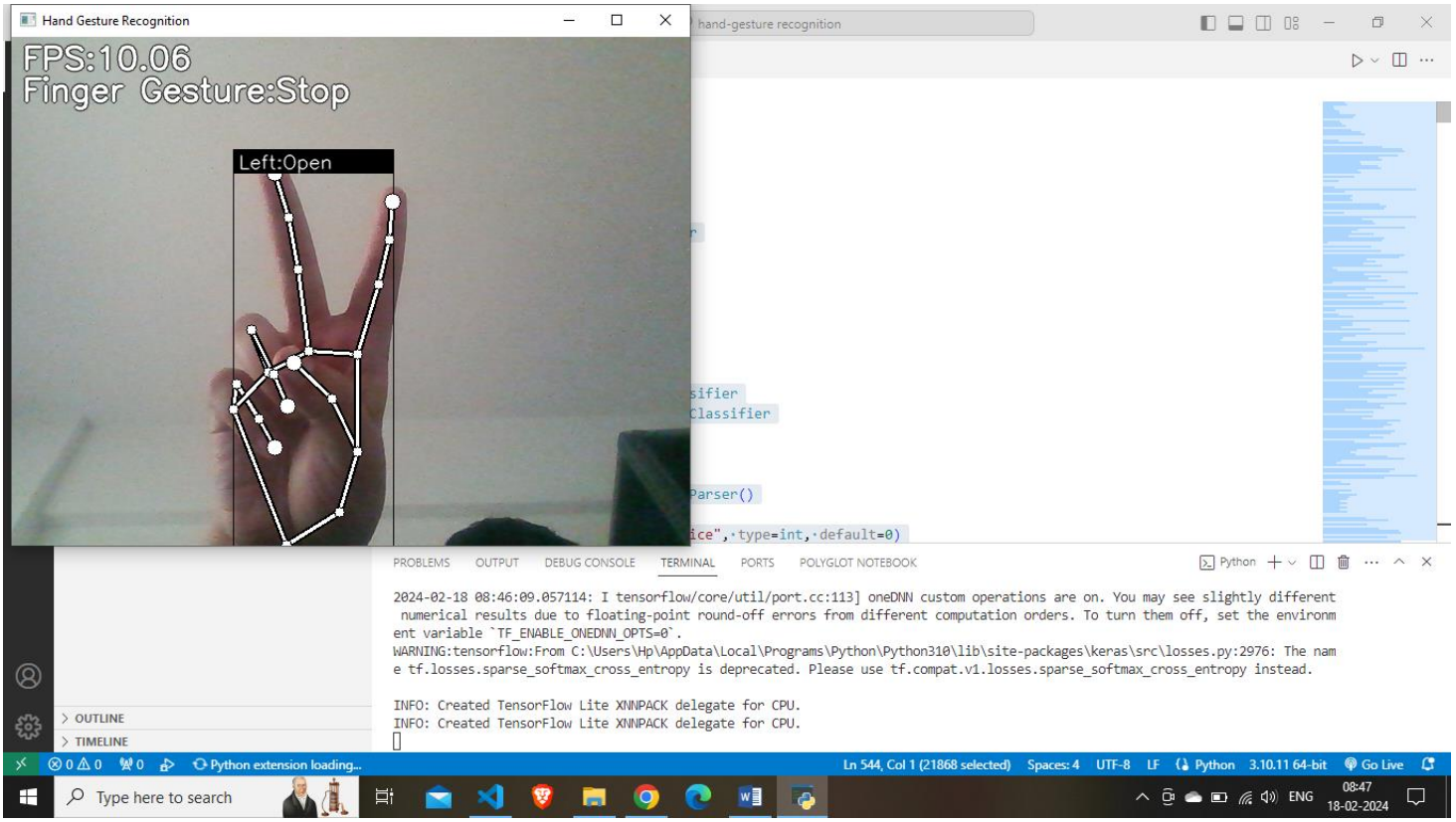
6. Gesture Classification:

- Using TensorFlow to build and train a deep learning model for gesture classification. This model takes the extracted features as input and predicts the corresponding gesture labels.

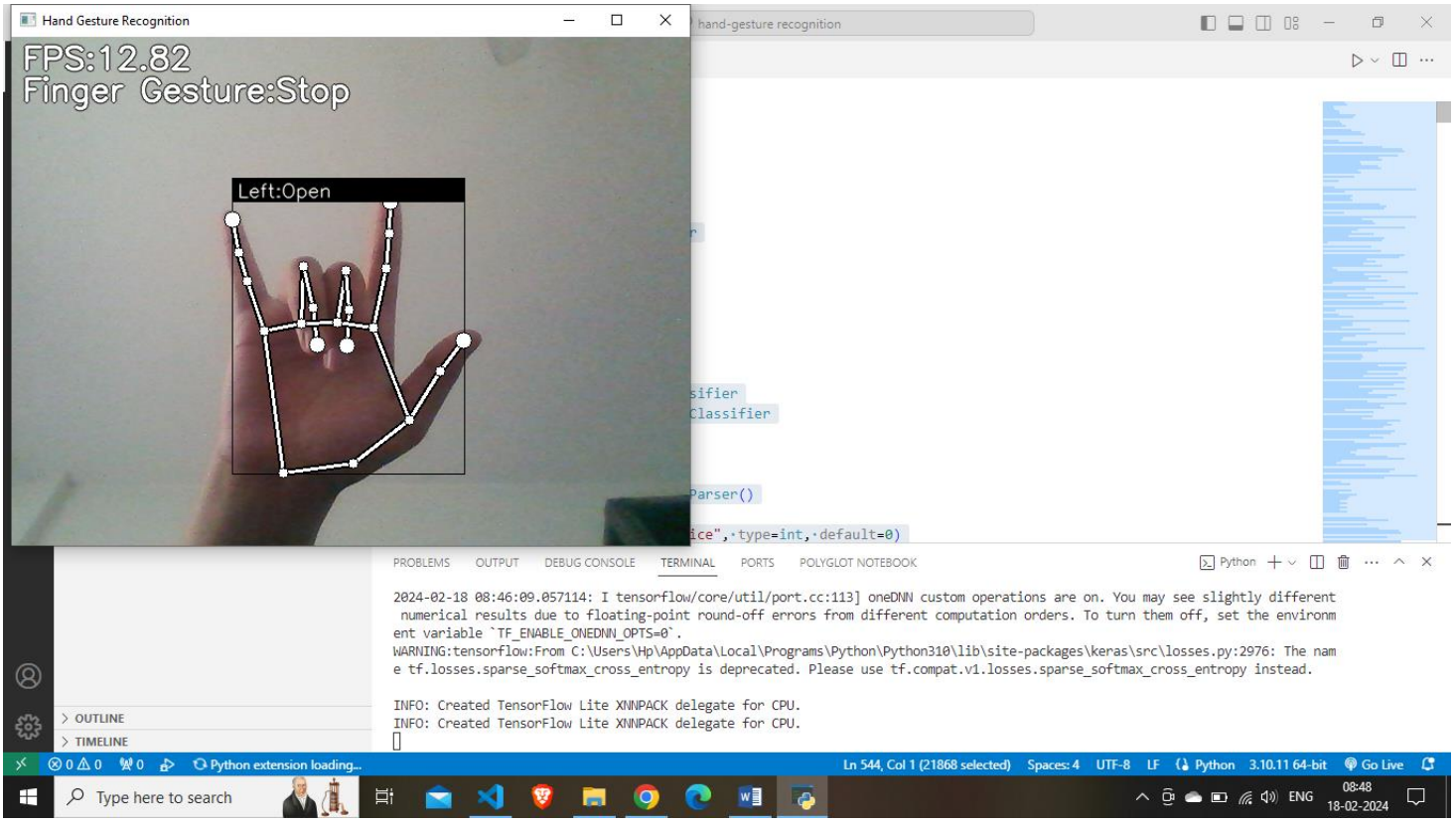
Workflow


Input gestures
Camera acquisition of images
Gesture detection and gesture tracking
Gesture segmentation
Hand trajectory feature extraction
Positioning of key points of the hand
Model training
System implementation

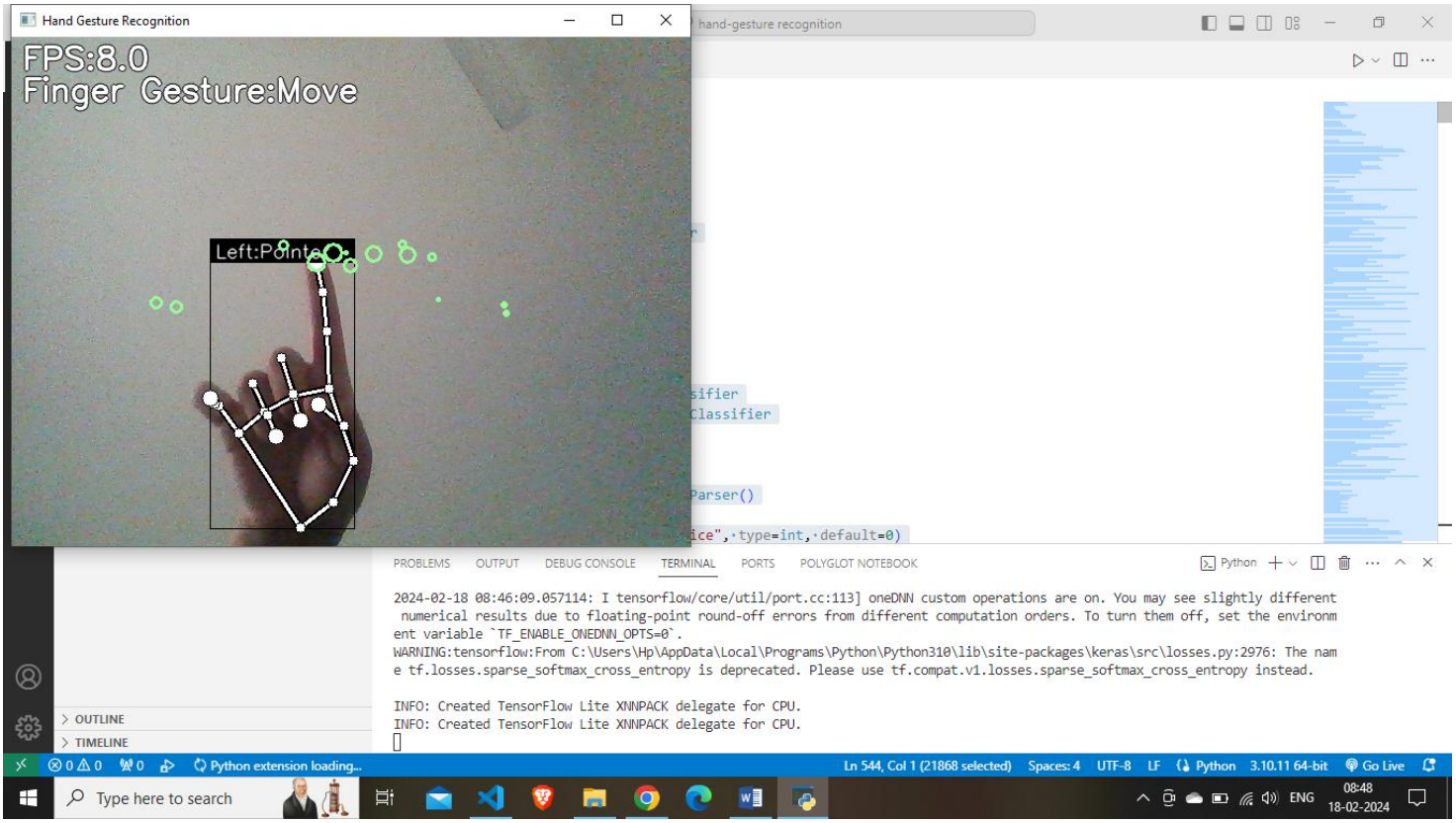
UI Snapshots



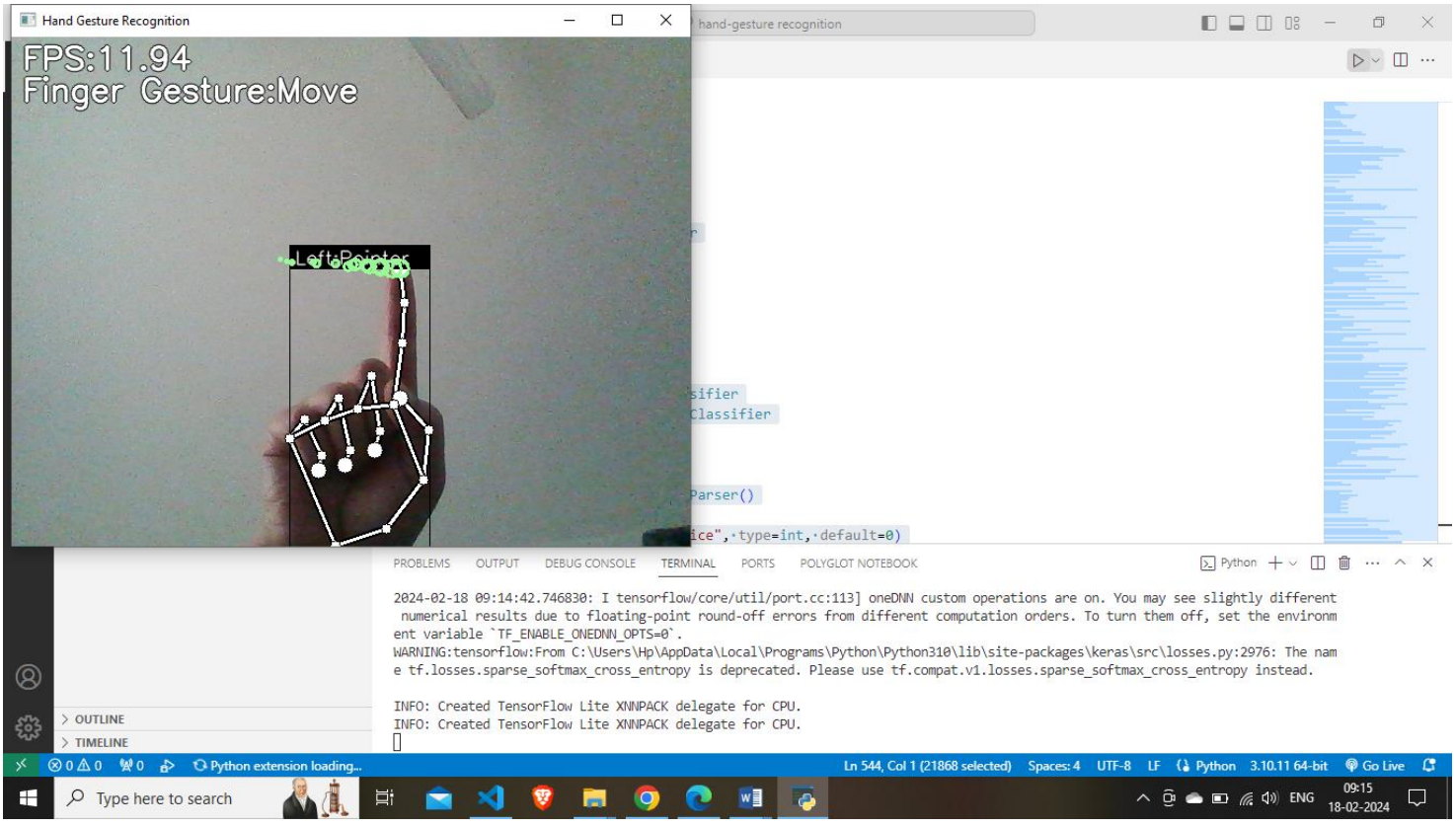
Snapshot 1



Snapshot 2



Snapshot 3



Snapshot 4

Code Snippets

app.py

app.py ●

app.py > draw_landmarks

```python
import csv
import copy
import argparse
import itertools
from collections import Counter
from collections import deque

import cv2 as cv
import numpy as np
import mediapipe as mp

from utils import CvFpsCalc
from model import KeyPointClassifier
from model import PointHistoryClassifier


def get_args():
    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, default=0)
    parser.add_argument("--width", help='cap width', type=int, default=960)
    parser.add_argument("--height", help='cap height', type=int, default=540)

    parser.add_argument('--use_static_image_mode', action='store_true')
    parser.add_argument("--min_detection_confidence",
                        help='min_detection_confidence',
                        type=float,
                        default=0.7)
    parser.add_argument("--min_tracking_confidence",
                        help='min_tracking_confidence',
                        type=int,
                        default=0.5)
```

Snippet 1

app.py ●

app.py > draw_landmarks

```
17  def get_args():
33
34      args = parser.parse_args()
35
36      return args
37
38
39  def main():
40      # Argument parsing #################################################################
41      args = get_args()
42
43      cap_device = args.device
44      cap_width = args.width
45      cap_height = args.height
46
47      use_static_image_mode = args.use_static_image_mode
48      min_detection_confidence = args.min_detection_confidence
49      min_tracking_confidence = args.min_tracking_confidence
50
51      use_brect = True
52
53      # Camera preparation ###############################################################
54      cap = cv.VideoCapture(cap_device)
55      cap.set(cv.CAP_PROP_FRAME_WIDTH, cap_width)
56      cap.set(cv.CAP_PROP_FRAME_HEIGHT, cap_height)
57
58      # Model load #############################################################
59      mp_hands = mp.solutions.hands
60      hands = mp_hands.Hands(
61          static_image_mode=use_static_image_mode,
62          max_num_hands=1,
63          min_detection_confidence=min_detection_confidence,
```

Snippet 2

app.py

app.py > draw_landmarks

```
39  def main():
64          min_tracking_confidence=min_tracking_confidence,
65      )
66
67      keypoint_classifier = KeyPointClassifier()
68
69      point_history_classifier = PointHistoryClassifier()
70
71      # Read labels ###########################################################
72      with open('model/keypoint_classifier/keypoint_classifier_label.csv',
73                encoding='utf-8-sig') as f:
74          keypoint_classifier_labels = csv.reader(f)
75          keypoint_classifier_labels = [
76              row[0] for row in keypoint_classifier_labels
77          ]
78      with open(
79              'model/point_history_classifier/point_history_classifier_label.csv',
80              encoding='utf-8-sig') as f:
81          point_history_classifier_labels = csv.reader(f)
82          point_history_classifier_labels = [
83              row[0] for row in point_history_classifier_labels
84          ]
85
86      # FPS Measurement ########################################################
87      cvFpsCalc = CvFpsCalc(buffer_len=10)
88
89      # Coordinate history #################################################################
90      history_length = 16
91      point_history = deque(maxlen=history_length)
92
93      # Finger gesture history ################################################
94      finger_gesture_history = deque(maxlen=history_length)
```

Snippet 3

app.py ●

app.py > draw_landmarks

```
39  def main():
95
96      #  ########################################################################
97      mode = 0
98
99      while True:
100         fps = cvFpsCalc.get()
101
102         # Process Key (ESC: end) #################################################
103         key = cv.waitKey(10)
104         if key == 27:  # ESC
105             break
106         number, mode = select_mode(key, mode)
107
108         # Camera capture #####################################################
109         ret, image = cap.read()
110         if not ret:
111             break
112         image = cv.flip(image, 1)  # Mirror display
113         debug_image = copy.deepcopy(image)
114
115         # Detection implementation #############################################################
116         image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
117
118         image.flags.writeable = False
119         results = hands.process(image)
120         image.flags.writeable = True
121
122         #  ####################################################################
123         if results.multi_hand_landmarks is not None:
124             for hand_landmarks, handedness in zip(results.multi_hand_landmarks,
125                                                   results.multi_handedness):
```

Snippet 4

app.py ●

app.py > draw_landmarks

```
39  def main():
126                 # Bounding box calculation
127                 brect = calc_bounding_rect(debug_image, hand_landmarks)
128                 # Landmark calculation
129                 landmark_list = calc_landmark_list(debug_image, hand_landmarks)
130
131                 # Conversion to relative coordinates / normalized coordinates
132                 pre_processed_landmark_list = pre_process_landmark(
133                     landmark_list)
134                 pre_processed_point_history_list = pre_process_point_history(
135                     debug_image, point_history)
136                 # Write to the dataset file
137                 logging_csv(number, mode, pre_processed_landmark_list,
138                             pre_processed_point_history_list)
139
140                 # Hand sign classification
141                 hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
142                 if hand_sign_id == 2:  # Point gesture
143                     point_history.append(landmark_list[8])
144                 else:
145                     point_history.append([0, 0])
146
147                 # Finger gesture classification
148                 finger_gesture_id = 0
149                 point_history_len = len(pre_processed_point_history_list)
150                 if point_history_len == (history_length * 2):
151                     finger_gesture_id = point_history_classifier(
152                         pre_processed_point_history_list)
153
154                 # Calculates the gesture IDs in the latest detection
155                 finger_gesture_history.append(finger_gesture_id)
156                 most_common_fg_id = Counter(
```

Snippet 5

app.py

app.py > draw_landmarks

```
39  def main():
157                     finger_gesture_history).most_common()
158
159                 # Drawing part
160                 debug_image = draw_bounding_rect(use_brect, debug_image, brect)
161                 debug_image = draw_landmarks(debug_image, landmark_list)
162                 debug_image = draw_info_text(
163                     debug_image,
164                     brect,
165                     handedness,
166                     keypoint_classifier_labels[hand_sign_id],
167                     point_history_classifier_labels[most_common_fg_id[0][0]],
168                 )
169             else:
170                 point_history.append([0, 0])
171
172             debug_image = draw_point_history(debug_image, point_history)
173             debug_image = draw_info(debug_image, fps, mode, number)
174
175             # Screen reflection #############################################################
176             cv.imshow('Hand Gesture Recognition', debug_image)
177
178         cap.release()
179         cv.destroyAllWindows()
180
181
182     def select_mode(key, mode):
183         number = -1
184         if 48 <= key <= 57:  # 0 ~ 9
185             number = key - 48
186         if key == 110:  # n
187             mode = 0
```

Snippet 6

app.py ●

app.py > draw_landmarks

```python
def select_mode(key, mode):
    if key == 107:  # k
        mode = 1
    if key == 104:  # h
        mode = 2
    return number, mode


def calc_bounding_rect(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

    landmark_array = np.empty((0, 2), int)

    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)

        landmark_point = [np.array((landmark_x, landmark_y))]

        landmark_array = np.append(landmark_array, landmark_point, axis=0)

    x, y, w, h = cv.boundingRect(landmark_array)

    return [x, y, x + w, y + h]


def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

    landmark_point = []

    # Keypoint
```

Snippet 7

app.py

app.py > draw_landmarks

```python
def calc_landmark_list(image, landmarks):
    # Keypoint
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)
        # landmark_z = landmark.z

        landmark_point.append([landmark_x, landmark_y])

    return landmark_point


def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates
    base_x, base_y = 0, 0
    for index, landmark_point in enumerate(temp_landmark_list):
        if index == 0:
            base_x, base_y = landmark_point[0], landmark_point[1]

        temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
        temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
    temp_landmark_list = list(
        itertools.chain.from_iterable(temp_landmark_list))

    # Normalization
    max_value = max(list(map(abs, temp_landmark_list)))

    def normalize_(n):
        return n / max_value
```

Snippet 8

app.py ●

app.py > draw_landmarks

```
229  def pre_process_landmark(landmark_list):
250
251      temp_landmark_list = list(map(normalize_, temp_landmark_list))
252
253      return temp_landmark_list
254
255
256  def pre_process_point_history(image, point_history):
257      image_width, image_height = image.shape[1], image.shape[0]
258
259      temp_point_history = copy.deepcopy(point_history)
260
261      # Convert to relative coordinates
262      base_x, base_y = 0, 0
263      for index, point in enumerate(temp_point_history):
264          if index == 0:
265              base_x, base_y = point[0], point[1]
266
267          temp_point_history[index][0] = (temp_point_history[index][0] -
268                                          base_x) / image_width
269          temp_point_history[index][1] = (temp_point_history[index][1] -
270                                          base_y) / image_height
271
272      # Convert to a one-dimensional list
273      temp_point_history = list(
274          itertools.chain.from_iterable(temp_point_history))
275
276      return temp_point_history
277
278
279  def logging_csv(number, mode, landmark_list, point_history_list):
280      if mode == 0:
```

Snippet 9

app.py

app.py > draw_landmarks

```python
def logging_csv(number, mode, landmark_list, point_history_list):
    pass
    if mode == 1 and (0 <= number <= 9):
        csv_path = 'model/keypoint_classifier/keypoint.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_list])
    if mode == 2 and (0 <= number <= 9):
        csv_path = 'model/point_history_classifier/point_history.csv'
        with open(csv_path, 'a', newline="") as f:
            writer = csv.writer(f)
            writer.writerow([number, *point_history_list])
    return


def draw_landmarks(image, landmark_point):
    if len(landmark_point) > 0:
        # Thumb
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[3]),
                (255, 255, 255), 2)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[3]), tuple(landmark_point[4]),
                (255, 255, 255), 2)

        # Index finger
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
                (0, 0, 0), 6)
        cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[6]),
                (255, 255, 255), 2)
```

Snippet 10

app.py

app.py > draw_landmarks

```
295  def draw_landmarks(image, landmark_point):
312          cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
313                  (0, 0, 0), 6)
314          cv.line(image, tuple(landmark_point[6]), tuple(landmark_point[7]),
315                  (255, 255, 255), 2)
316          cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
317                  (0, 0, 0), 6)
318          cv.line(image, tuple(landmark_point[7]), tuple(landmark_point[8]),
319                  (255, 255, 255), 2)
320
321          # Middle finger
322          cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
323                  (0, 0, 0), 6)
324          cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[10]),
325                  (255, 255, 255), 2)
326          cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
327                  (0, 0, 0), 6)
328          cv.line(image, tuple(landmark_point[10]), tuple(landmark_point[11]),
329                  (255, 255, 255), 2)
330          cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
331                  (0, 0, 0), 6)
332          cv.line(image, tuple(landmark_point[11]), tuple(landmark_point[12]),
333                  (255, 255, 255), 2)
334
335          # Ring finger
336          cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
337                  (0, 0, 0), 6)
338          cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[14]),
339                  (255, 255, 255), 2)
340          cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
341                  (0, 0, 0), 6)
342          cv.line(image, tuple(landmark_point[14]), tuple(landmark_point[15]),
```

Snippet 11

app.py

app.py > draw_landmarks

```
295  def draw_landmarks(image, landmark_point):
343              (255, 255, 255), 2)
344          cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
345              (0, 0, 0), 6)
346          cv.line(image, tuple(landmark_point[15]), tuple(landmark_point[16]),
347              (255, 255, 255), 2)
348
349          # Little finger
350          cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
351              (0, 0, 0), 6)
352          cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[18]),
353              (255, 255, 255), 2)
354          cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
355              (0, 0, 0), 6)
356          cv.line(image, tuple(landmark_point[18]), tuple(landmark_point[19]),
357              (255, 255, 255), 2)
358          cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
359              (0, 0, 0), 6)
360          cv.line(image, tuple(landmark_point[19]), tuple(landmark_point[20]),
361              (255, 255, 255), 2)
362
363          # Palm
364          cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
365              (0, 0, 0), 6)
366          cv.line(image, tuple(landmark_point[0]), tuple(landmark_point[1]),
367              (255, 255, 255), 2)
368          cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
369              (0, 0, 0), 6)
370          cv.line(image, tuple(landmark_point[1]), tuple(landmark_point[2]),
371              (255, 255, 255), 2)
372          cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
373              (0, 0, 0), 6)
```

Snippet 12

app.py ●

app.py > draw_landmarks

```
295  def draw_landmarks(image, landmark_point):
374          cv.line(image, tuple(landmark_point[2]), tuple(landmark_point[5]),
375                  (255, 255, 255), 2)
376          cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
377                  (0, 0, 0), 6)
378          cv.line(image, tuple(landmark_point[5]), tuple(landmark_point[9]),
379                  (255, 255, 255), 2)
380          cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
381                  (0, 0, 0), 6)
382          cv.line(image, tuple(landmark_point[9]), tuple(landmark_point[13]),
383                  (255, 255, 255), 2)
384          cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
385                  (0, 0, 0), 6)
386          cv.line(image, tuple(landmark_point[13]), tuple(landmark_point[17]),
387                  (255, 255, 255), 2)
388          cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
389                  (0, 0, 0), 6)
390          cv.line(image, tuple(landmark_point[17]), tuple(landmark_point[0]),
391                  (255, 255, 255), 2)
392
393      # Key Points
394      for index, landmark in enumerate(landmark_point):
395          if index == 0:
396              cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
397                        -1)
398              cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
399          if index == 1:
400              cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
401                        -1)
402              cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
403          if index == 2:
404              cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
```

Snippet 13

app.py ●

app.py > draw_landmarks

```
295   def draw_landmarks(image, landmark_point):
405                   -1)
406           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
407       if index == 3:
408           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
409                     -1)
410           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
411       if index == 4:
412           cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
413                     -1)
414           cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
415       if index == 5:
416           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
417                     -1)
418           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
419       if index == 6:
420           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
421                     -1)
422           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
423       if index == 7:
424           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
425                     -1)
426           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
427       if index == 8:
428           cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
429                     -1)
430           cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
431       if index == 9:
432           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
433                     -1)
434           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
435       if index == 10:
```

Snippet 14

app.py

app.py > draw_landmarks

```
295   def draw_landmarks(image, landmark_point):
436           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
437                     -1)
438           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
439       if index == 11:
440           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
441                     -1)
442           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
443       if index == 12:
444           cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
445                     -1)
446           cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
447       if index == 13:
448           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
449                     -1)
450           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
451       if index == 14:
452           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
453                     -1)
454           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
455       if index == 15:
456           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
457                     -1)
458           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
459       if index == 16:
460           cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
461                     -1)
462           cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
463       if index == 17:
464           cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
465                     -1)
466           cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
```

Snippet 15

app.py

app.py > draw_landmarks

```
295  def draw_landmarks(image, landmark_point):
467          if index == 18:
468              cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
469                        -1)
470              cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
471          if index == 19:
472              cv.circle(image, (landmark[0], landmark[1]), 5, (255, 255, 255),
473                        -1)
474              cv.circle(image, (landmark[0], landmark[1]), 5, (0, 0, 0), 1)
475          if index == 20:
476              cv.circle(image, (landmark[0], landmark[1]), 8, (255, 255, 255),
477                        -1)
478              cv.circle(image, (landmark[0], landmark[1]), 8, (0, 0, 0), 1)
479
480      return image
481
482
483  def draw_bounding_rect(use_brect, image, brect):
484      if use_brect:
485          # Outer rectangle
486          cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[3]),
487                       (0, 0, 0), 1)
488
489      return image
490
491
492  def draw_info_text(image, brect, handedness, hand_sign_text,
493                     finger_gesture_text):
494      cv.rectangle(image, (brect[0], brect[1]), (brect[2], brect[1] - 22),
495                   (0, 0, 0), -1)
496
497      info_text = handedness.classification[0].label[0:]
```

Snippet 16

app.py ●

app.py > draw_landmarks

```
492   def draw_info_text(image, brect, handedness, hand_sign_text,
498       if hand_sign_text != "":
499           info_text = info_text + ':' + hand_sign_text
500       cv.putText(image, info_text, (brect[0] + 5, brect[1] - 4),
501                  cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1, cv.LINE_AA)
502
503       if finger_gesture_text != "":
504           cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
505                      cv.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 0), 4, cv.LINE_AA)
506           cv.putText(image, "Finger Gesture:" + finger_gesture_text, (10, 60),
507                      cv.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), 2,
508                      cv.LINE_AA)
509
510       return image
511
512
513   def draw_point_history(image, point_history):
514       for index, point in enumerate(point_history):
515           if point[0] != 0 and point[1] != 0:
516               cv.circle(image, (point[0], point[1]), 1 + int(index / 2),
517                         (152, 251, 152), 2)
518
519       return image
520
521
522   def draw_info(image, fps, mode, number):
523       cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
524                  1.0, (0, 0, 0), 4, cv.LINE_AA)
525       cv.putText(image, "FPS:" + str(fps), (10, 30), cv.FONT_HERSHEY_SIMPLEX,
526                  1.0, (255, 255, 255), 2, cv.LINE_AA)
527
528       mode_string = ['Logging Key Point', 'Logging Point History']
```

Snippet 17

app.py ●

app.py > draw_landmarks

```
522  def draw_info(image, fps, mode, number):
527
528      mode_string = ['Logging Key Point', 'Logging Point History']
529      if 1 <= mode <= 2:
530          cv.putText(image, "MODE:" + mode_string[mode - 1], (10, 90),
531                     cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
532                     cv.LINE_AA)
533          if 0 <= number <= 9:
534              cv.putText(image, "NUM:" + str(number), (10, 110),
535                         cv.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1,
536                         cv.LINE_AA)
537      return image
538
539
540  if __name__ == '__main__':
541      main()
542
```

Snippet 18

Future Work

- **Increase Gesture Repertoire:** We can train our system to recognize more complex or custom gestures.
- **Enhance Recognition Accuracy:** We can improve the robustness of our system by incorporating techniques to handle variations in lighting, background clutter, and hand posture.
- **Multi-Hand Tracking:** We can extend our project to recognize gestures from both hands simultaneously. This opens up possibilities for more intricate interactions.
- **3D Hand Pose Estimation:** We can take our project a step further by estimating the 3D pose of the hand.
- **Gesture-Controlled Applications:** We can integrate our gesture recognition with an application - control a media player, navigate a web interface, or even design a virtual reality experience.
- **Combined Input with Other Sensors:** We can explore how hand gestures can interact with other sensors like voice commands or head tracking for a richer user experience.

CONCLUSION

This project uses Mediapipe, OpenCV and Tensorflow for recognizing the hand gestures. This project was made using Python. When a user makes a gesture, it detects the hand gesture, recognizes it and displays the frames per second along with the detected gesture to the user.

REFERENCES

[1]Paulo Trigueiros, “Computer Vision and Machine Learning based Hand Gesture Recognition”, 2015

[2]Geron Aurelien, “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow”, 2022

[3]Nishant Shukla, “Machine Learning with TensorFlow”, 2018

[4]Adrian Kaehler, “Learning OpenCV”, 2008

[5]Andreas Muller, “Introduction to Machine Learning with Python”, 2016