Interface Programming
CIS3149

James Fletcher

25397869

1 TABLE OF CONTENTS

2 TASK ONE

2.1 INTRODUCTION

The intention of this task is to outline how human computer interaction (HCI) have evolved throughout their lifespan. HCI is exactly what the name suggests, it is the method in which the user interacts with the computer. The methods of application within the HCI field have expanded tremendously over time as Shneiderman is put forth in Rodgers' (2012). Carrol (2009) stating that HCI grew in popularity on conjunction with the emergence of personal computing towards the latter stages of the 1970's.

2.2 TIMELINE

HCI has improved massively since the inception of computers from the earlier form of calculator to computer-to-computer interaction as demonstrated in the more modern programming usable interface (PUI).

The first believed computer was the calculating clock built by Wilhelm Schickard in 1623 (Freiberger and Swaine, n.d.). This calculator worked using 9 wooden slats comprising of numbers as well as 6 cylinders spanning the front of the machine with Napier's logs laid over top (Freid and Sweitzer-Lamme, 2014). Napier's log is an early mathematical logarithm that focused on the idea of geometric progression (The Open University, n.d.), which upon application to this early form of calculator would provide reason for the given output. In 1920, Thomas de Colmar progressed the calculator by inventing and producing the Arithmometer, which would become the first calculator to be mass produced, with production being maintained for 90 years (Freiberger and Swaine, n.d.).

The Jacquard loom punch card machine was a revolutionary step in computing as this machine allowed detailed patterns to be etched into fabrics and mass produced for the first time in history. The use of binary for this machine opened many doors for similar methods to be used and expanded upon going forward, which would become apparent when IBM released the IBM computer card in 1928. This improved punch card system would include 45 columns and 12 punch positions which would allow for much larger stores of data as well as the use in more complex tasks due to the ability to write lines of code (IBM, n.d.).

In 1945, Mauchly and Eckert created the ENIAC (Electronic Numerical Integrator and Computer). This revolutionary machine was the first general-purpose electronic computer (University of Pennsylvania, n.d.), taking up a 1,500 square foot room and consisted of over 70,000 resistors, 17,000 vacuum tubes, and 10,000 capacitors. The primary focus of this machine was to calculate artillery range tables however, the flexibility of the machine meant it was capable of being reprogramed for many other uses (HP, n.d.).

Another machine that was developed during the second World War was the Enigma Machine. This infamous deciphering machine was integral to the war efforts against the Axis of Power as through the use of this machine, the British were able to decipher German communications in order to counteract any plans they were making on the war front. The Enigma machines settings gave 15 quadrillion possible solutions however by the end of the war "the British were reading 10 percent of all German Enigma communications" (CIA, n.d.).

In 1945, 30 years before the invention of the personal computer and 50 years before the advent of the world wide web (MIT, n.d.), Vannevar Bush put forth the idea of a device in which "an individual

stores all his books, records and communications which is mechanized so that it may be consulted with exceeding speed and flexibility" (Bush, 1945). This machine which he coined the "Memex" would be a revolutionary method of supplementing one's memory as they suddenly became able to store and retrieve data in a way that was never previously possible. Whilst the implementation of this method in the present is vastly different that the technology available at the time, the principle of the idea is vital to how the world operates on a day-to-day basis.

Time sharing was an idea put forth to allow multiple users to access a computer system without interrupting each other. This idea was put forth by John Backus in 1955, who theorised that the large computers could be used as several small ones (IBM, n.d.). This theory would be put into practice in the early 1960's as IBM incorporated keyboards and individual terminals to allow many people to work without interruption (McCarthy, 1989).

The first interactive computer graphics program "Sketchpad," was designed but Ivan Sutherland in the early 1960's. This program allowed users to "visualise and control program functions" which would become a foundation of computer graphics and operating system interfaces (Pyfer, n.d.). Rodden and Blackwell (2003), claim the Sketchpad had a long-standing effect on how computers were perceived and had massive impact on a multitude of new evolutions including some of the current market leaders Macintosh and Windows. Sutherland (1963) claimed in his doctoral thesis that the Sketchpad allows for communications between man and machine to be slowed down tremendously as they become able to use line drawings instead of written statements (Sutherland, 1963).

One of the most impactful inventions to take place within the computer industry was completed by Douglas Englelbart with the invention of the mouse. Engelbart's computer mouse, originally patented as the "X-Y Position Indicator for a Display System" (SRI, n.d.), consisted of a block of wood with a pair of metal wheels underneath that would track the X and Y movement of the mouse and a button(s) on top. Engelbart's original vision for the project was to "broaden the connection between humans and computers" (Doug Engelbart Institute, n.d.). The mouse was so revolutionary that many personal computer systems still use a mouse as a primary form of control, although a much more refined and efficient model. Engelbart's inventions did not stop with just the computer mouse, he also revolutionised the use of a keyboard, which before hand operated similarly to a type writer, to include the ability to delete keys also known as the backspace. He also implemented the copy and paste function and the ability to save files independently. Also, he showcased the ability to drag and move items within a list as well as the use of hypertext to link to other datasets (Landau, 2018). Not only did Engelbart innovate computing to a level that had rarely been seen, he announced all of these revolutionary ideas within one single demonstration referred to as the "Mother of All Demos" where he announced his oN-Line System (NLS) (Doug Engelbart Institute, n.d.).

Alan Kay was one of the pioneers that pushed computers towards a household item. His idea was to create a smaller computer that were easy enough for children to operate. He released a mock up design of a computer that was a flat panel display with a stylus, similar to a modern-day tablet (Barnes, n.d.). Whilst Kay did not manage to bring forth his vision into reality due to limited technology, his idea was eventually brought into fruition.

In 1965, Ted Nelson invented a model for creating and using linked context which he coined "hypertext"(Computing History, n.d.). This hypertext changed how information could be stored and accessed as for the first time, it allowed for data to be linked other data very easily. He also coined the term "hypermedia," which has the same premise, except for the linking data would be elements

such as graphics, video, and sound (W3C, n.d.). This idea is a vital part of how people manoeuvre the internet in the present, through the use of “hyperlinks” that link to other websites.

In 1975, IBM released their first portable computer the IBM 5100, which weighed 50 pounds and cost $18,000, which would equate to over $100,000 dollars in 2025 adjusted for inflation (HistoryOfInformation, n.d.). The “high cost and lack of interfacing capability” (Berg, n.d.) lead this to pale in comparison to the IBM 5150 Personal Computer which was released just 6 years later in 1981. Whilst IBM had made a name by creating a shipping all of their own products, in order to match the given timeline and cost scale, they purchased Intel’s 8088 chip and used Microsoft’s operating system. This allowed them to release their PC within the year as James Cortada (2019 cited in IBM (n.d.) claimed “the PC market was moving too quickly” to take normal means at risk of falling behind competitors. The computer was so well received that TIME magazine announced it as “Machine of the Year” replacing the annual “Man of the Year” and remains the only machine ever to win the award (McCracken, 2013).

The Xerox Alto was a personal computer not like any others upon its release in 1973. Many of the qualities that are taken for granted in current computers were available with this computer starting with a high-resolution screen, keyboard, and mouse. However, more than 50 years ago, these were far from a staple for a household to have. Another aspect which set the Alto apart from the others of its kind it that much like the computers of today, this PC was built as a tower computer to be kept under a desk with all the peripherals, mouse, keyboard and monitor, to be kept above (Brock, 2023).

WIMP is a graphical user interface invented at Xerox Parc, the creator of the Alto, and stands for Windows, Icons, Menus and Pointers (Teach-ICT, n.d.). The WIMP system was popularised by the Apple Macintosh and has since been co-opted by many others including being used for the Windows operating system (Interaction Design Foundation, n.d.). One of the main benefits of the WIMP system is that it is very beginner friendly as it a WYSIWYG (What You See Is What You Get) software, meaning that there will be no changes in visuals upon configuring an output, such as printing from a text editor (Kanade, 2023).

The Apple Macintosh 128K was originally described as “revolutionary” by the New York Times, however, due to the large price and low specs, it was quickly superseded by the next generation of PC’s. Isaacson cited in the LA Times (2014), states that the machine was a “woefully slow and underpowered computer”. One of the major selling points when it came to the Macintosh was the user-friendly interface that became one the first widely sold PC with a GUI and a mouse (Kirvan, n.d.). In 1986, Apple released the much improved Macintosh Plus, which included 1 megabyte of memory. With the improved capability of the new Mac, it opened up many possibilities to run new software, which made it very successful with creative people (O’Brien, 2014).

Few industries have progressed as quickly as the mobile phone industry. The original mobile phones consisted of a microphone, a speaker, an antenna, and could weight up to a kilogram. As they progressed, they got thinner and lighter, features have been added and removed. Some of the large changes made include the era of flip phones and slide phones. These phones were more lightweight and portable as more and more people got mobile phones, however they were quickly outclassed by phones such as the Blackberry and the iPhone as the Blackberry was an ideal candidate for the working man, whereas there were few alternatives that offered the personalisation of the iPhone. Whilst there have been other phones to challenge the iPhone since its release, there have been few if any challengers outside of the touch screen bracket. The modern day mobile phones are more powerful than dedicated personal computers of the past, with the iPhone 12 being roughly 5,000

times faster than the CRAY-2 Supercomputer designed for the United States Departments of Defence and Energy in 1980 (Adobe Acrobat Team, 2022).

3 TASK TWO

3.1 INTRODUCTION

The objective of this task is to use creativity in order to create a scene in OpenCV using primitive shapes such as rectangles and polygons. The approach that was decided upon for this task was to create a football net and accompanying pitch with an animation on the ball that would move it into the net. The reasoning behind this decision was to explore how perspective can be achieved with primitive shapes and a limited background in the subject.

3.2 PROJECT AT WORK

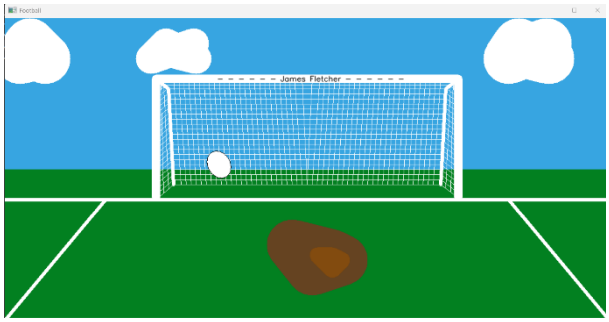

Figure 1 - Football Drawing Frame One

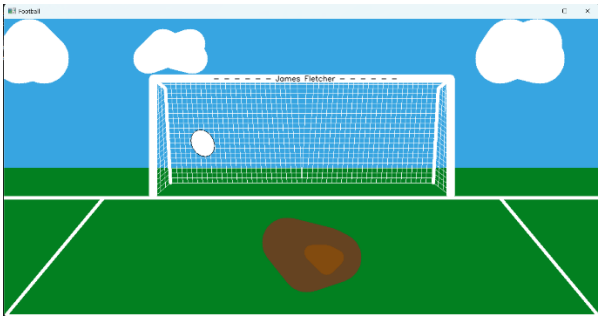

Figure 2 - Football Drawing Frame Two

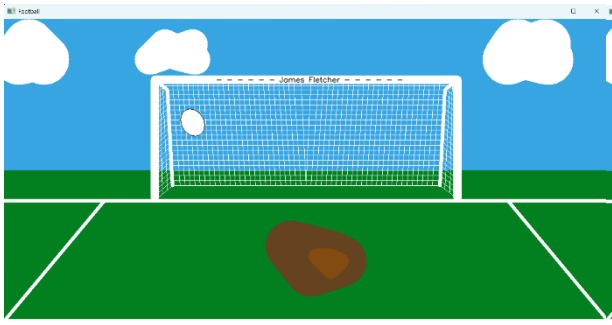

Figure 3 - Football Drawing Frame Three

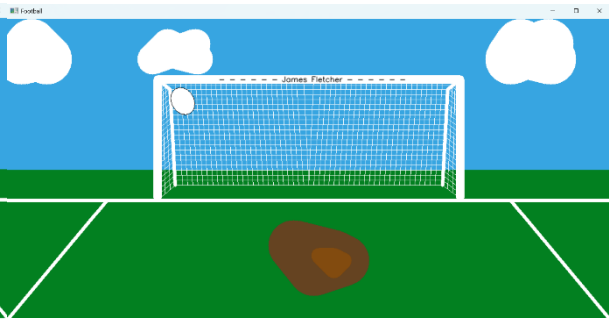

Figure 4 - Football Drawing Frame Four

3.3 SOURCE CODE

```
#changes bg colour
class Sky:
    def draw(self, bg):
        cv2.rectangle(bg, (0, 0), (1200, 300), sky_blue, -1)

#initializes polygon shape for cloud
class Cloud:
    def __init__(self, points, thickness=100):
        self.points = np.array(points, np.int32).reshape((-1, 1, 2))
        self.thickness = thickness

    def draw(self, bg):
        cv2.polylines(bg, [self.points], True, white, self.thickness)

#draws pitch
class Pitch:
    def draw(self, bg):
        cv2.rectangle(bg, (0, 600), (1200, 300), grass_green, -1)
        cv2.line(bg, (0, 600), (200, 360), white, 5)
        cv2.line(bg, (1200, 600), (1000, 360), white, 5)
        cv2.line(bg, (0, 360), (1200, 360), white, 5)
        cv2.line(bg, (0, 598), (1200, 598), white, 5)
```

Figure 7 - Code for Scene Elements (1)

```
#draws goal posts then each netting
class Goal:
    def draw(self, bg):
        # Goalposts
        cv2.line(bg, (300, 355), (300, 120), white, 15)
        cv2.line(bg, (900, 355), (900, 120), white, 15)
        cv2.line(bg, (300, 120), (900, 120), white, 15)
        cv2.line(bg, (300, 120), (325, 140), white, 5)
        cv2.line(bg, (900, 120), (875, 140), white, 5)
        cv2.line(bg, (325, 140), (335, 330), white, 5)
        cv2.line(bg, (875, 140), (865, 330), white, 5)

        #net
        #center netting
        cv2.line(bg,(315,132), (885,132), white, 1)
        cv2.line(bg,(325,140), (875,140), white, 1)
        cv2.line(bg,(325,150), (875,150), white, 1)
        cv2.line(bg,(325,160), (875,160), white, 1)
        cv2.line(bg,(325,170), (875,170), white, 1)
        cv2.line(bg,(325,180), (875,180), white, 1)
        cv2.line(bg,(325,190), (875,190), white, 1)
        cv2.line(bg,(325,200), (875,200), white, 1)
        cv2.line(bg,(325,210), (875,210), white, 1)
        cv2.line(bg,(330,220), (870,220), white, 1)
        cv2.line(bg,(330,230), (870,230), white, 1)
        cv2.line(bg,(330,240), (870,240), white, 1)
        cv2.line(bg,(335,250), (865,250), white, 1)
        cv2.line(bg,(335,260), (865,260), white, 1)
        cv2.line(bg,(335,270), (865,270), white, 1)
        cv2.line(bg,(335,280), (865,280), white, 1)
        cv2.line(bg,(335,290), (865,290), white, 1)
        cv2.line(bg,(335,300), (865,300), white, 1)
        cv2.line(bg,(335,310), (865,310), white, 1)
        cv2.line(bg,(335,320), (865,320), white, 1)
        cv2.line(bg,(335,330), (865,330), white, 1)
```

Figure 5 - Code for Scene Elements (2)

```
#writes name on goalposts
class Name:
    def draw(self, bg):
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(bg,' - - - - - - James Fletcher - - - - - - ',(413,125), font, 0.5,black,1,cv2.LINE_AA)


class DirtPatch:
    def __init__(self, points, color, thickness):
        self.points = np.array(points, np.int32).reshape((-1, 1, 2))
        self.color = color
        self.thickness = thickness

    def draw(self, bg):
        cv2.polylines(bg, [self.points], True, self.color, self.thickness)
```

Figure 6 - Code for Scene Elements (3)

```python
class Ball:
    def __init__(self, x=350,y=163):
        self.positions = [(425, 290), (400, 250), (375, 205), (350, 163)]
        self.index = 0  # Start at the first position

    def update_position(self, frame_count):
        """Change position every 10 frames by cycling through the list"""
        if frame_count % 10 == 0:
            self.index = (self.index + 1) % len(self.positions)  # cycle through positions

    def draw(self, bg):
        x1, y1 = self.positions[self.index]     #position data for white fill
        x2, y2 = self.positions[self.index]     #position data for black outline

        cv2.ellipse(bg, (x1, y1), (22, 28), -30, 0, 360, white, -1)  # white fill
        cv2.ellipse(bg, (x2, y2), (22, 28), -30, 0, 360, black, 1)   # black outline
```

Figure 8 - Code for Scene Elements (4)

```python
#initializes scene essentials
class Scene:
    def __init__(self, width=1200, height=600):
        self.bg = np.zeros((height, width, 3), np.uint8)
        self.elements = []
        self.width = width
        self.height = height
        self.ball = Ball()
        self.frame_count = 0

    def add_element(self, element):
        self.elements.append(element)

    def update(self):
        self.frame_count += 1
        self.ball.update_position(self.frame_count)  # move ball

    def draw(self):
        frame = self.bg.copy()  # start with a clean background

        # draw all scene elements
        for element in self.elements:
            element.draw(frame)

        self.ball.draw(frame)  # draw ball on top

        return frame
```

Figure 10 - Scene Logic

```python
# Add elements to the scene
scene.add_element(Sky())
scene.add_element(Cloud([[30, 80], [50, 50], [80, 80], [70, 70]], 100))
scene.add_element(Cloud([[290, 80], [320, 50], [380, 80], [340, 70], [375, 50]], 60))
scene.add_element(Cloud([[1000, 80], [1020, 50], [1080, 80], [1040, 70], [1075, 50]], 100))
scene.add_element(Pitch())
scene.add_element(Goal())
scene.add_element(Name())
scene.add_element(DirtPatch([[570, 450], [610, 500], [670, 480], [650, 470]], dark_brown, 100))
scene.add_element(DirtPatch([[620, 470], [650, 500], [670, 480], [650, 470]], brown, 30))
```

Figure 9 - Adding Elements to Scene

4 TASK THREE

4.1 INTRODUCTION

The aim of this task was to use the provided Hand/Face tracking code to build a software that would take assumed gesture and use them in a creative manner. For this task, the gesture would display on screen, and if the person on camera would open their mouth, an audio file would activate.

4.2 PROJECT AT WORK

https://youtu.be/OGHjMrPO_Js

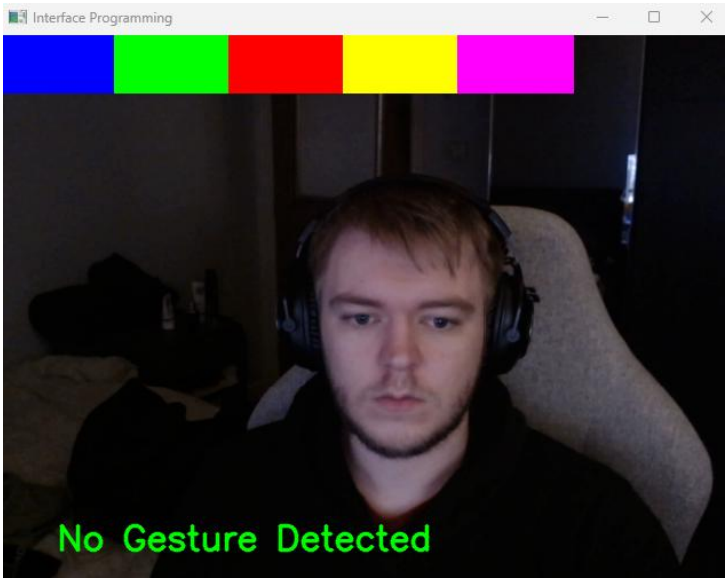

Figure 11 - No Gesture

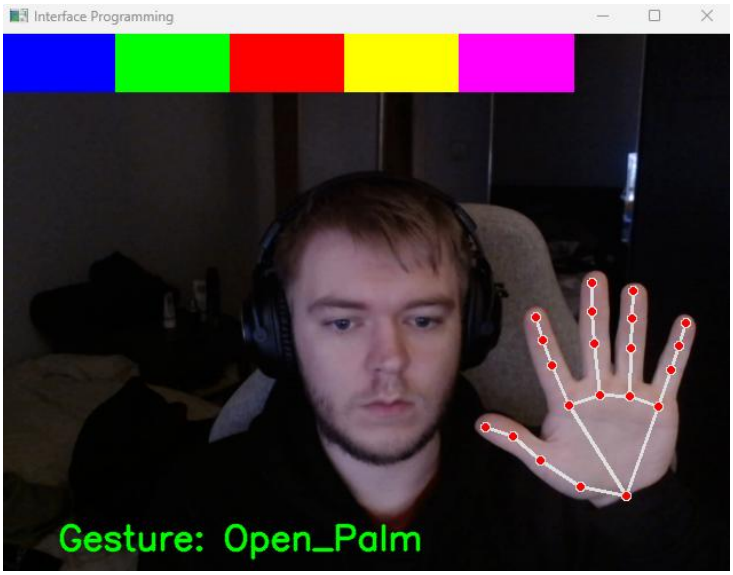

Figure 12 - Open Palm

Figure 13 - Closed Fist

4.3 Source Code

```python
#class for recognising gestures
class GestureRecognition:
    def __init__(self, model_path):
        self.options = GestureRecognizerOptions(
            base_options=BaseOptions(model_asset_path=model_path),
            running_mode=VisionRunningMode.IMAGE
        )
        self.recognizer = GestureRecognizer.create_from_options(self.options)

    def recognize_gesture(self, image):
        #recognises the gesture in the image
        #returns a list of detected gestures / confident score
        try:
            mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=image)
            result = self.recognizer.recognize(mp_image)
            gestures = []
            if result.gestures:
                for gesture in result.gestures:
                    name = gesture[0].category_name
                    score = gesture[0].score
                    gestures.append((name, score))

            return gestures
        except Exception as e:
            print(f"Error in Gesture Recognition: {e}")
            return []
```

Figure 14 - Gesture Recogniser Logic

```python
#class for detecting landmarks in the face
class FaceRecognition:
    def __init__(self):
        self.face_detection = mp_face_detection.FaceDetection(model_selection=0, min_detection_confidence=0.5)
        self.face_mesh = mp_face_mesh.FaceMesh(refine_landmarks=True, min_detection_confidence=0.5)

    def detect_face(self, image):
        return self.face_detection.process(image)

    def detect_face_mesh(self, image):
        return self.face_mesh.process(image)
```

Figure 15 - Face Recognition Logic

```python
#class for detecting landmarks in the hand
class HandRecognition:
    def __init__(self):
        self.hands = mp_hands.Hands(
            model_complexity=0,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5
        )

    def detect_hands(self, image):
        return self.hands.process(image)
```

Figure 18 - Hand Recogniser Logic

```python
#initialize pygame to play audio
pygame.mixer.init()
mouth_sound = "interface_audio.mp3"  # Change this to your audio file
pygame.mixer.music.load(mouth_sound)
```

Figure 17 - Audio Initialisation

```python
#function to check if the mouth is open
def is_mouth_open(face_landmarks, threshold=0.05):
    if face_landmarks:
        # Get landmarks for upper & lower lips
        upper_lip = face_landmarks.landmark[13]  # Upper lip (middle)
        lower_lip = face_landmarks.landmark[14]  # Lower lip (middle)


        mouth_open_ratio = abs(upper_lip.y - lower_lip.y)

        return mouth_open_ratio > threshold  #returns True if mouth is open
    return False
```

Figure 16 - Function to Check if Mouth is Open

```
while cap.isOpened():
    success, image = cap.read()
    if not success:
        continue

    image = cv2.flip(image, 1)  # flips the image so movement appears correct
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #convert from BGR to RGB for mediapipe
```

Figure 21 - While Loop Initialisation

```
#checks if mout is open and plays sound
if face_mesh_results.multi_face_landmarks:
    for face_landmarks in face_mesh_results.multi_face_landmarks:

        if is_mouth_open(face_landmarks) and not mouth_open:
            mouth_open = True
            pygame.mixer.music.play()

        elif not is_mouth_open(face_landmarks):
            mouth_open = False  #reset when mouth closes
```

Figure 20 - Logic for Playing Audio on Mouth Open

```
    #adds drawings on camera
    output = cv2.addWeighted(image, 1, canvas, 0.5, 0)

    gesture_text = "No Gesture Detected"
    if gestures:
        gesture_text = f"Gesture: {gestures[0][0]}"  # display first recognized gesture

    # displays gesture on screen
    cv2.putText(output, gesture_text, (50, 450), cv2.FONT_HERSHEY_SIMPLEX,
                1, (0, 255, 0), 2, cv2.LINE_AA)

    #show everything
    cv2.imshow("Interface Programming", output)

    #close on esc press
    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Figure 19 - Application/Displaying of Logic

5 Task Four

5.1 Introduction

This task was to build upon the work done in task three to use the gestures to interact with drawings on the screen. For this task the user needed a way of drawing multiple shapes on the screen as well as a way of clearing the screen of any clutter they may draw. Also added to this project would be the ability to draw from the tip of the users' index finger as well as changing from an array of colours.

5.2 Project at Work

https://youtu.be/6_wohYPx2Sc

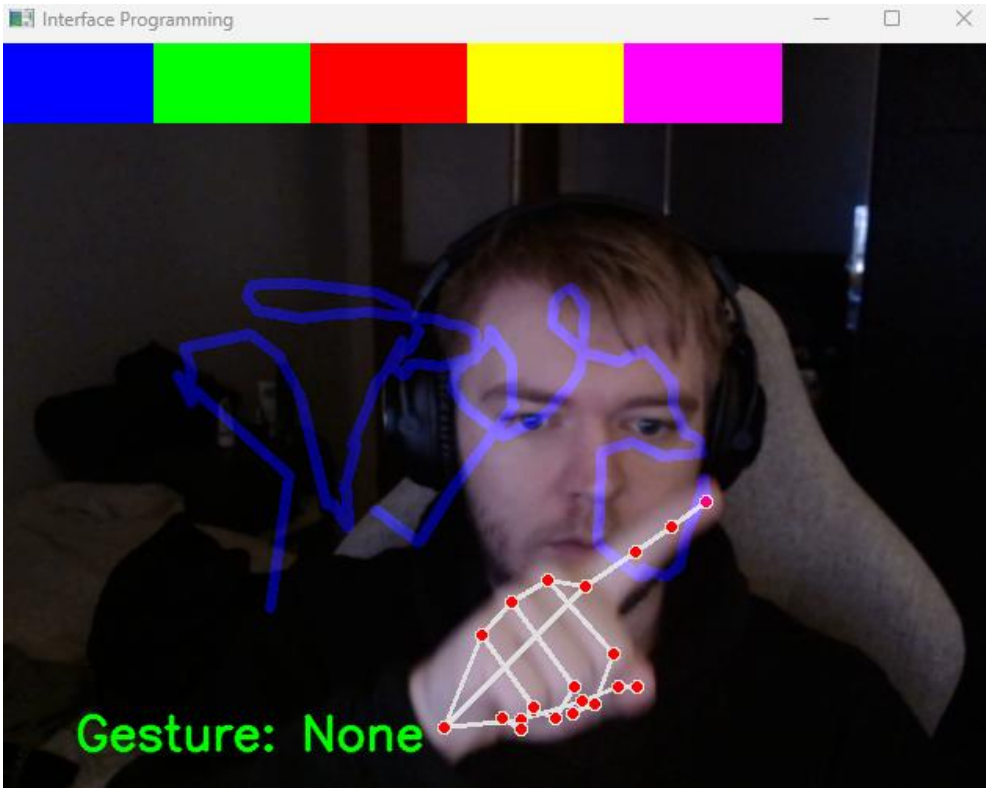

Figure 22 - Drawing on Screen

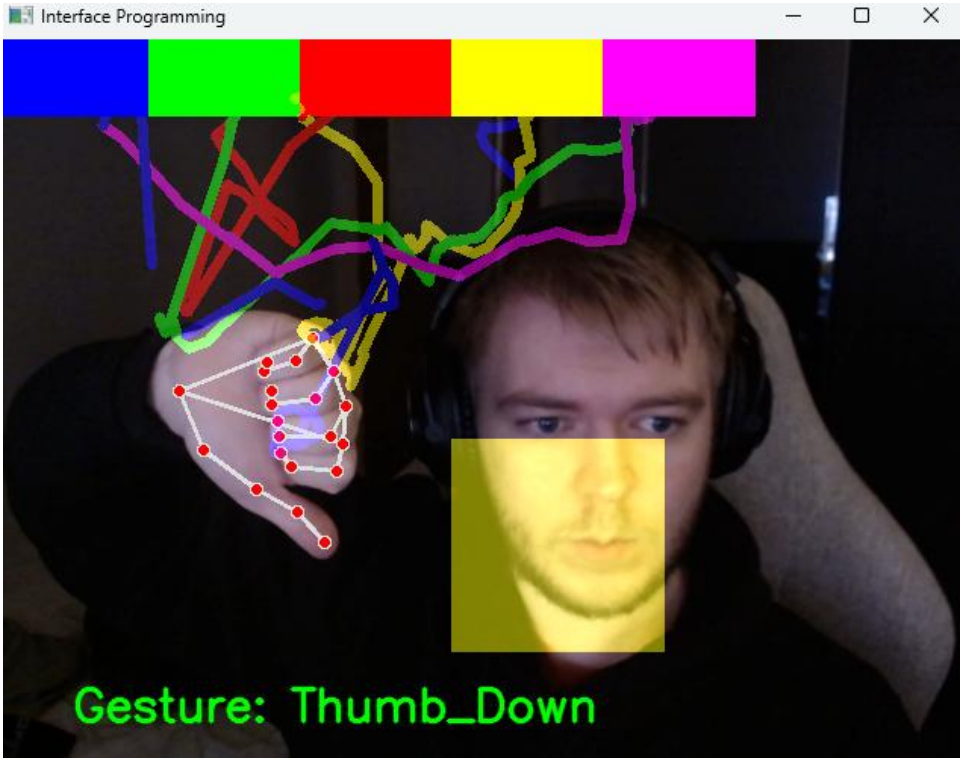

Figure 23 - Drawing Shapes / Changing Colour

5.3 SOURCE CODE

```
#function to check if the index finger is pointing up
def is_pointing_up(hand_landmarks):
    if hand_landmarks:
        index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
        thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
        return index_finger_tip.y < thumb_tip.y  #tip of index finger must be above thumb tip
    return False
```

Figure 24 - Finger Up Logic

```python
#function to check if the thumb is pointing down
#checks if tip of thumb is below base of thumb, finger tip and wrist
def is_thumb_down(hand_landmarks):
    if not hand_landmarks:
        return False

    thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
    index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
    thumb_mcp = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP]
    wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]

    #thumb tip must be below the index finger tip
    thumb_below_index = thumb_tip.y > index_finger_tip.y

    #thumb tip must be below the thumb MCP
    thumb_pointing_down = thumb_tip.y > thumb_mcp.y

    #thumb tip must be below the wrist
    thumb_below_wrist = thumb_tip.y > wrist.y

    return thumb_below_index and thumb_pointing_down and thumb_below_wrist
```

Figure 25 - Thumb Down Logic

```python
# detect hands
hand_results = hand_recognition.detect_hands(rgb_image)
# detect gestures
gestures = gesture_recognition.recognize_gesture(rgb_image)
#detect Face Mesh
face_mesh_results = face_recognition.detect_face_mesh(rgb_image)
#check if Open Palm gesture is detected
clear_screen = any(gesture == "Open_Palm" and score >= 0.5 for gesture, score in gestures)

# reset canvas if Open Palm is detected
if clear_screen:
    canvas[:] = 0
    prev_x, prev_y = None, None
    shape_on_screen = False
```

Figure 27 - Gesture Detection and Canvas Clearing

```python
# drawing functionality
if hand_results.multi_hand_landmarks:
    for hand_landmarks in hand_results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

        index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
        thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
        x, y = int(index_finger_tip.x * 640), int(index_finger_tip.y * 480) #sets co-ordinates of fingertip
        Tx, Ty = int(thumb_tip.x * 640), int(thumb_tip.y * 480) #sets co-ordinates of thumb tip

        tri_pts = np.array([[Tx, Ty], [Tx - 30, Ty + 50], [Tx + 30, Ty + 50]], np.int32) #points for triangle
        tri_pts = tri_pts.reshape((-1, 1, 2))  # reshape to correct format
```

Figure 26 - Drawing Logic Set up

```
#checks if colour needs changing
new_color = check_color_selection(x, y, color_boxes)
if new_color:
    selected_color = new_color

#sets draw to true if finger up
if is_pointing_up(hand_landmarks):
    drawing = True
else:
    drawing = False
    prev_x, prev_y = None, None

#draws if finger up
if drawing and prev_x is not None and prev_y is not None:
    cv2.line(canvas, (prev_x, prev_y), (x, y), selected_color, 5)
prev_x, prev_y = x, y

#checks if thumb is down and there is not a shape on screen
if is_thumb_down(hand_landmarks) and not shape_on_screen:
    draw_shape = True

else:
    draw_shape = False
```

Figure 28 - Drawing Checkers

```
#draws shape and applied cooldown so multiple are not drawn
current_time = time.time()
if draw_shape and (current_time - last_draw_time > draw_cooldown):
    if circle_next:
        cv2.circle(canvas, (Tx, Ty), 100, selected_color, -1)
        shape_on_screen = True
        tri_next = True
        circle_next = False

    elif tri_next:
        tri_pts = np.array([[Tx, Ty], [Tx - 130, Ty + 150], [Tx + 130, Ty + 150]], np.int32)
        tri_pts = tri_pts.reshape((-1, 1, 2))
        cv2.fillPoly(canvas, [tri_pts], selected_color)
        shape_on_screen = True
        rect_next = True
        tri_next = False

    elif rect_next:
        cv2.rectangle(canvas, (Tx, Ty), (Tx + 140, Ty + 140), selected_color, -1)
        shape_on_screen = True
        circle_next = True
        rect_next = False

    last_draw_time = current_time  #adds cooldown
```

Figure 29 - Shape Drawing Logic

6 TASK FIVE

6.1 INTRODUCTION

This task requires an analysis and evaluation of Human-Computer Interaction (HCI) guidelines, with a specific focus on gesture-based interactions. The objective is to research existing interface guidelines, including those from the Kinect for Windows Developer Toolkit, Nielsen's usability heuristics, and mobile touchscreen principles. By comparing and evaluating these guidelines, the task aims to identify best practices for designing intuitive and efficient, simple gesture-based applications. Key considerations include usability, accessibility, environmental factors, and user experience. Based on this analysis, a set of guidelines will be developed to support the creation of applications that rely on gesture recognition for interaction.

6.2 GUIDELINE RESEARCH

Jakob Nielsen was one of the pioneers of the user experience (UX) revolution from the early 2000's. (nngroup, n.d.). As an expert in the UX field, he developed the Ten Usability Heuristics (Nielsen, 1994) which include;

6.2.1 Visibility of System Status

It is imperative that the user gets some form of immediate feedback when interacting with the machine as without some form of visual or auditory cue, the user may not know if their input was successful. A method of application for this is a button changing shade when it is clicked on and reverting after the click is released. Another form of this in action is a loading bar or a pinwheel to signify that the machine is at work.

6.2.2 Match Between the System and the Real World

It should be considered that gestures would be natural and intuitive as to mimic real-world actions where possible. This is so that users, in particular new users, can easily understand how to operate the task as they are already accustomed to what is being asked of them. An example of this in practice is the ability to swipe to scroll, similarly to turning a page in a book.

6.2.3 User Control and Freedom

The ability to undo and redo a gesture is a large factor in what makes a user experience enjoyable, specifically for newer users, as without this option, all actions would have to be re-implemented without error. One of the biggest drawbacks of the type writer was the inability to remove mistakes, which is a large factor for why text editors and the keyboard were so imperative to the computer revolution.

6.2.4 Consistency and Standards

It is important that all applications follow the same rules for gestures as a way of keeping actions easy for the users. This is important to both the user and the developer as users are less likely to switch between products that operate in different ways as they would have to reprogram their brain to operate it. This would essentially lock users out of being able to use certain products meaning the business is being locked out of new clientele at the same time.

6.2.5 Error Prevention

Error handling is a crucial endeavour for all products, most important are the large scale errors that cause massive disruptions, followed by the smaller scale errors that make the user experience worse overall.

6.2.6 Recognition Rather than Recall

A way of making the user experience less overwhelming is by allowing them to recognise certain gestures or actions rather than recalling the location of specific data or memorising a multitude of steps.

6.2.7 Flexibility and Efficiency of Use

Everyone experiences things in a different manner so it is important that the users are given the freedom to customise their experience to fit themselves. This can vastly improve how some people interact with a piece of software/hardware allowing the more experienced to make their use more efficient whilst the newer user can make their experience easier.

6.2.8 Aesthetic and Minimalist Design

A minimalist design makes it easy for new users to learn as clutter just adds more information for the user to indulge. By only adding the necessary elements, confusion is minimised and more time can be spent doing that task that was intended.

6.2.9 Help Users Recognize, Diagnose, and Recover from Errors

By allowing users to diagnose their own issues, it gives those with experience the ability to fix any errors themselves and those without the experience are more able to find a solution, either online or through an engineer. This allows issues to be solved much faster which is a massive benefit to the user.

6.2.10 Help and Documentation

Tooltips are a great way of allowing the manufacturer to help a new user understand what their device is doing and should be doing. An example of this is the mini tutorials that are offered, and skippable, when setting up a new device such as a mobile phone or a TV. By ensuring all documentation is available for the find, it allows the user to quickly identify how they can solve a problem.

6.3 Recommended Guidelines

Nielsens guidelines work well on a broad spectrum, however, the Kinect is vastly different from most other devices when it comes to intractability. The usual methods of input (mouse and keyboard) are not used, instead other methods such as gesture tracking have been used in their place. The guidelines designed for the creation of Kinect based applications are as follows;

6.3.1 Use Natural and Intuitive Gestures

It is important that the user should not be overwhelmed with learning new gestures that would not make sense to them so gesture interaction should be kept as close to mimicking real-world activities as possible.

6.3.2 Provide Immediate Feedback

It is important that the user is given immediate feedback to any gesture they make as it demonstrates the action of their impact. This could be done through an audio or visual method and it could be for both successful and unsuccessful gesture recognition.

6.3.3 Minimise Physical Fatigue

The user should not be forced to continue gestures that will make them fatigued such as holding gestures for long period of time or repeating gestures quickly.

6.3.4 Ensure a Clear Interaction Zone

A well-defined interaction zone ensures that gestures are detected reliably. Users should always be aware of the space they can be detected within and promptly informed when they step outside of the recognition bounds.

6.3.5 Allow Customisation and Calibration

Every user is different so customisation and calibration should be available to adjust gestures to a user's preference. This is even more important with a device such as the Kinect as it requires physical effort for input, so it is imperative that accessibility measures are taken so all types of physical abilities are catered towards.

6.3.6 Prevent Unintended Gestures

In order to avoid unintended gestures from being recognised by the system, the user should be required to make obvious and deliberate movements.

6.3.7 Support Multi-User Interaction

It is important that when it comes to multiple users on screen at one time, that gestures are being tracked correctly. It is vital that the system can differentiate between the different users when they perform gestures simultaneously.

6.3.8 Consider Environment Factors

Environmental factors should not affect how they gesture recognition works, therefore the application should be able to function regardless of the variation in environment, such as lighting and a cluttered background.

6.3.9 Offer Alternative Inputs

Whilst the main input of the Kinect is a visual one, the option for an alternative method of input should always be available. This is particularly important for those with accessibility issues that may need some other method of inputting data such as voice commands or a foot pedal.

6.3.10 Help and Documentation

Tutorials are always a good method of teaching new users how to operate and interact with the system. These demonstrations should teach the user the necessary gestures that make their experience easier to enjoy.

6.4 Evaluation

These guidelines for Kinect-based applications were developed with a focus on intuitive, responsive, and accessible interactions. The design prioritizes natural gestures, immediate feedback, and minimal physical strain to create a more user-friendly experience. Key considerations, such as customization, error prevention, and adaptability to different environments, enhance the system's reliability across diverse users and settings. Nielsen's heuristics played a significant role in shaping these guidelines, particularly in areas like visibility of system status, which influenced the need for real-time feedback, and error prevention, which led to the inclusion of deliberate gesture recognition. Additionally, the emphasis on recognition over recall led to the inclusion of interactive tutorials and clear on-screen prompts. By incorporating these usability principles, the guidelines aim to improve efficiency and accessibility, delivering a seamless experience for users of all skill levels.

7 References

CARROLL, J., 2009. Human Computer Interaction (HCI) [online]. Available from: https://snoopedu.com/app/uploads/2022/03/Reading1_HCI.pdf [Accessed 10/03/2025].

FREIBERGER, P., SWAINE, M., n.d. Calculating Clock [online]. Available from: https://www.britannica.com/technology/Calculating-Clock [Accessed 20/03/2025].

FRIED, G., SWEITZER-LAMME, J., 2014. 1.6 SHICKARD'S CALCULATING CLOCK [online]. Available from: http://ds-wordpress.haverford.edu/bitbybit/bit-by-bit-contents/chapter-one/1-6-shickards-calculating-clock/ [Accessed 20/02/2025].

THE OPEN UNIVERSITY., n.d. 3 Napier's approach to logarithms [online]. Available from: https://www.open.edu/openlearn/science-maths-technology/mathematics-statistics/john-napier/content-section-5 [Accessed 20/03/2025].

FREIBERGER, P., SWAINE, M., n.d. Arithmometer [online]. Available from: https://www.britannica.com/technology/Arithmometer [Accessed 20/03/2025].

IBM., n.d. The IBM punched card [online]. Available from: https://www.ibm.com/history/punched-card [Accessed 20/03/2025].

UNIVERSITY OF PENNSYLVANIA., n.d. Celebrating Penn Engineering History: ENIAC [online]. Available from: https://www.seas.upenn.edu/about/history-heritage/eniac/ [Accessed 20/03/2025].

HP., n.d. Computer History: All About the ENIAC [online]. Available from: https://www.hp.com/ca-en/shop/offer.aspx?p=computer-history-all-about-the-eniac [Accessed 20/03/2025].

CIA., n.d. Enigma Machine [online]. Available from: https://www.hp.com/ca-en/shop/offer.aspx?p=computer-history-all-about-the-eniac [Accessed 21/03/2025].

MASSACHUSETTS INSTITUTE OF TECHNOLOGY., n.d. Vannevar Bush [online]. Available from: https://www.hp.com/ca-en/shop/offer.aspx?p=computer-history-all-about-the-eniac [Accessed 21/03/2025].

BUSH, V., 1945. As We May Think [online]. Available from: https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/ [Accessed 21/03/2025].

IBM., n.d. Time-sharing [online] Available from: https://www.ibm.com/history/time-sharing [Accessed 21/03/2025].

MCCARTHY, J., 1989. Reminiscences on the Theory of Time-Sharing [online] Available from: http://jmc.stanford.edu/computing-science/timesharing.html [Accessed 21/03/2025].

PYFER, J., n.d. Sketchpad [online]. Available from: https://www.britannica.com/technology/Sketchpad [Accessed 21/03/2025].

BLACKWELL, A., RODDEN, K., 2003. Sketchpad: A man-machine graphical communication system [online]. Available from: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf [Accessed 21/03/2025].

SUTHERLAND, I., 1963. Sketchpad: a man-machine graphical communication system. AFIPS '63 (Spring): Proceedings of the May 21-23, 1963, spring joint computer conference pp. 329-346. Available from: https://dl.acm.org/doi/10.1145/1461551.1461591 [Accessed 21/03/2025].

SRI., n.d. The computer mouse and interactive computing [online]. Available from: https://www.sri.com/hoi/computer-mouse-and-interactive-computing/ [Accessed 21/03/2025].

DOUG ENGELBART INSTITUTE., n.d. Historic Firsts: The Mouse [online]. Available from: https://dougengelbart.org/content/view/162/ [Accessed 21/03/2025].

LANDAU, V., 2018., How Douglas Engelbart Invented the Future [online]. Available from: https://www.smithsonianmag.com/innovation/douglas-engelbart-invented-future-180967498/ [Accessed 21/03/2025].

DOUG ENGELBART INSTITUTE., n.d. About NLS / Augment [online]. Available from: https://dougengelbart.org/content/view/155/#1 [Accessed 21/03/2025].

BARNES, S., n.d. ALAN KAY [online]. Available from: https://amturing.acm.org/award_winners/kay_3972189.cfm [Accessed 21/03/2025].

CENTRE FOR COMPUTING HISTORY., n.d. Ted Nelson [online]. Available from: https://www.computinghistory.org.uk/det/1818/ted-nelson/ [Accessed 21/03/2025].

W3C., n.d. What is HyperText [online]. Available from: https://www.w3.org/WhatIs.html [Accessed 22/03/2025].

HISTORYOFINFORMATION., n.d. The IBM 5100, IBM's First "Portable" Computer: $19,975 [online] Available from: https://historyofinformation.com/detail.php?id=924 [Accessed 22/03/2025].

BERG, G., n.d. IBM 5100 Personal Portable Computer [online]. Available from: https://www.obsoletecomputermuseum.org/ibm5100/ [Accessed 22/03/2025].

IBM., n.d. The IBM PC [online]. Available from: https://www.ibm.com/history/personal-computer [Accessed 22/03/2025].

MCCRACKEN, H., 2013. TIME's Machine of the Year, 30 Years Later [online]. Available from: https://techland.time.com/2013/01/04/times-machine-of-the-year-30-years-later/ [Accessed 22/03/2025].

BROCK, D., 2023. 50 YEARS LATE, WE'RE STILL LIVING IN THE XEROX ALTO'S WORLD [online]. Available from: https://spectrum.ieee.org/xerox-alto [Accessed 22/03/2025].

TEACH-ICT., n.d. 12. WIMP or GUI Interface [online]. Available from: https://www.teach-ict.com/as_a2_ict_new/ocr/AS_G061/312_software_hardware/user_interfaces/miniweb/pg12.htm [Accessed 22/03/2025].

INTERACTION DESIGN FOUNDATION., n.d. 4. WIMP [online]. Available from: https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/wimp [Accessed 22/03/2025].

KANADE, V., 2023., What Is WYSIWYG? Meaning, Characteristics, and Functions [online]. Available from: https://www.spiceworks.com/tech/tech-general/articles/wysiwyg-meaning-and-characteristics/ [Accessed 22/03/2025].

KIRVAN, P., n.d. What is the Macintosh (Mac)? [online]. Available from: https://www.techtarget.com/whatis/definition/Macintosh [Accessed 22/03/2025].

O'BRIEN, C., n.d. How Steve Jobs' Macintosh failed and still changed computing [online]. Available from: https://www.latimes.com/business/technology/la-fi-tn-how-the-first-macintosh-failed-and-still-changed-computing-20140123-story.html [Accessed 22/03/2025].

ADOBE ACROBAT TEAM., 2022. Fast-forward – comparing a 1980's supercomputer to the modern smartphone [online]. Available from: https://blog.adobe.com/en/publish/2022/11/08/fast-forward-comparing-1980s-supercomputer-to-modern-smartphone [Accessed 22/03/2025].

NNGROUP., n.d. About NN/g [online]. Available from: https://www.nngroup.com/about/ [Accessed 23/03/2025].

NIELSEN, J., 1994. 10 Usability Heuristics for User Interface Design [online]. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/ [Accessed 23/03/2025].