Name: POEDJIONO Keshia
SID: 57902018

CS3483 Multimodal Interface Design
Assignment

Introduction

The aim of the project is to gain a deeper understanding of hand gestures and screen interaction, which includes viewing images, drawing, and interacting with an image through hand gestures. This project is implemented using two libraries, namely p5.js and ml5.js.

System Design and Program Architecture

The main logic of the program is written in sketch.js file. The p5.js and ml5.js libraries for drawing and detecting hand pose respectively are retrieved through index.html.

1. **Initial setup**

   First, upload any image file into the code directory. In this program, 'bali.jpg' is used as an example. The code for the initial setup is shown below.

   The 'preload()' function runs before the program starts, and it loads the image into memory as 'img'. In our setup, 'capture' stores the webcam video stream, while the 'hands' array will later be used to store detected hand features.

   In the 'setup()' function, a drawing area is created based on the browser window size. The webcam is initialized with the specified width and height. In the 'draw()' function, we display the webcam on the left side and the loaded image on the right side. To ensure both views are aligned and have the same size, their widths are set to half of the canvas.

Name: POEDJIONO Keshia

SID: 57902018

```
let img;
let capture;
let hands = [];

function preload() {
  img = loadImage('bali.jpg');
}

function setup() {
  createCanvas(windowWidth, windowHeight);

  capture = createCapture(VIDEO, {
    audio: false,
    video: { width: 640, height: 480 }
  });
  capture.hide();
}

function draw() {
  background(255);

  // Calculate camera display dimensions
  let cameraAspectRatio = capture.width / capture.height;
  let cameraDisplayHeight = width / 2 / cameraAspectRatio;
  let cameraYOffset = (height - cameraDisplayHeight) / 2;

  // Draw camera on left half
  if (capture.loadedmetadata) {
    image(capture, 0, cameraYOffset, width/2, cameraDisplayHeight);
  }

  // Draw image on right half
  image(img, width/2, cameraYOffset, width/2, cameraDisplayHeight);
}
```

This is the output of the initial setup.

Name: POEDJIONO Keshia

SID: 57902018

2. Index fingertip detection

This section aims to draw a position indicator of the index fingertip. The circle of position indicator will move as the finger move in front of the camera. To perform this hand detection, we must first load the p5.js libraries in 'index.html' as below.

```
<script src="libraries/p5.min.js"></script>
<script src="libraries/p5.sound.min.js"></script>
<script src="https://unpkg.com/ml5@1.2.1/dist/ml5.min.js"></script>
```

Modify the preload() function by adding this line to import the hand detection model.

```
handPose = ml5.handPose();
```

Add line below to the setup() function to start hand detection process. We also have to initialize gotHands() function accordingly.

```
handPose.detectStart(capture, gotHands);
```

```
function gotHands(results) {
  hands = results;
}
```

Add the drawHands() function as shown below to draw a position indicator on the index fingertip. This function displays the camera view along with the index finger point. We also save the x and y coordinates of the fingertip because we want to draw the same indicator at the corresponding position on the image.

```
let fingertipX;
let fingertipY;
```

```
function drawHands(camH, camYoff) {
  if (hands.length === 0) return;

  let hand = hands[0];
  let tip = hand.keypoints[8];

  fingertipX = (tip.x / capture.width) * (width / 2);
  fingertipY = (tip.y / capture.height) * camH + camYoff;

  fill(0, 255, 0);
  circle(fingertipX, fingertipY, 12);


}
```

The function below is used to draw circle on our image according to the x and y fingertip coordinate that we have collected before. The X position should be added by the width/2 as it is the width of each view.

Name: POEDJIONO Keshia

SID: 57902018

```
function drawCircleOnImage(camH, camYoff) {
  if (fingertipX === undefined || fingertipY === undefined) return;

  let translatedX = width / 2 + fingertipX;
  let translatedY = fingertipY;

  fill(255, 0, 0);
  circle(translatedX, translatedY, 12);
}
```

Modify draw() function to call drawHands() and drawCircleOnImage() function.

```
function draw() {
  background(255);

  // Camera and image dimensions
  let camAspect = capture.width / capture.height;
  let camH = (width / 2) / camAspect;
  let camYoff = (height - camH) / 2;

  // Camera view
  if (capture.loadedmetadata) {
    image(capture, 0, camYoff, width / 2, camH);
    drawHands(camH, camYoff);
  }

  // Image view
  image(img, width / 2, camYoff, width / 2, camH);

  drawCircleOnImage(camH, camYoff);


}
```

Below is the result of the index fingerprint detection.

3. Viewing the image

   The goal for this functionality is to display blurred image when the ‘v’ key is pressed. When the user hover over the blurred image, the original image will be displayed inside rectangular area.

   First, we should initialize blur.

Name: POEDJIONO Keshia
SID: 57902018

```
let blurImg;
```

Then, create a blurred version of our image in the setup() function through adding the line below.

```
blurImg = createGraphics(800, 800);
blurImg.image(img, 0, 0, blurImg.width, blurImg.height);
blurImg.filter(BLUR, 5);
```

We should also initialize viewImageMode variable as Boolean to indicate the current activation status.

```
let viewImageMode = false;
```

Add below code to draw() function to trigger the viewImageMode. When user is not in viewImageMode, the function will only display the original image with the position indicator. However, if viewImageMode is true then displayViewMode function is called.

```
if (!viewImageMode) {
  image(img, width / 2, camYoff, width / 2, camH);
  drawCircleOnRight(camH, camYoff);
} else {
  displayViewMode(camH, camYoff);
}
```

The displayViewMode function will first display the blurImg that we have created in the setup function. We need to specify the rectangle size around the finger, which this time is 40 (size of the rectangle can be changed according to the preference). Then we must map the original image coordinates to the area around our index finger point. The purpose is to display original version of the image upon tracing.

Name: POEDJIONO Keshia

SID: 57902018

```
function displayViewMode(camH, camYoff) {
  // Display blurred version
  image(blurImg, width / 2, camYoff, width / 2, camH);

  if (fingertipX === undefined) return;

  // Rectangle area around fingertip
  let rSize = 40;

  let rx = fingertipX;
  let ry = fingertipY;

  // Map display → original image coordinates
  let srcX = (rx / (width/2)) * img.width;
  let srcY = ((ry - camYoff) / camH) * img.height;

  // Draw patch
  image(
    img,
    width/2 + rx - rSize/2, ry - rSize/2, rSize, rSize,
    srcX - (rSize/2)*(img.width/(width/2)),
    srcY - (rSize/2)*(img.height/camH),
    (rSize)*(img.width/(width/2)),
    (rSize)*(img.height/camH)
  );

  noFill();
  stroke(255,0,0);
  strokeWeight(2);
  rect(width/2 + rx - rSize/2, ry - rSize/2, rSize, rSize);

}
```

Initialize keyPressed() function. When ‘v’ key is pressed, the view image mode will be true and triggered through the draw() function.

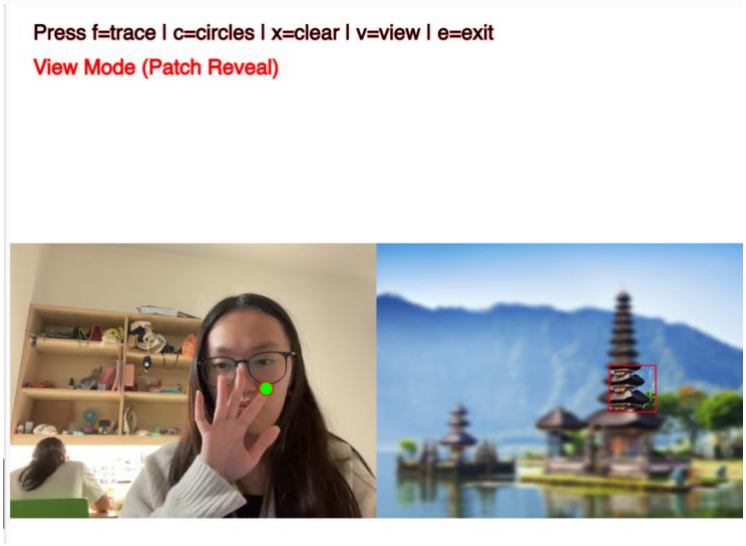
```
function keyPressed() {
  // View Mode
  if (key === 'v') {
    viewImageMode = true;
  }
}
```

Below is the view image demo:

When ‘v’ is pressed:

Name: POEDJIONO Keshia

SID: 57902018



4. Freehand drawing on the image

   The purpose of this function is to draw a continuous freehand line following the movement of the fingertip. First, we should define four variables as below. TracingMode will be our Boolean variable, painting is used to store the canvas while px and py will be the coordinate of the fingerpoint.

```
let tracingMode = false;
```

```
let painting;
let px = 0;
let py = 0;
```

   Then, add below line to the setup to create the canvas for our painting.

```
painting = createGraphics(windowWidth, windowHeight);
painting.clear();
```

   Create a traceLine() function. TraceLine() function is used to draw a continuous freehand line that follows the motion of the user's index fingertip. The function first checks the previous fingerpoint coordinate that we stored. Then, it continuously draws a line between the previous fingertip position and the current fingertip position on the left (camera) side of the screen. It also trace the same line on the image view by shifting the x-coordinates by half the width of the canvas.

```
function traceLine() {
  if (fingertipX === undefined) return;

  painting.stroke(0, 0, 255);
  painting.strokeWeight(8);
  painting.line(px, py, fingertipX, fingertipY);

  painting.line(px + width/2, py,
                fingertipX + width/2, fingertipY);

  px = fingertipX;
  py = fingertipY;
}
```

Name: POEDJIONO Keshia

SID: 57902018

Call the traceLine() function in the draw() function.

```
function draw() {
  background(255);

  // Camera and image dimensions
  let camAspect = capture.width / capture.height;
  let camH = (width / 2) / camAspect;
  let camYoff = (height - camH) / 2;

  // Camera view
  if (capture.loadedmetadata) {
    image(capture, 0, camYoff, width / 2, camH);
    drawHands(camH, camYoff);
  }

  // Image view
  image(img, width / 2, camYoff, width / 2, camH);

  drawCircleOnImage(camH, camYoff);

  if (tracingMode) {
    image(painting, 0, 0);
  }

  drawStatusText();
}
```

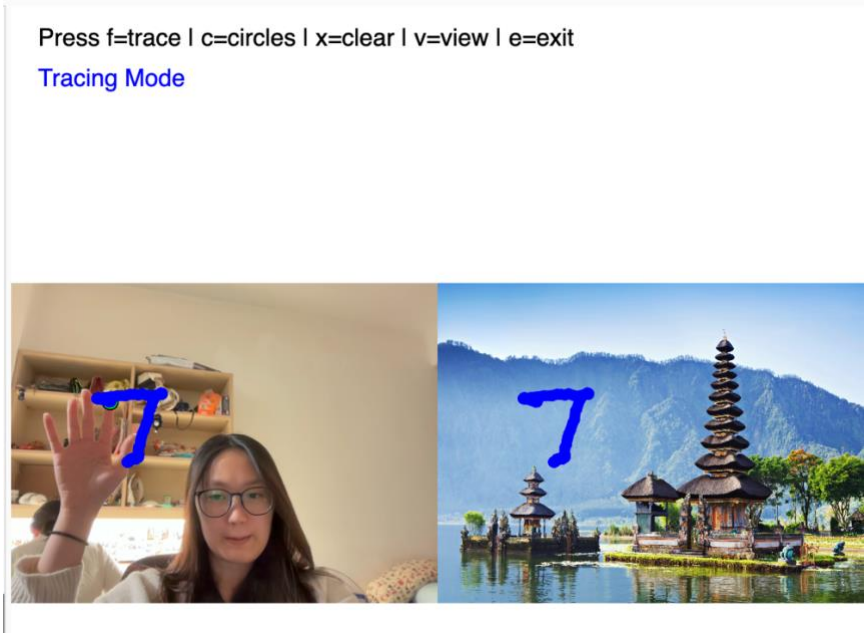
Add the line below so that the trace mode is activated upon ‘f’ key pressed.

```
function keyPressed() {

  // Turn ON/OFF tracing
  if (key === 'f') {
    tracingMode = !tracingMode;
    if (tracingMode) {
      px = fingertipX ?? 0;
      py = fingertipY ?? 0;
    }
  }

  // View Mode
  if (key === 'v') {
    viewImageMode = true;
  }
}
```

Below is the result of activating the traceLine function.

Name: POEDJIONO Keshia

SID: 57902018



5. Drawing circles on the image

   The purpose of this functionality is to draw circles on the image using the same color we are currently hovering over. The radius of the circle depends on the distance between the thumb and index finger.

   We first initialize drawCircleMode variable as Boolean false to indicate the current activation status.

```
let drawCircleMode = false;
```

   Add below code to the draw() function to trigger function call when the drawCircleMode is activated.

```
// Circle mode draw
if (drawCircleMode) drawCircleAtColor(camH, camYoff);
```

   The drawCircleAtColor() is as below. We first should read the position indicator of the thumb and index finger point. After that, we calculate the distance between two position using distThumbIndex function() as we will set the distance as the circle radius. The program finds the fingertip coordinate and converts that point to the corresponding location on the original image. It uses that location to read the pixel color from the original image. Then, we draw with the correct matching color. It will draw a semi-transparent circle at the fingertip position onto a separate drawing layer.

Name: POEDJIONO Keshia

SID: 57902018

```
function drawCircleAtColor(camH, camYoff) {
  if (hands.length === 0 || fingertipX === undefined) return;

  let hand = hands[0];
  let indexTip = hand.keypoints[8];
  let thumbTip = hand.keypoints[4];

  let dist = distThumbIndex(thumbTip, indexTip);
  let r = map(dist, 0, 150, 8, 40);

  let rx = fingertipX;
  let ry = fingertipY;

  let sx = (rx / (width / 2)) * img.width;
  let sy = ((ry - camYoff) / camH) * img.height;
  let col = img.get(int(sx), int(sy));

  painting.noStroke();
  painting.fill(col[0], col[1], col[2], 180);
  painting.circle(rx + width/2, ry, r);
}

function distThumbIndex(t, i) {
  return Math.hypot(t.x - i.x, t.y - i.y);
}
```

```
function distThumbIndex(t, i) {
  return Math.hypot(t.x - i.x, t.y - i.y);
}
```

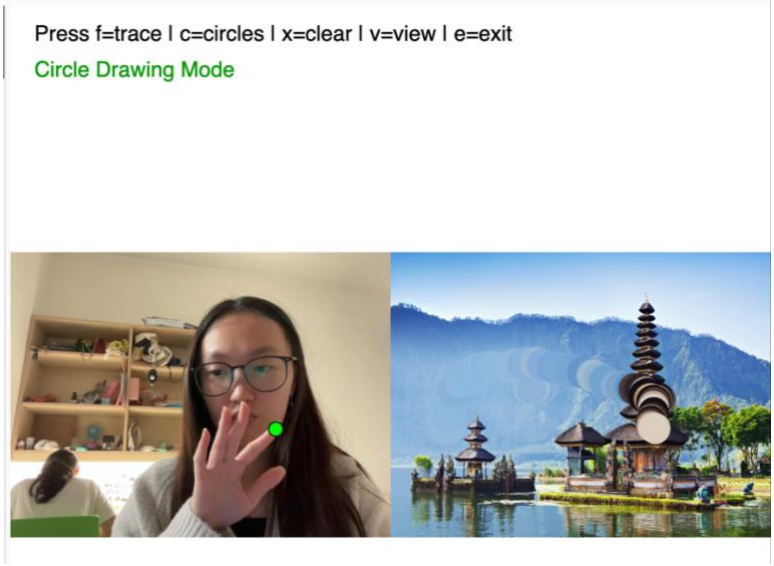
The final keyPressed() function is as below to accommodate all functionality.

```
function keyPressed() {

  // TRACE mode
  if (key === 'f') {
    tracingMode = !tracingMode;
    drawCircleMode = false;
    viewImageMode = false;
  }

  // CIRCLE MODE
  if (key === 'c') {
    drawCircleMode = true;
    tracingMode = false;
    viewImageMode = false;
  }

  // CLEAR ALL DRAWINGS
  if (key === 'x') {
    painting.clear();
  }

  // VIEW MODE
  if (key === 'v') {
    viewImageMode = true;
    tracingMode = false;
    drawCircleMode = false;
  }

  // EXIT
  if (key === 'e') {
    viewImageMode = false;
    tracingMode = false;
    drawCircleMode = false;
  }
}
```

Below is the result:

Name: POEDJIONO Keshia

SID: 57902018

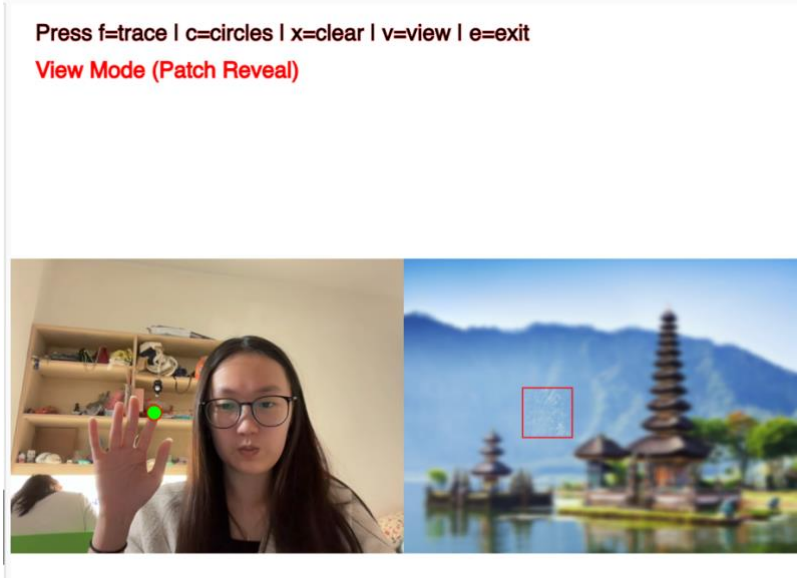


6. Exiting the view image/freehand drawing/circle drawing mode

   When the user clicked ‘e’ key, they should exit from any mode that they are currently in. This is achieved through changing all variables indicating the mode status to false. We include below code in keyPressed() function.

```
// EXIT
if (key === 'e') {
  viewImageMode = false;
  tracingMode = false;
  drawCircleMode = false;
}
```

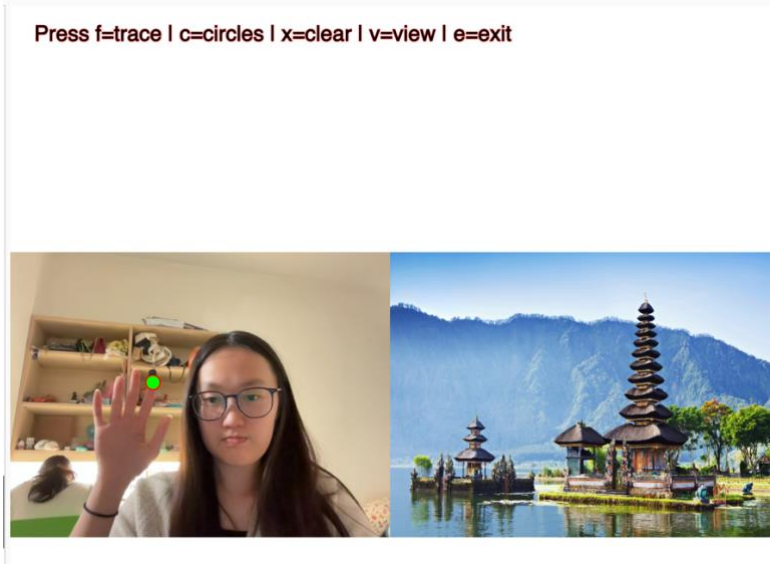
   Below is the result before ‘e’ key pressed:



   Below is after ‘e’ key pressed:

Name: POEDJIONO Keshia

SID: 57902018



Challenges of the Current Implementation

1. **Hard to control hand tracing.**

   The hand detection updates the finger points frequently, making it sensitive to hand tremors. This may cause the drawing produced by the user to appear shaky. This problem can be addressed by applying a motion smoothing technique or a filter.

2. **Sensitivity of finger detection according to lightning conditions**

   To perform an accurate finger detection, the user should position themselves under bright lighting. This is to ensure that the hand can be clearly shown on the screen. The main limitation of the program is that it may not accurately recognize hand and finger points under poor lighting conditions. Therefore, user should adjust their lightning to prevent inaccurate detection.

Possible Improvement of the Program

While the current implementation has fulfilled all the requirements and performs well, there are still some improvements that can be made to enhance the user experience according to human–computer interaction principles.

1. **Provide more details, instructions, and feedback for user actions**

   The current program heavily relies on the user's memory to remember actions associated with each functionality. Giving information about the keyboard keys pressed and the corresponding outputs will help users seamlessly navigate through different functionalities and improve their experience.

Name: POEDJIONO Keshia
SID: 57902018

Providing clearer and more detailed instructions reduces the learning curve for new users while guiding them to accurately perform actions. Instructions are essential to ensure two-way human–computer interaction. They improve learnability and support error prevention. For instance, if the camera does not detect any hand or fingertip, an on-screen instruction such as **"No hand detected, please place your hand in front of the camera"** can be shown as feedback.

Clear and relevant feedback is desirable for every action performed by the user. Feedback works as confirmation that reduces uncertainty and helps the user feel more in control, thereby increasing trust in the interface.

2. **Increase variation in drawing tools**

   More tools could be added to allow more functionality and variation. This includes adding different colour options for users to choose from in the freehand drawing mode. Users may also be able to choose different shapes when tracing.

3. **Save the final image**

   All changes currently made in the program are lost once the user exits. Allowing a saving option would enable users to store the results of their creation for later use, such as documentation, sharing, or further editing.

4. **Support multiple hands**

   Currently, the program only works well when a single hand is detected. It can be extended to support multiple hands to provide a more sophisticated experience, especially when multiple users or gestures are involved.

5. **Implement smoothing filters for fingertip motion**

   The fingertip detection currently reacts to movement very sensitively (almost every millisecond), which allows accurate tracing but may overwhelm users with unstable motions. By providing an option to adjust motion smoothness or applying a filter, we can accommodate a wider range of users and improve stability.

6. **Provide a better interface**

Name: POEDJIONO Keshia
SID: 57902018

The current implementation only runs through a code editor or web editor, making it less accessible for general users. It would be better to provide a more polished interface where users can focus on performing actions rather than running the program.

An improved approach would be to wrap the program into an app that can be accessed easily and provides a more user-friendly layout. As an app, it could support storing previous images, adjusting display settings (such as fonts and colours), and providing clearer information about the program's capabilities and limitations.