CS3483 ASM Report

Design of the program:

```
let originalImage;
let blurredImage;
let bodyPose, bodyDetections = [];
let handPose, video, detections = [];
let handPoseOptions = { maxHands: 1, flipped: false };
let indexFingerTip, thumbTip = null;
let isViewMode, isZoomInMode, isFindMode = false;
let viewRegionSize = 100;

function preload() {...}

function setup() {...}

function gotResults(results) {...}

function gotBodyResults(results) {...}

function draw() {...}

function keyReleased() {...}

function drawHandKeypoints(detections) {...}

function drawIndexFingerTracker() {...}

function drawThumbTracker() {...}

function handleViewMode() {...}

function handleZoomInMode() {...}

function handleFindMode() {...}
```

1. Global variables and setup

2. preload()

```
function preload() {
  originalImage = loadImage("3women-v2.jpg");
  handPose = ml5.handPose(handPoseOptions);
  bodyPose = ml5.bodyPose();
}
```

   - Loads the image (originalImage)
   - Initializes hand pose model (using ml5.handPose)
   - Initializes body pose model (using ml5.bodyPose)
   - Runs before setup() to ensure assets are ready

3. setup()

```
16 function setup() {
17   createCanvas(400, 558);
18   video = createCapture(VIDEO);
19   video.size(width, height/2);
20   video.hide();
21
22   handPose.detectStart(video, gotResults);
23   bodyPose.detectStart(originalImage, gotBodyResults);
24 }
```

- Create canvas
- Set up video capture of webcam
- Start handPose detection on webcam
- Start bodyPose detection on input image

4. gotResults()

```
27 function gotResults(results) {
28   detections = results;
29
30   if (detections.length > 0) {
31     let hand = detections[0];
32     indexFingerTip = hand.keypoints[8];
33     thumbTip = hand.keypoints[4];
34   } else {
35     indexFingerTip = null;
36     thumbTip = null;
37   }
38 }
```

- Handles hand detection results
- Stores detected hand keypoints into detections
- Specifically tracks index finger tip (hand.keypoint[8]) and thumb tip(hand.keypoint[4]) positions
- Stores keypoints of tips in indexFingerTip and thumbTip if hand is detected
- Clears keypoint tracking if no hand is detected

5. gotBodyResults()

```
39 function gotBodyResults(results) {
40   bodyDetections = results;
41 }
```

- Captures information of body pose and store it into bodyDetections

6. draw()

```
43  function draw() {
44    image(video, 0, 0, width, height/2);
45
46    if (isViewMode) {
47      handleViewMode();
48    }
49    else if (isZoomInMode) {
50      handleZoomInMode();
51    }
52    else if (isFindMode) {
53      handleFindMode();
54    }
55    else
56      image(originalImage, 0, height/2, width, height/2);
57
58    if (detections && detections.length > 0) {
59      drawHandKeypoints(detections);
60    }
61
62    drawIndexFingerTracker();
63
64    if(isZoomInMode)
65      drawThumbTracker();
66
67    fill(0, 150);
68    noStroke();
69    textSize(16);
70    textAlign(LEFT, TOP);
71
72    if (isViewMode)
73      text("View Image Mode", 10, height/2 + 10);
74    else if (isZoomInMode)
75      text("Zoom-in Mode", 10, height/2 + 10);
76    else if (isFindMode)
77      text("Find Persons Mode", 10, height/2 + 10);
78  }
```

- Main rendering loop
- Display webcam on screen by image()
- Manages different interaction modes, controlled by isViewMode, isZoomInMode and isFindMode:
    - View Mode
    - Zoom-In Mode
    - Find Persons Mode
- Draw finger trackers, highlighting index finger, and thumb (if in zoom-in mode)
- Indicate current mode on the screen

7. keyReleased()

```
function keyReleased() {
  if (key === 'v' || key === 'V') {
    isViewMode = !isViewMode;
    isZoomInMode = false;
    isFindMode = false;
    blurredImage = null;
  }

  if (key === 'z' || key === 'Z') {
    isZoomInMode = !isZoomInMode;
    isViewMode = false;
    isFindMode = false;
  }

  if (key === 'p' || key === 'P') {
    isFindMode = !isFindMode;
    isViewMode = false;
    isZoomInMode = false;
  }

  if (key === 'e' || key === 'E') {
    isViewMode = false;
    isZoomInMode = false;
    isFindMode = false;
  }
}
```

- The function is triggered when user release a pressed button
- Control the switching of modes by detecting user's key input:
  - 'V': View Mode
  - 'Z': Zoom-In Mode
  - 'P': Find Persons Mode
  - 'E': Exit all modes

8. drawKeypoints()

```
function drawHandKeypoints(detections) {
  noStroke();
  fill(255, 0, 0);
  for (let detection of detections) {
    for (let keypoint of detection.keypoints) {
      if (keypoint.y < height/2)
        circle(keypoint.x, keypoint.y, 10);
    }
  }
}
```

- Help to draw detected hand landmarks on the webcam video

9. drawIndexFingerTracker()

```
118 function drawIndexFingerTracker() {
119   if (indexFingerTip && indexFingerTip.y < height/2) {
120     noStroke();
121     fill(0, 255, 0);
122     circle(indexFingerTip.x, indexFingerTip.y, 15);
123 
124     let imageY = indexFingerTip.y + height/2;
125     fill(0, 255, 0);
126     circle(indexFingerTip.x, imageY, 15);
127   }
128 }
```

- Highlight and draw index finger position as green on both image and video

10. drawThumbTracker()

```
130 function drawThumbTracker() {
131   if (thumbTip && thumbTip.y < height/2) {
132     noStroke();
133     fill(0, 255, 0);
134     circle(thumbTip.x, thumbTip.y, 15);
135 
136     let imageY = thumbTip.y + height/2;
137     fill(0, 255, 0);
138     circle(thumbTip.x, imageY, 15);
139   }
140 }
```

- Highlight and draw thumb position as green in Zoom-in mode on both image and video

11. handleViewMode()

```
function handleViewMode() {
  if (!blurredImage) {
    blurredImage = createGraphics(width, height/2);
    blurredImage.image(originalImage, 0, 0, width, height/2);
    blurredImage.filter(BLUR, 10);
  }

  image(blurredImage, 0, height/2, width, height/2);

  if (indexFingerTip && indexFingerTip.y < height/2) {
    let imageX = indexFingerTip.x;
    let imageY = indexFingerTip.y + height/2;

    let sourceX = constrain(imageX - viewRegionSize/2, 0, width - viewRegionSize);
    let sourceY = constrain(imageY - viewRegionSize/2, height/2, height - viewRegionSize/2);

    noFill();
    strokeWeight(2);
    stroke(255, 255, 0);
    rect(sourceX, sourceY, viewRegionSize, viewRegionSize);

    image(
      originalImage,
      sourceX, sourceY,
      viewRegionSize, viewRegionSize,
      sourceX, sourceY - height/2,
      viewRegionSize, viewRegionSize
    );
  }
}
```

- If blurredImage is null, creates blurred background for displaying
- Calculate the area of the image that needed to be unblurred from the position of mouse cursor
- Display the clear image within the calculated area on the screen to cover the blurred image

12. handleZoomInMode()

```
function handleZoomInMode() {
  image(originalImage, 0, height/2, width, height/2);

  if (thumbTip && indexFingerTip) {
    let minX = min(thumbTip.x, indexFingerTip.x);
    let maxX = max(thumbTip.x, indexFingerTip.x);
    let minY = min(thumbTip.y, indexFingerTip.y) + height/2;
    let maxY = max(thumbTip.y, indexFingerTip.y) + height/2;

    let regionWidth = maxX - minX;
    let regionHeight = maxY - minY;

    minX = constrain(minX, 0, width - regionWidth);
    minY = constrain(minY, height/2, height - regionHeight);

    let zoomedMinX = minX + regionWidth / 4;
    let zoomedMinY = minY + regionHeight / 4;

    let zoomedRegionWidth = regionWidth / 2;
    let zoomedRegionHeight = regionHeight / 2;

    noFill();
    strokeWeight(2);
    stroke(255, 255, 0);
    rect(minX, minY, regionWidth, regionHeight);

    image(
      originalImage,
      minX, minY,
      regionWidth, regionHeight,
      zoomedMinX, zoomedMinY - height/2,
      zoomedRegionWidth, zoomedRegionHeight
    );
  }
}
```

- Display original image on the screen
- Calculate the region between the detected thumb and index finger
- Calculate a smaller region inside the region
- Zoom in the image by replacing the image bounded by region between thumb and index finger by the image bounded by smaller region

13. handleFindMode()

```
function handleFindMode() {
  image(originalImage, 0, height/2, width, height/2);

  if (bodyDetections && bodyDetections.length > 0 && indexFingerTip) {
    let normalizedX = floor((indexFingerTip.x / width) * originalImage.width);
    let normalizedY = floor((indexFingerTip.y / (height/2)) * originalImage.height);
    let selectedPerson = bodyDetections.find(person => {
      const box = person.box;
      return normalizedX >= box.xMin &&
             normalizedX <= box.xMax &&
             normalizedY >= box.yMin &&
             normalizedY <= box.yMax;
    });

    if (selectedPerson) {
      const box = selectedPerson.box;

      let zoomFactor = 1.1;
      let centerX = (box.xMin + box.xMax) / 2;
      let centerY = (box.yMin + box.yMax) / 2;

      let newWidth = box.width * zoomFactor;
      let newHeight = box.height * zoomFactor;

      let newXMin = centerX - newWidth / 2;
      let newYMin = centerY - newHeight / 2;

      let constrainedXMin = constrain(newXMin, 0, originalImage.width - newWidth);
      let constrainedYMin = constrain(newYMin, 0, originalImage.height - newHeight);

      image(
        originalImage,
        constrainedXMin, constrainedYMin + height/2,
        newWidth, newHeight,
        box.xMin, box.yMin,
        box.width, box.height
      );
    }
  }
}
```

- Display original image on the screen
- Detects and zooms into a specific person
- Uses index finger to select person
- Finds person based on body detection
- zooms into the person regions that under the mouse cursor
- Constrains zoom to image boundaries

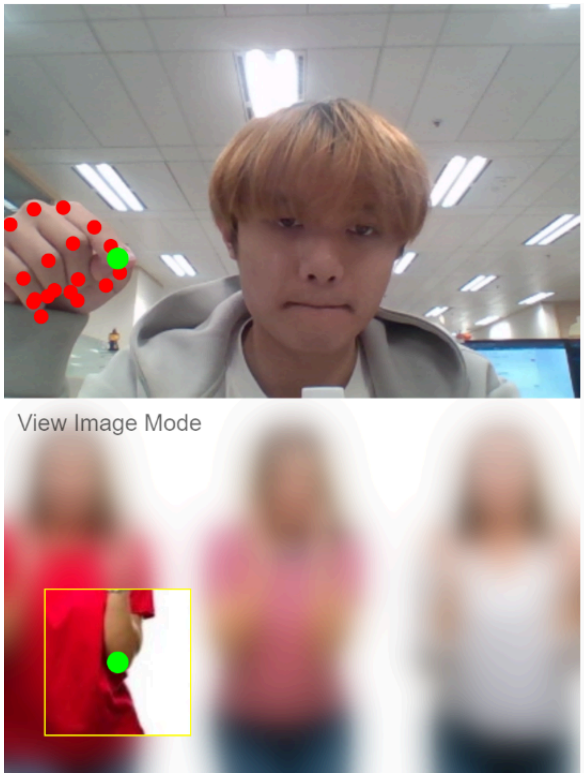
Interaction Flow:

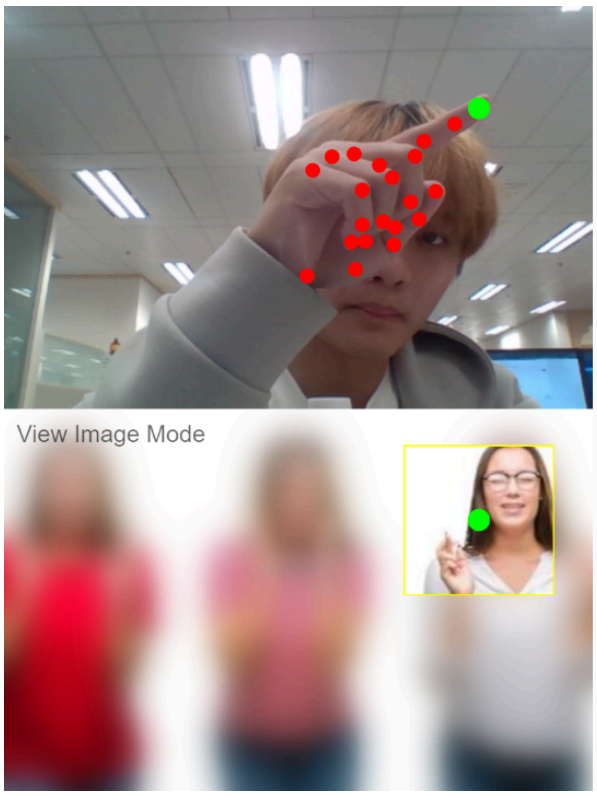
1. Camera captures video feed
2. Hand pose detection tracks hand movements
3. User switches modes using keyboard
4. Interaction adapts based on finger positions
5. Provides dynamic image exploration

Output:

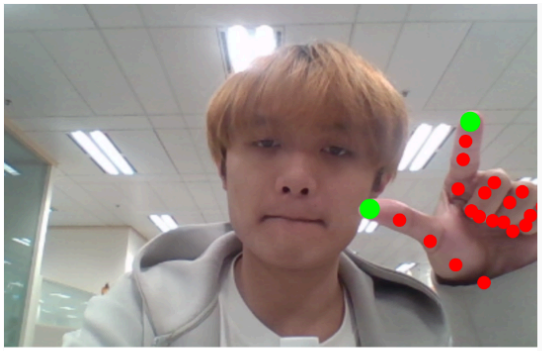
In default mode:

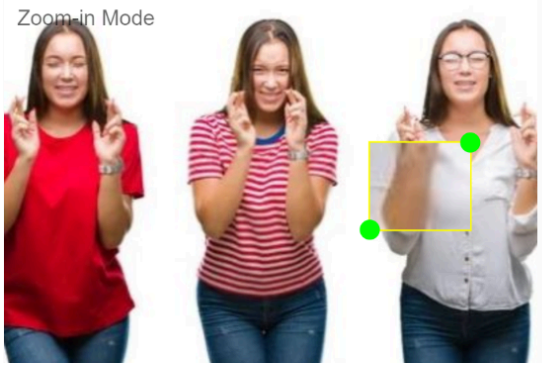
In view image mode:

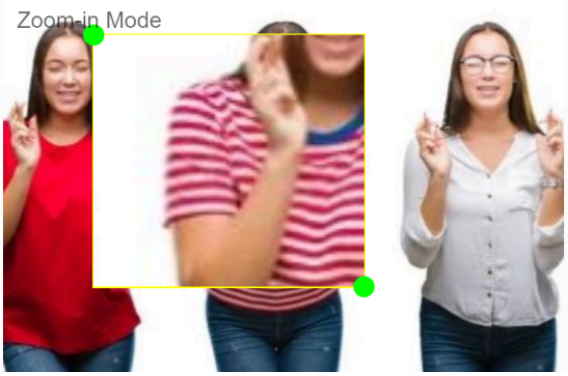




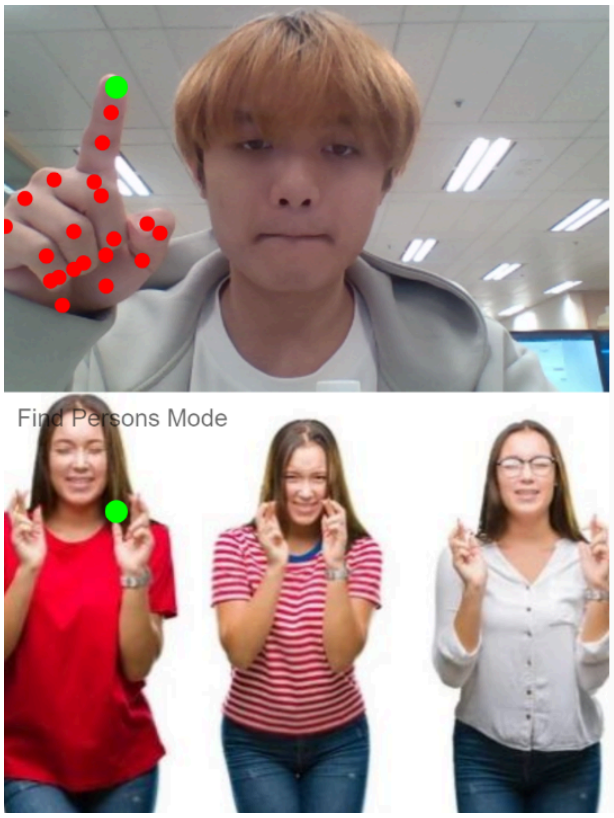
In zoom-in mode:

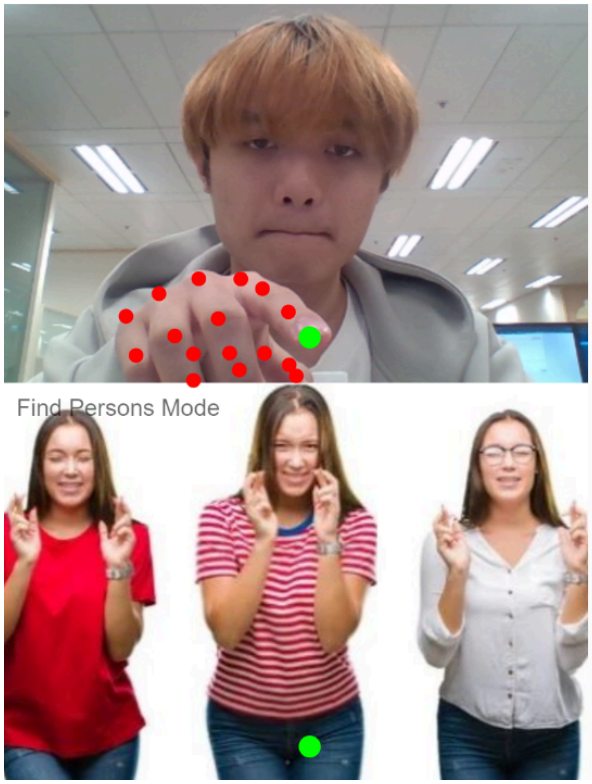






In find persons mode:





Limitations

1. Slow performance due to usage of machine learning
    - The application relies on machine learning models for hand pose and body pose estimation. While these models provide powerful capabilities, they can be computationally intensive, leading to slower performance, especially on less powerful hardware.

2. Frame rate drop with increasing complexity of interaction
    - As the user interacts with the application, performing more complex hand gestures or analyzing more complex scenes, the processing load increases, causing the frame rate to drop.

3. Less hand gesture stability
    - The hand pose estimation model may not be able to consistently and accurately track hand movements, performing less precise interactions.

4. Only support single hand tracking
    - The current implementation of the hand pose estimation is limited to tracking a single hand, resulting to less types of interaction

5. May not detect person successfully with complex backgrounds
    - The body pose model may struggle to accurately detect and isolate people in scenes with complex or cluttered backgrounds, leading to incorrect or incomplete person detection.

Possible Improvement

1. Expand to multi-hand interactions
    - By modifying the "maxHands" option in the handPoseOptions to a higher value (e.g., 2), the application could be extended to support tracking and recognizing gestures from both hands simultaneously. This would greatly expand the range of possible interactions.

2. Add more robust hand gesture recognition
    - to improve the stability and accuracy of hand gesture detection.

3. Using thresholds for person detection to avoid distraction from image background
    - help the person detection model better distinguish people from complex backgrounds, reducing false positives and improving overall performance.

4. Using visual/audio for mode switching
    - Incorporating visual cues (e.g., on-screen indicators) or audio feedback (e.g., button clicks) could provide the user with clear feedback when switching between different modes of operation, enhancing the overall user experience.

5. Implement hand gesture stability filtering
    - Applying techniques like Kalman filtering or other smoothing algorithms to the hand pose data could help stabilize the hand gesture tracking, reducing jitter and improving the responsiveness of the application.

6. Adding error handling
    - Implementing error handling could help the application handle unexpected inputs or scenarios more gracefully, providing the user with clear feedback and maintaining a positive user experience even in the face of errors or failures.