Media Engineering and Technology Faculty
German University in Cairo

Touchless screens using computer vision and ML

Bachelor Thesis

Author: Kareem Mohamed Eid

Supervisors: Dr. Milad Ghantous

Submission Date: 1 June, 2023

Media Engineering and Technology Faculty
German University in Cairo

Touchless screens using computer vision and ML

Bachelor Thesis

Author: Kareem Mohamed Eid

Supervisors: Dr. Milad Ghantous

Submission Date: 1 June, 2023

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

---

Kareem Mohamed Eid
1 June, 2023

Acknowledgments

I would like to express my deepest gratitude and appreciation to my supervisor Dr. Milad who have supported and encouraged me throughout the completion of this bachelor thesis. I would like to thank him for His patience and dedicated time spent in our meetings. Working under his supervision has been a remarkable experience. His insightful feedback, continuous support, and guidance played a pivotal role in shaping the direction and quality of this thesis. I would also like to express my gratitude to my parents, who have supported and encouraged me during challenging times. Their belief in my abilities has been a source of strength throughout this journey.

Abstract

In the past few years, touch screen technologies have made a huge leap. This comes with a lot of hygienic challenges especially after the COVID-19 pandemic. Therefore, Thouchless technology shines because it eliminates the need for physical touch and reduces the risk of viral transmission. It is a promising technology with a fast-growing market. In this research, computer vision and machine learning techniques are used to develop a touchless tickets screen for The Grand Egyptian Museum to provide a new hygienic and engaging way of booking tickets. This was achieved using MediaPipe to detect hand gestures,RobotJS for cursor control and ReactJS to develop a simple and appealing user interface. Evaluation of the touchless tickets screen demonstrated high accuracy, responsiveness, and user satisfaction under several conditions. Users appreciated the hygienic and convenient touchless interactions, enabling them to browse, select, and purchase tickets with ease.

Contents

Chapter 1

Introduction

1.1 Motivation

The emergence of touch screens has led to rapid technological advancements in various fields. However, working with touch screens can pose certain challenges, such as the issue of screen scratches caused by frequent touching. Over time, this can cause the touch screen to become less sensitive or unresponsive to touch inputs, potentially leading to its failure. [10] To address this problem, huge efforts are done in touchless technology development.

Moreover, the COVID-19 pandemic made a lot of hygienic challenges and Touchless technology allows users to interact with devices without physically touching them, which can reduce the spread of germs and bacteria in public spaces so touchless technology has become a necessity not a luxury.

At the start of the pandemic, it was estimated that the touchless sensing market would increase from USD 6.8 billion in 2020 to USD 15.3 billion in 2025, with a Compound Annual Growth Rate (CAGR) of 17.4%. Additionally, the gesture recognition market was predicted to grow from USD 9.8 billion in 2020 to USD 32.3 billion in 2025. Due to COVID-19, there is now an even greater need for touchless technology to prevent similar diseases in the future. Such technology will offer hygienic digital interfaces in the future as well.[5]

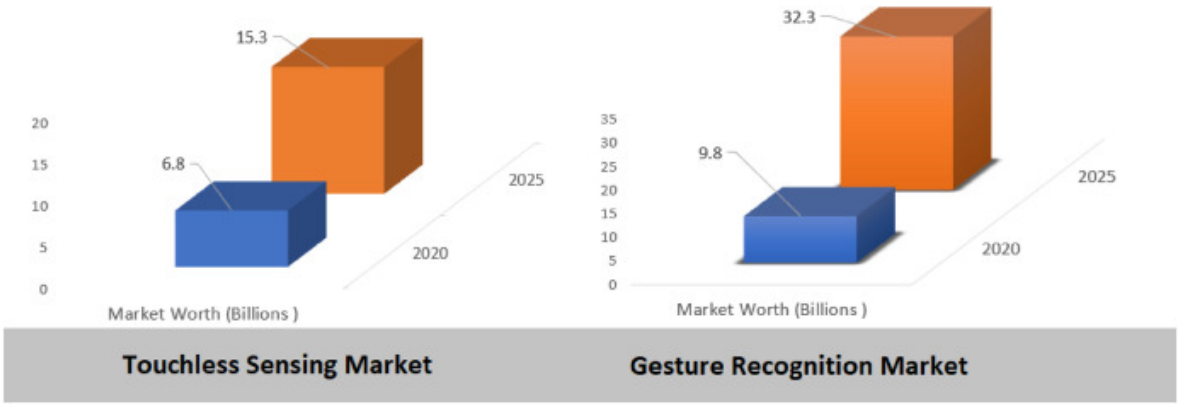


Figure 1.1: Expected rise in the touchless technology market

Although there are many ways of touchless interaction such as motion sensors or voice recognition, The computer vision solution is more convenient as it only requires a simple camera and can be applied easily at low cost anywhere and it's more accurate than voice recognition and less complex than motion sensors.

Despite the potential benefits of touchless technology, there are also challenges that must be addressed. One of the key challenges is designing systems that accurately interpret user gestures and commands, which requires a deep understanding of human behavior and communication. In addition, there are also concerns around privacy and security, particularly when using facial recognition technology.

1.2 Aim

The objective of this thesis is to design and implement a touchless tickets screen for The Grand Egyptian Museum, which aims to offer a more convenient and hygienic way for customers to purchase tickets without having to touch a physical interface. This screen will utilize a simple camera and computer vision techniques to detect users and their hand gestures, as well as to provide an intuitive and seamless user experience. By developing and testing this innovative touchless ticket screen, this project seeks to provide a novel solution to the current challenges of physical touch-based interfaces in public spaces, especially after the COVID-19 pandemic.

Furthermore, this project aims to contribute to the advancement of touchless technology and human-computer interaction by exploring the potential benefits and limitations of touchless interfaces in various domains and applications. Through the development and evaluation of the touchless ticket screen, this thesis aims to provide practical insights and recommendations on how to design effective and engaging touchless interfaces that meet user needs and expectations.

1.3 Outline

The contents of this thesis are organized as follows:

- Chapter 2 discusses background information to correctly understand the topic and related researches.
- Chapter 3 explains the methodology behind this thesis.
- Chapter 4 discusses and analyses the results.
- Chapter 5 concludes the thesis and provides suggested future work.

Chapter 2

Background and related work

2.1 Background

2.1.1 Touchless technology approaches

Touchless technology is a type of technology that allows users to interact with devices without physically touching them.It's getting more and more popular as we interact with it every day in our lives when we walk through an automated door or we pay in supermarkets after scanning the groceries QR codes.

Some of the most popular Touchless technology approaches are:[7]

- Radio frequency identification (RFID)/Near field communication (NFC) which rely on radio-frequency signals to exchange data between devices.
- Gesture recognition: Another common form of touchless technology is gesture recognition. Users can make simple gestures to control or interact with devices without physically touching them.
- Touchless sensing which detects the presence or motion of a person under a sensor such as gesture recognition and automatated doors.
- Voice activation which analyzes sounds and performs tasks based on voice command (e.g., Apple's Siri, Google's Home, or Amazon's Alexa).
- Bluetooth signals with an encrypted identifier which is picked up by a reader, connecting a mobile app with an existing sensor and system.
- QR codes/barcodes: On-demand mobile Quick response (QR) codes provide instant access to information. A QR code is a version of a barcode, a technology that is easy for mobile apps to scan. QR codes can direct users to download a mobile app or visit a website.

- Biometric authentication: Biometrics measure the physical characteristics (e.g., face or palm) of a person to verify identity.

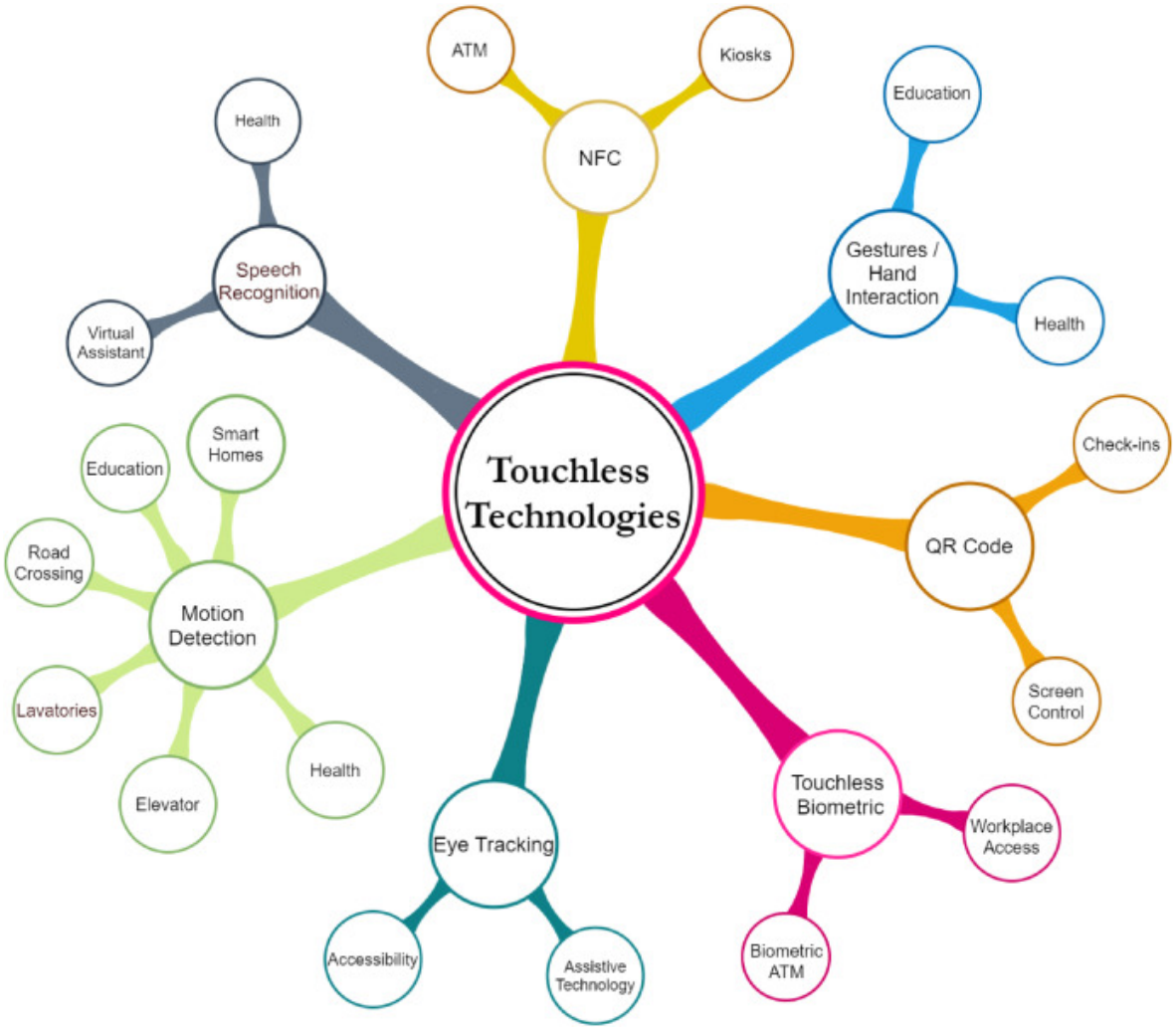


Figure 2.1: Touchless technology approaches

2.1.2 Gesture recognition using computer vision and machine learning

The computer vision and machine learning solution has made touchless screen technology more accessible and user-friendly. This approach utilizes cameras and sensors to detect human gestures and movements, allowing users to interact with the screen without physical contact. This technology has become increasingly popular due to its simplicity, accuracy, and ease of use.

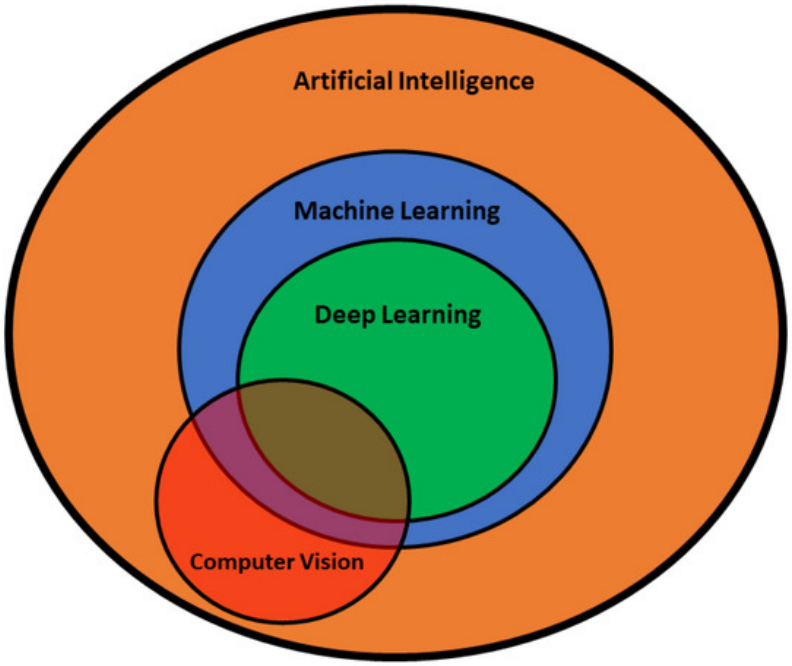


Figure 2.2: The relation between machine learning and computer vision

2.1.2.1 Machine learning

In the past, computers needed specific instructions or algorithms to carry out tasks. They were unable to learn from past experiences and improve on past mistakes. Programming a computer required a complete and accurate algorithm, which is a time-consuming task that requires specialized training. Machine learning comes to play here as it aims to change this by teaching computers to learn from experience and improve their performance, potentially easing the burden of programming and handling complex information.[2]

Machine learning is a crucial part of artificial intelligence, which allows a system to learn and adapt to changes in a dynamic environment. This ability enables the system to find solutions to various problems, such as vision, speech recognition, and robotics. For example, recognizing faces is a complex task for computers that humans do effortlessly but cannot explain in detail. However, by analyzing sample face images, a learning program can capture patterns specific to a person and recognize them by checking for those patterns in a given image. Machine learning involves programming computers to optimize a performance criterion using past experiences or example data. The process involves building mathematical models based on the theory of statistics and using algorithms to solve the optimized problem, as well as store and process massive amounts of data efficiently. Once a model is learned, its representation and algorithmic solution for inference need to be efficient as well. The efficiency of the learning or inference algorithm may be as crucial as its predictive accuracy, especially in certain applications.

2.1.2.2 Deep learning

Deep learning is a part of machine learning that involves using computational models with multiple processing layers to learn and represent data with varying levels of abstraction, in a similar way to how the brain perceives and understands complex information. This approach enables the capturing of intricate structures in large-scale data. The deep learning methodology encompasses various techniques, such as neural networks and hierarchical probabilistic models. Deep learning has gained widespread interest due to its demonstrated superiority over previous state-of-the-art techniques in various tasks and the increasing availability of complex data from diverse sources, including visual, audio, medical, social, and sensor data.[1]

Deep learning has had a significant impact on computer vision, with improved performance in various tasks such as object detection, motion tracking, action recognition, human pose estimation, and semantic segmentation. There are alot of deep learning models which are used in solving computer vision problems such as Convolutional Neural Networks (CNNs), the "Boltzmann family" including Deep Belief Networks (DBNs) and Deep Boltzmann Machines (DBMs), and Stacked (Denoising) Autoencoders.

2.1.2.3 Computer vision

Computer Vision involves using AI algorithms to extract information from images or videos. While image-processing algorithms work directly on the pixels with predetermined rules, natural images are often too complex to be effectively processed in this manner, especially with the large size of modern high-resolution images[8]. To solve this problem, Computer Vision uses Machine learning (ML) algorithms that can learn and solve tasks without being explicitly programmed by a human developer. This is done by combining image processing algorithms with ML models to create powerful applications.

Computer vision algorithms follow a generic framework with multiple steps:

- preprocessing the images to make the following tasks easier, such as normalization, background removal, denoising, and feature extraction.
- performing The main task of the application, which can be image classification, object detection, or segmentation. This step produces a raw output.
- Post-processing of the output to correct and refine the raw output and make it interpretable.

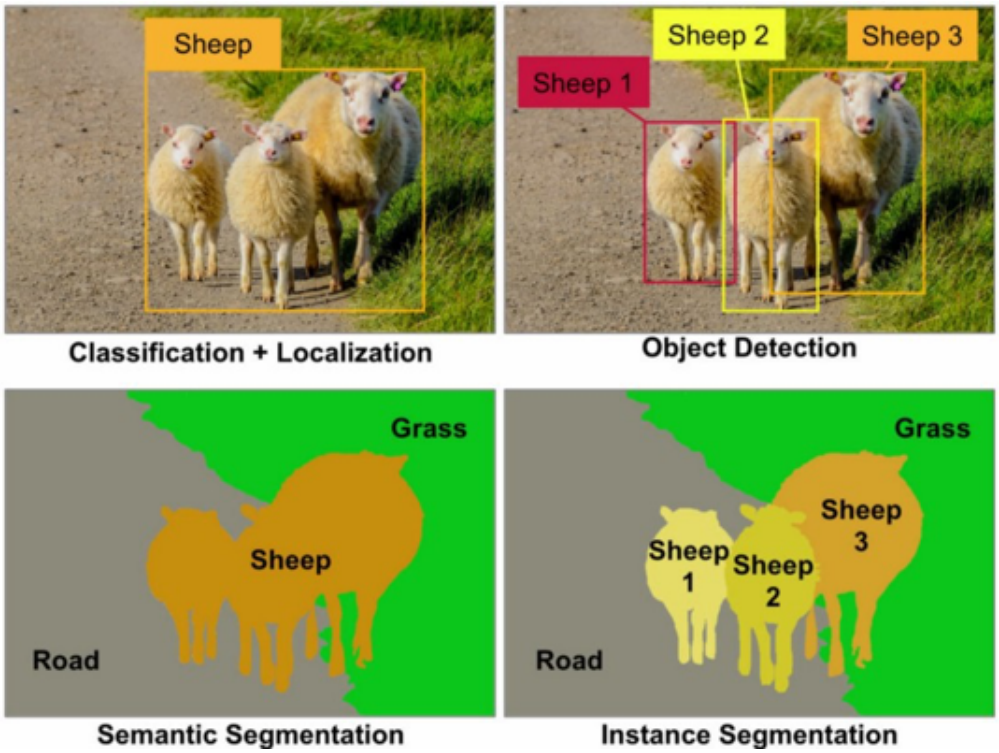


Figure 2.3: Computer vision algorithms

2.1.2.4 Google MediaPipe framework

Google MediaPipe[6], which will be used in this paper for hand detection, is an open-source framework developed by Google that provides a set of customizable building blocks for building real-time, cross-platform computer vision applications. It offers a range of pre-built modules that can be used to perform tasks such as face detection, hand tracking, pose estimation, object detection and tracking, and more.

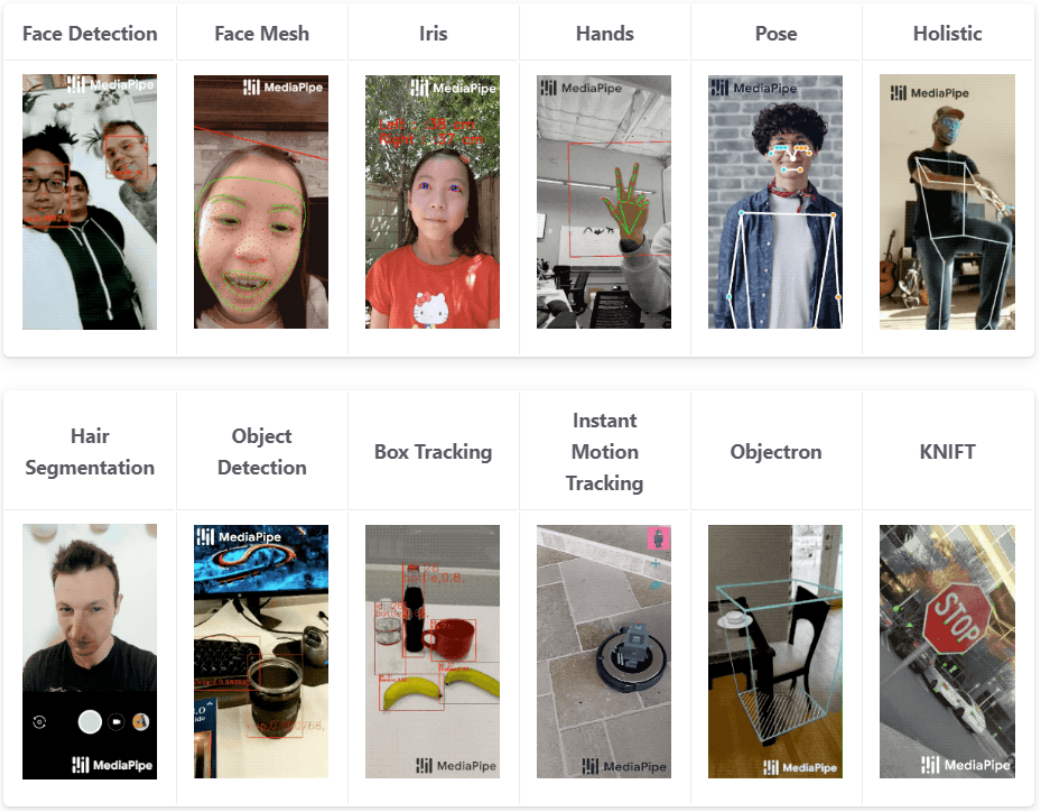


Figure 2.4: MediaPipe solutions

MediaPipe offers a wide range of features and benefits for developers, including its ease of use and ability to run on a variety of devices, including smartphones, desktop computers, and specialized hardware such as edge devices and Internet of things (IoT) devices. Its flexible architecture allows developers to create custom pipelines for their specific use cases, and it supports a variety of programming languages including Python and C++.

For hand detection, MediaPipe uses an ML pipeline consisting of several models:[3]

- A palm detector model (called BlazePalm) that operates on the full image and returns an oriented hand bounding box.
- A hand landmark model that operates on the cropped image region defined by the palm detector and returns high fidelity 3D hand keypoints.
- A gesture recognizer that classifies the previously computed keypoint configuration into a discrete set of gestures.

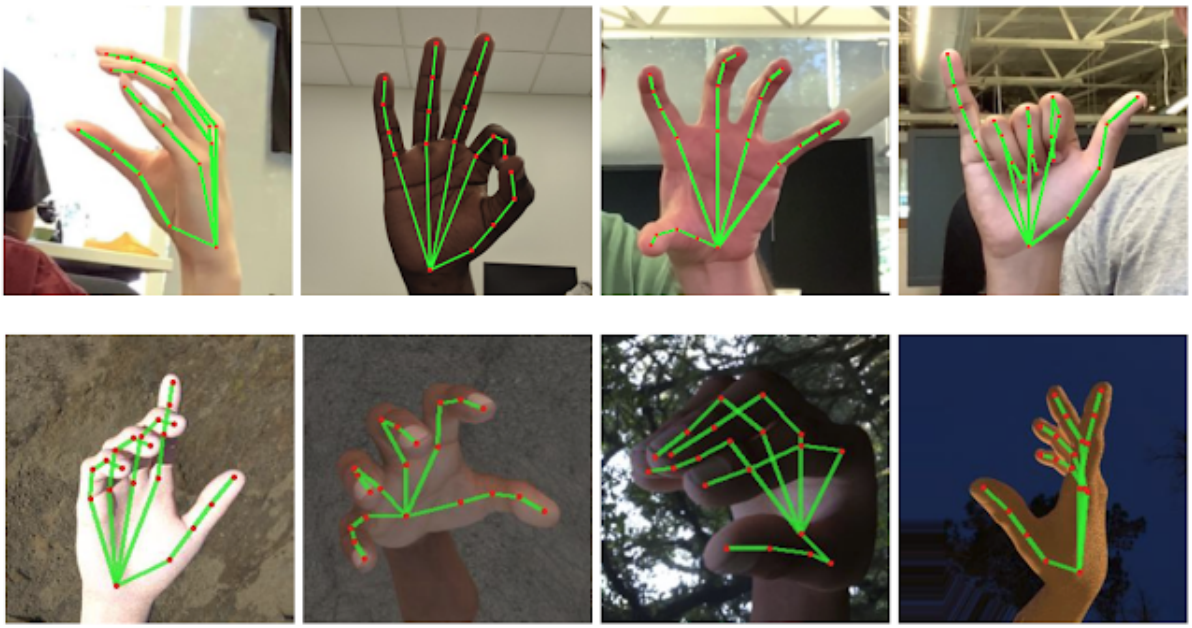


Figure 2.5: MediaPipe hand detection

2.2 Related work

2.2.1 Touchless interaction in Surgery

There are a lot visual displays for accessing pre- and intra-operative images in any operating room, including computer tomography (CT), magnetic resonance imagery (MRI), and fluoroscopy.They aid in planning and diagnosis and give the surgeon a virtual "line of sight" into the patient's anatomy[9]. Despite the fact that surgeons depend on the capture, browsing, and manipulation of these pictures, they are limited by conventional interaction methods (like a keyboard and mouse).The requirement to uphold a clear distinction between what is sterile and what is not lies at the core of the limitations. Surgeons cannot handle these input devices while they are gloved and scrubbed so they request help from other members of the surgical team who are not always available, producing frustration and delay.Touchless displays come to help in these situations as they make surgeons in full control of these displays without the fear of breaking the sterile field which reduces the risk of infection and improve efficiency. There are also challenges of implementing touchless technology in surgery, such as accuracy, reliability, and ease of use. Overall, touchless interaction has the potential to transform surgical procedures and improve patient outcomes, but further research and development are needed to address the current limitations of touchless technology in the surgical context.

Figure 2.6: Touchless screen interaction in surgery

2.2.2 BMW gesture control

BMW Gesture Control[12] is a feature that allows drivers to control certain iDrive functions with hand gestures captured by a 3D camera. The system enables the driver to make simple gestures to accept or decline phone calls, adjust audio volume, set navigation destinations, and change the angle in the 360-degree view of the vehicle. The purpose of this technology is to help drivers interact with the iDrive system safely while remaining focused on driving. The article highlights the convenience of the system, especially in situations where the driver may need to answer an important call without taking their eyes off the road.

Figure 2.7: BMW gesture control

2.2.3 VLC media player hand gesture control

hand gesture recognition (HGR) system was developed for controlling the VLC media player using deep learning based two stream transfer learning[11]. The system uses a webcam to acquire images and recognizes hand gestures as commands for the VLC media player functions such as play, pause, and stop. The system employs two pre-trained models, MobileNet and Inception V3, and uses YOLO architecture to find the region of interest in the image. The MobileNet based transfer learning architecture is trained separately on edge map images and spatial images, and joint probabilities are combined to determine the hand sign of the image. The system achieves high accuracy of 100% in real-time experimentation.

2.2.4 AIRxTOUCH touchless kiosk

iNUI Studio SA, in collaboration with Samsung and Microsoft, launched its touchless kiosk called Air Touch, which is designed to replace public touchscreens and safeguard against viruses, bacteria, and germs. According to a press release, this kiosk allows users to interact directly with objects on the screen without moving a cursor at a distance like camera-based technologies. It detects the user's finger at a distance of 14 centimeters (5.5 inches) from the screen, reducing the risk of accidental touching of the screen that is often seen in solutions based on capacitive foils. This technology works well in various light conditions, including bright sunlight up to 120,000 lux.

Figure 2.8: AIRxTOUCH kiosk

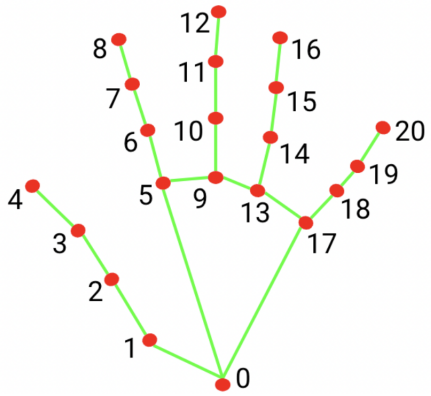
Chapter 3

Methodology

3.1 Hand recognition using MediaPipe

The MediaPipe Hand Landmarker task[3] enables the identification of hand landmarks within an image. This task allows for the localization of key points on the hands, enabling the application of visual effects over the detected hand regions. It processes image data, either as static data or in a continuous stream, using a machine learning model. The output of the task includes the coordinates of hand landmarks in both image and world coordinates, as well as the determination of handedness (left or right hand) for multiple hands detected in the image.

The hand landmark model that is used by MediaPipe is designed to identify and locate the positions of 21 key points or knuckle coordinates within the detected hand regions. This model has been trained using a dataset consisting of around 30,000 real-world images, as well as synthetic hand models placed in different backgrounds to enhance its accuracy and robustness.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 3.1: MediaPipe hand landmarks

To integrate MediaPipe with the camera the following code is used. It also specifies the minimum hand detection confidence and tracking confidence to be 0.5 in order to avoid noise and in the same time detect hands that are not so obvious or in a strange position and the maximum number of hands is set to 1 which allows only the first hand detected to be in control. Then it calls the onResults function which uses the hand landmarks to detect gestures.

```
useEffect(() => {
  const hands = new Hands({
    locateFile: (file) => {
      return `https://cdn.jsdelivr.net/npm/@mediapipe/hands/${file}`;
    }
  });
  hands.setOptions({
    maxNumHands: 1,
    modelComplexity: 1,
    minDetectionConfidence: 0.5,
    minTrackingConfidence: 0.5,
    selfieMode: 1
  });
  hands.onResults(onResults);
  if (typeof webcamRef.current !== "undefined" && webcamRef.current !== null) {
    camera = new cam.Camera(webcamRef.current.video, {
      onFrame: async () => {
        await hands.send({ image: webcamRef.current.video });
      },
      width: 640,
      height: 480,
    });
    camera.start();
  }
}, []);
```

Figure 3.2: MediaPipe integration with camera code

3.2 Gesture detection

The implemented gestures are as follows:

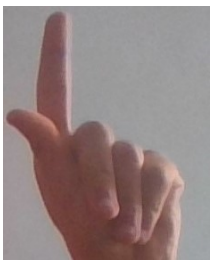
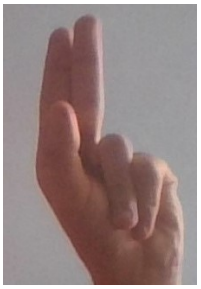
Functionality	Hand Gesture	Brief description
Mouse movement		Extend the index finger upwards and extend other fingers (excluding the thumb) downwards. The cursor moves with the position of the index finger.
Mouse clicking		Extend the index finger and the middle finger upwards and extend other fingers (excluding the thumb) downwards. This makes a clicking action in the current cursor position.
Vertical scrolling	 	Make a horizontal line with the tips of the fingers (excluding the thumb) then move them together downwards to scroll down or upwards to scroll up.
Horizontal scrolling	 	Make a vertical line with the tips of the fingers (excluding the thumb) then move them together to the right to make right scrolling or to the left to make left scrolling.

Table 3.1: Implemented hand gestures

3.2.1 Mouse movement and clicking

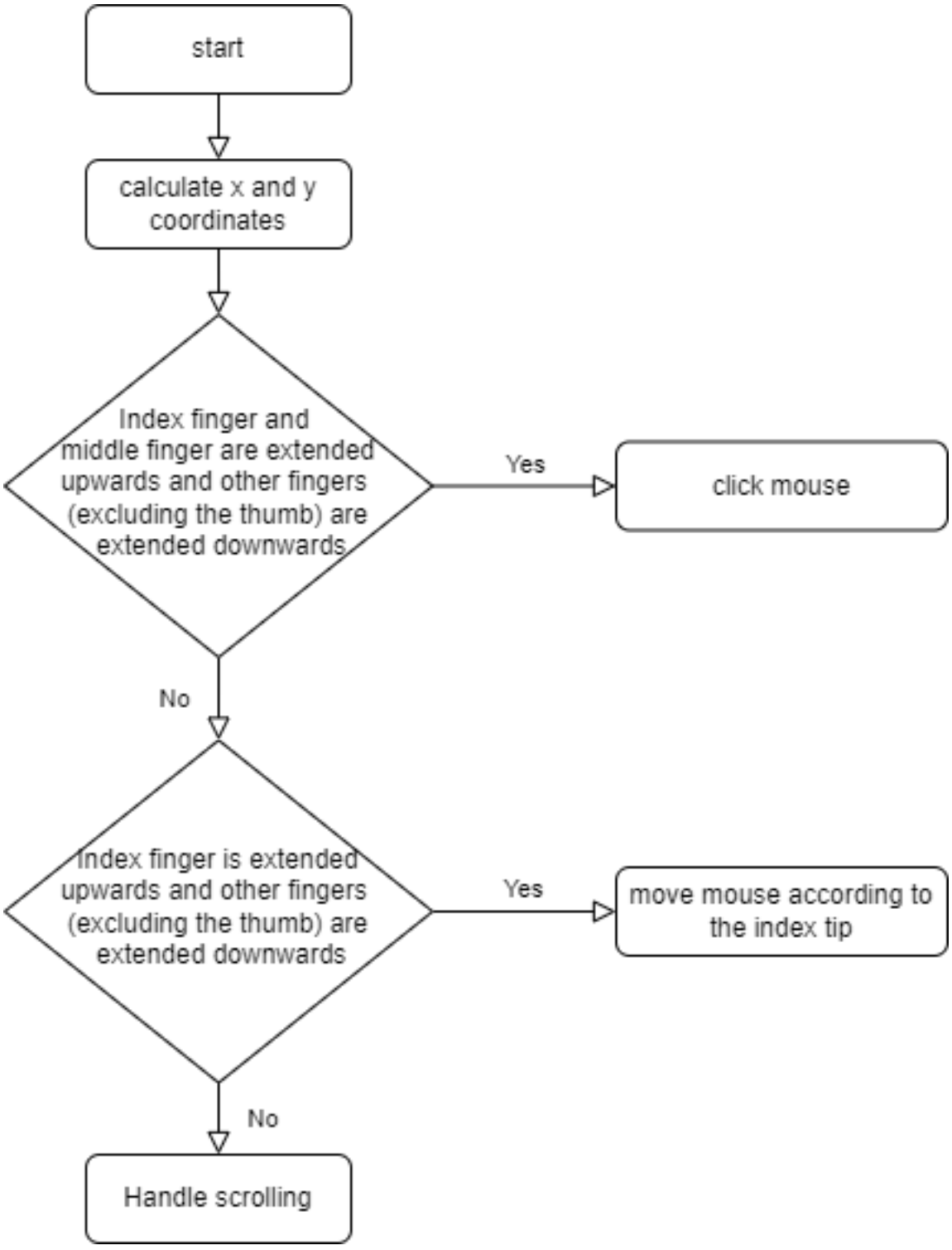


Figure 3.3: Flowchart of gestures detection

In the onResults function called in the previous useEffect function, the position of index finger tip is translated screen coordinates in pixels and the y coordinate of it is scaled by 1.25 to cover the bottom side of the screen. Then, two checks for mouse clicking and movement and scrolling function are called.

```
function onResults(results) {

  if (results.multiHandLandmarks) {
    if (results.multiHandLandmarks[0] && results.multiHandedness) {
      hand = results.multiHandLandmarks[0];
      const pointer = hand[8];
      // 100% zoom screen size = 1536 * 864
      const x = pointer.x * 1536;
      const y = Math.min(864, pointer.y * 864 * 1.25);
      if (extendedUp(5, 8) && extendedUp(9, 12) && extendedDown(13, 16)
        && extendedDown(17, 20) && distance(hand[8], hand[12]) < 0.09) {
        clickMouse()
      } else if (extendedUp(5, 8) && extendedDown(9, 12)
        && extendedDown(13, 16) && extendedDown(17, 20)) {
        moveMouse(x, y)
      } else {
        handleUpDownScroll()
        handleLeftRightScroll()
      }
    }
  }
}
```

Figure 3.4: onResults function code

The condition for moving the mouse checks if the index finger is extended upwards and other fingers (excluding the thumb) are extended downwards. Then, it calls the moveMouse function to move the cursor according to the position of index finger tip which was converted to pixels earlier.

The condition for clicking mouse checks if the index finger and the middle finger are extended upwards and other fingers (excluding the thumb) are extended downwards and the index finger tip and middle finger tip are near each other. Then, it calls the clickMouse function to click in the current position.

```
function extendedUp(bottom, top) {
  return hand[bottom].y > hand[top].y
}

function extendedDown(bottom, top) {
  return hand[bottom].y < hand[top].y
}
```

Figure 3.5: extendendUp and extendedDown functions code

3.2.2 Vertical Scrolling

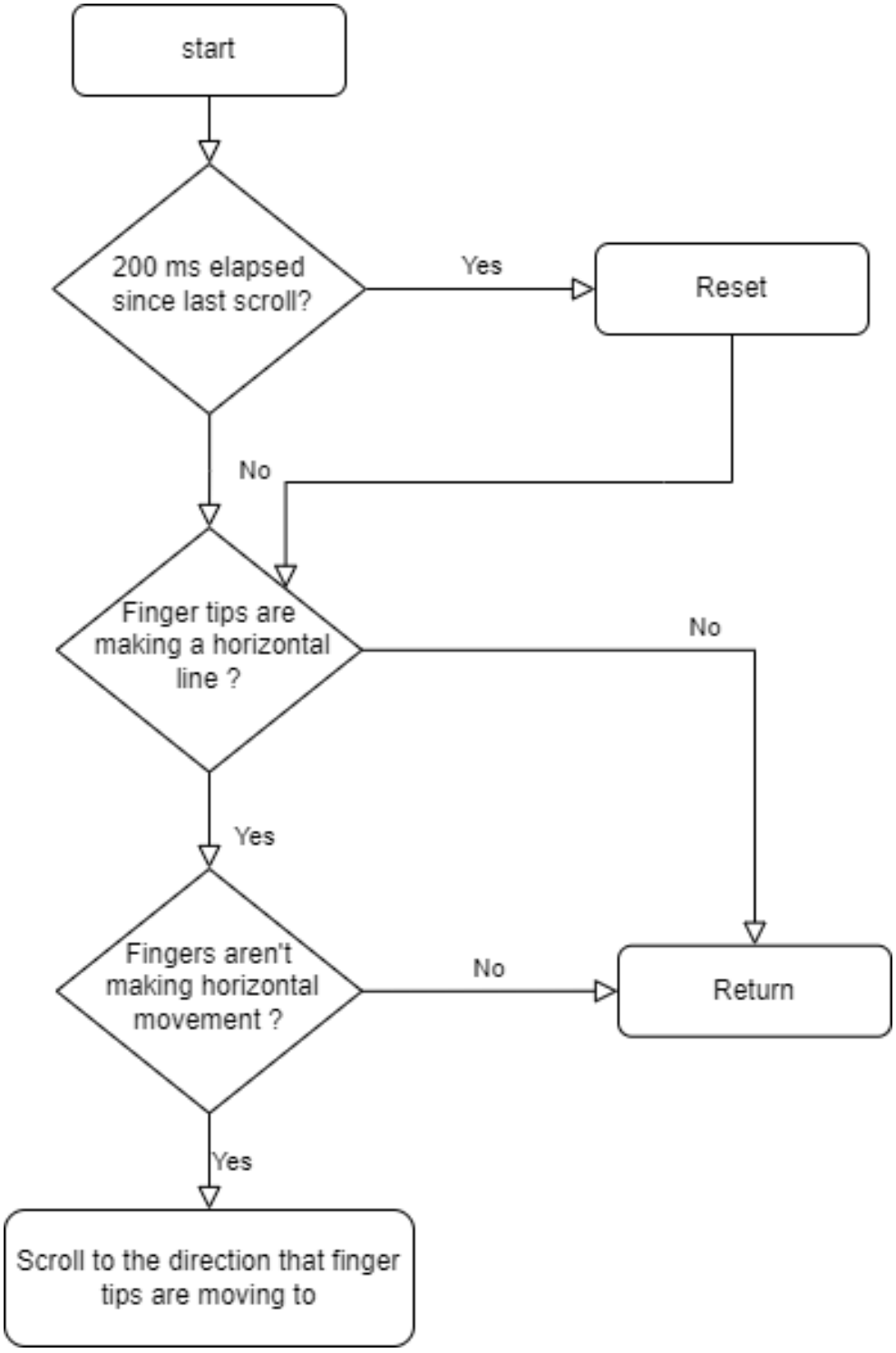


Figure 3.6: Flowchart of up and down scrolling

Then, the function for checking up and down scrolling is called. This function works as follows: Firstly, it checks if the tips of the fingers (excluding the thumb) are making a horizontal line and saves the hand in a variable called lastHorizontal. Then, in each call afterwards it checks if the finger tips are in the same horizontal level as the tips in the lastHorizontal variable. Then, if the tips move down it scrolls down and vice versa calling the navigate function with the amount and the direction which is vertical. All of the previous checks are done with an error margin of 0.1 of the canvas. The scrolling happens for 200 ms then the lastHorizontal is reset in order to stop the scrolling.

```
function handleUpDownScroll() {
  if (lastHorizontal && Date.now() - lastHorizontalTime > 200) {
    lastHorizontal = null
    lastHorizontalTime = null
  }
  if (lastHorizontal == null) {
    if (isHorizontal()) {
      lastHorizontal = hand
      lastHorizontalTime = Date.now()
    }
  } else {
    if (Math.abs(hand[8].x - lastHorizontal[8].x) < 0.1 && Math.abs(hand[12].x - lastHorizontal[12].x) < 0.1
      && Math.abs(hand[16].x - lastHorizontal[16].x) < 0.1 && Math.abs(hand[20].x - lastHorizontal[20].x) < 0.1) {
      // positive is down & negative is up
      if (hand[8].y - lastHorizontal[8].y > 0.1 && hand[12].y - lastHorizontal[12].y > 0.1
        && hand[16].y - lastHorizontal[16].y > 0.1 && hand[20].y - lastHorizontal[20].y > 0.1)
        navigate(1, 'vertical')
      if (lastHorizontal[8].y - hand[8].y > 0.1 && lastHorizontal[12].y - hand[12].y > 0.1
        && lastHorizontal[16].y - hand[16].y > 0.1 && lastHorizontal[20].y - hand[20].y > 0.1)
        navigate(-1, 'vertical')
    }
  }
}
```

Figure 3.7: handleUpDownScroll function code

The isHorizontal function called in the handleUpDownScroll function simply checks if the four finger tips are making a horizontal line with the difference in the y position between any 2 tips is at most 0.1 of the canvas.

```
function isHorizontal() {
  let min = 1, max = 0
  for (let i = 8; i < hand.length; i += 4) {
    min = Math.min(min, hand[i].y)
    max = Math.max(max, hand[i].y)
  }
  return max - min <= 0.1
}
```

Figure 3.8: isHorizontal function code

3.2.3 Horizontal Scrolling

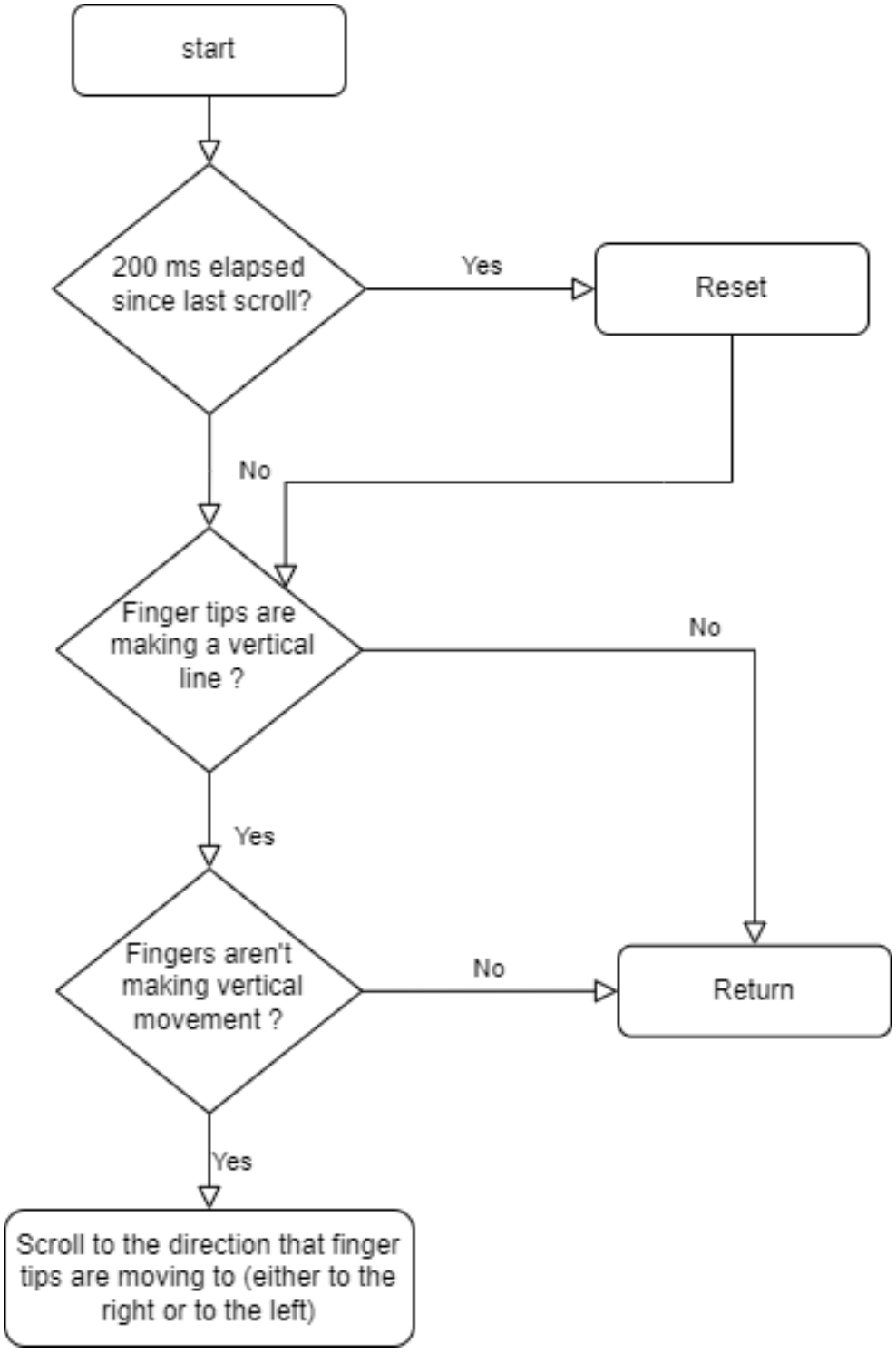


Figure 3.9: Flowchart of right and left scrolling

Similar to what happens for up and down scrolling, the function for checking left and right scrolling is called, too. This function works as follows: Firstly, it checks if the tips of the fingers (excluding the thumb) are making a vertical line and saves the hand in a variable called lastVertical. Then, in each call afterwards it checks if the finger tips are in the same vertical level as the tips in the lastVertical variable. Then, if the tips move right it scrolls right and vice versa calling the navigate function with the amount and the direction which is horizontal. All of the previous checks are done with an error margin of 0.1 of the canvas. The scrolling happens for 200 ms then the lastVertical is reset in order to stop the scrolling.

```
function handleLeftRightScroll() {
  if (lastVertical && Date.now() - lastVerticalTime > 200) {
    lastVertical = null
    lastVerticalTime = null
  }
  if (lastVertical == null) {
    if (isVertical()) {
      lastVertical = hand
      lastVerticalTime = Date.now()
    }
  } else {
    if (Math.abs(hand[8].y - lastVertical[8].y) < 0.1 && Math.abs(hand[12].y - lastVertical[12].y) < 0.1
      && Math.abs(hand[16].y - lastVertical[16].y) < 0.1 && Math.abs(hand[20].y - lastVertical[20].y) < 0.1) {
      // positive is left & negative is right
      if (hand[8].x - lastVertical[8].x > 0.1 && hand[12].x - lastVertical[12].x > 0.1
        && hand[16].x - lastVertical[16].x > 0.1 && hand[20].x - lastVertical[20].x > 0.1)
        navigate(-1, 'horizontal')
      if (lastVertical[8].x - hand[8].x > 0.1 && lastVertical[12].x - hand[12].x > 0.1
        && lastVertical[16].x - hand[16].x > 0.1 && lastVertical[20].x - hand[20].x > 0.1)
        navigate(1, 'horizontal')
    }
  }
}
```

Figure 3.10: handleLeftRightScroll function code

The isVertical function called in the handleLeftRightScroll function, which works like the isHorizontal function, simply checks if the four finger tips are making a vertical line with the difference in the x position between any 2 tips is at most 0.1 of the canvas.

```
function isVertical() {
  let min = 1, max = 0
  for (let i = 8; i < hand.length; i += 4) {
    min = Math.min(min, hand[i].x)
    max = Math.max(max, hand[i].x)
  }
  return max - min <= 0.1
}
```

Figure 3.11: isVertical function code

After the checks for different gestures are done, each gesture is mapped to its corresponding action via a function. These functions are moveMouse, navigate and clickMouse. These are actually functions which use axios Application Programming Interface (API) to connect with the backend which is responsible for mouse actions. The navigate function is responsible for scrolling and it takes scrollAmount and scrollDirection as parameters. The clickMouse function utilizes an additional condition to prevent continuous clicking as it allows clicking only once per 700 ms.

```
const moveMouse = (x, y) => {
  const data = { x: x, y: y }
  axios({ method: "post", url: `http://localhost:5000/moveMouse`, data })
    .then(response => console.log(response.data))
    .catch(error => console.error(error));
}

const navigate = (x, y) => {
  const data = { scrollAmount: x, scrollDirection: y }
  axios({ method: "post", url: `http://localhost:5000/navigate`, data })
    .then(response => console.log(response.data))
    .catch(error => console.error(error));
}

const clickMouse = () => {
  if (!lastClickTime || Date.now() - lastClickTime >= 700) {
    axios({ method: "post", url: `http://localhost:5000/clickMouse` })
      .then(response => console.log(response.data))
      .catch(error => console.error(error));
    lastClickTime = Date.now()
  }
}
```

Figure 3.12: Axios APIs code

3.3 Mouse control using RobotJS

RobotJS is a JavaScript library that allows for mouse and keyboard control. It provides functions to move the mouse cursor, simulate mouse clicks, scroll the mouse wheel, and send keyboard input programmatically. With RobotJS, we can automate tasks that require mouse and keyboard interactions, create macros, and perform UI testing. It works on multiple platforms and provides a cross-platform solution for controlling user input. It is mainly used in this project to control the mouse.

After requiring RobotJS in a constant called robot, we can use it to perform mouse moving and clicking.

For mouse movement, It's only required to call the function moveMouse(x, y) to move the cursor to the given point.

```
app.post('/moveMouse', (req, res) => {
    const { x, y } = req.body;
    robot.moveMouse(x, y);

    res.send('Mouse moved successfully');
});
```

Figure 3.13: backend for moveMouse function

moveMouse(x, y)

Moves mouse to x, y instantly, with the mouse button up.

Arguments

Argument	Description	Default
x		None
y		None

Figure 3.14: RobotJS moveMouse syntax

Similar to moveMouse, clickMouse() function is called to perform a left click in the current position. It takes no parameters because its default is to perform a left click.

```
app.post('/clickMouse', (req, res) => {
    robot.mouseClick()
    res.send('Mouse clicked successfully');
});
```

Figure 3.15: backend for clickMouse function

mouseClick([button], [double])

Clicks the mouse.

Arguments

Argument	Description	Default
`button`	Accepts left, right, or middle.	left
`double`	Set to true to perform a double click.	false

Figure 3.16: RobotJS mouseClick syntax

Unfortunately, RobotJS scrollMouse function doesn't work on windows 10/11 so The navigate function works a bit different. It initializes a java process to perform the scrolling. This java process utilizes java awt robot class which is capable of controlling the mouse. It passes the scroll amount and direction in the arguments.

```
app.post('/navigate', (req, res) => {
    const scrollAmount = req.body.scrollAmount;
    const scrollDirection = req.body.scrollDirection;
    const javaProcess
        = spawn('java', ['-cp', '.', 'scroll', scrollAmount, scrollDirection]);
    console.log(scrollAmount, scrollDirection)
    javaProcess.stdout.on('data', (data) => {
        console.log(`stdout: ${data}`);
    });

    javaProcess.stderr.on('data', (data) => {
        console.error(`stderr: ${data}`);
    });

    javaProcess.on('close', (code) => {
        console.log(`child process exited with code ${code}`);
        res.sendStatus(200);
    });
});
```

Figure 3.17: backend for navigate function

In the scroll.java class called by the navigate function, Java awt robot is initialized and the arguments are parsed. Then, a check for the scroll direction occurs. If the direction is vertical the scrolling happens immediately throw the mouseWheel function. On the other hand, if the direction is horizontal, the shift key is pressed with the scroll to perform horizontal scrolling and is released afterwards.

```java
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;

public class scroll {
    public static void main(String[] args) throws Exception {
        Robot robot = new Robot();
        int scrollAmount = Integer.parseInt(args[0]);
        String scrollDirection = args[1];

        if (scrollDirection.equals("vertical")) {
            robot.mouseWheel(scrollAmount);
        } else if (scrollDirection.equals("horizontal")) {
            robot.keyPress(KeyEvent.VK_SHIFT);
            robot.mouseWheel(-scrollAmount);
            robot.keyRelease(KeyEvent.VK_SHIFT);
        } else {
            System.out.println("Invalid scroll direction");
        }
    }
}
```

Figure 3.18: scroll.java class

3.4 User interface design using React

React [4], referred to as React.js or ReactJS, is a free to use JavaScript library utilized to construct user interfaces (UI) and UI components. It is overseen by Facebook, along with contributions from independent developers and organizations. React serves as a foundation for developing single-page or mobile applications. However, its primary focus lies in managing state and displaying that state on the Document Object Model (DOM).

In this project, React is used as the front-end framework for building the user interface. React provides a robust and efficient way to create interactive UI components that can seamlessly update and re-render based on changes in data or user interactions. It also allows to efficiently manage state and handle the rendering of components.

To enhance the visual appearance of the UI, the project also utilizes styles from MUI (Material-UI). MUI is a popular open-source library that offers pre-built components and styling options based on the Material Design guidelines. By integrating MUI with React, developers can easily create visually appealing and consistent UI elements, such as buttons, forms, navigation bars, and more.

3.4.1 User stories

The user interface is developed so that the user is able to:

- Navigate through available tickets
- Select a ticket to view its details
- Change the amount of copies of any booked ticket by increasing or decreasing its number or deleting completely.
- Navigate through booked tickets
- Proceed to payment screen after booking the desired tickets

Chapter 4

Results

In this section, we are going to discuss the results of The touchless screen museum ticket application developed using the integration of MediaPipe, RobotJS, and ReactJS. We will go through a user journey in the application discussing the process of booking tickets and the hand gestures needed in it.

4.1 Booking a ticket

Firstly, The application starts with a home screen which has a brief fact about the museum and the opening hours. The user can move the cursor using the index finger extended upwards and other fingers extended downwards as discussed in the methodology section. Then, the user can click the button labeled "Book tickets" to continue to the main booking screen. This can be done with the clicking gesture raising his index and middle fingers as discussed before.

Figure 4.1: The home screen

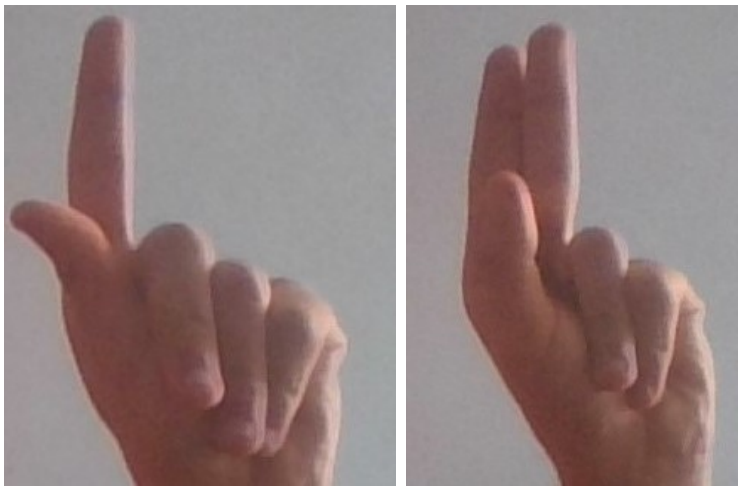


Figure 4.2: hand gestures for moving and clicking the mouse

4.1.1 Navigating through available tickets

After heading to the booking screen, The user can see the tickets in the left side, click on them to select and scroll between them. The main area of the screen appears blank until the user chooses a ticket to by displayed with its info and number.

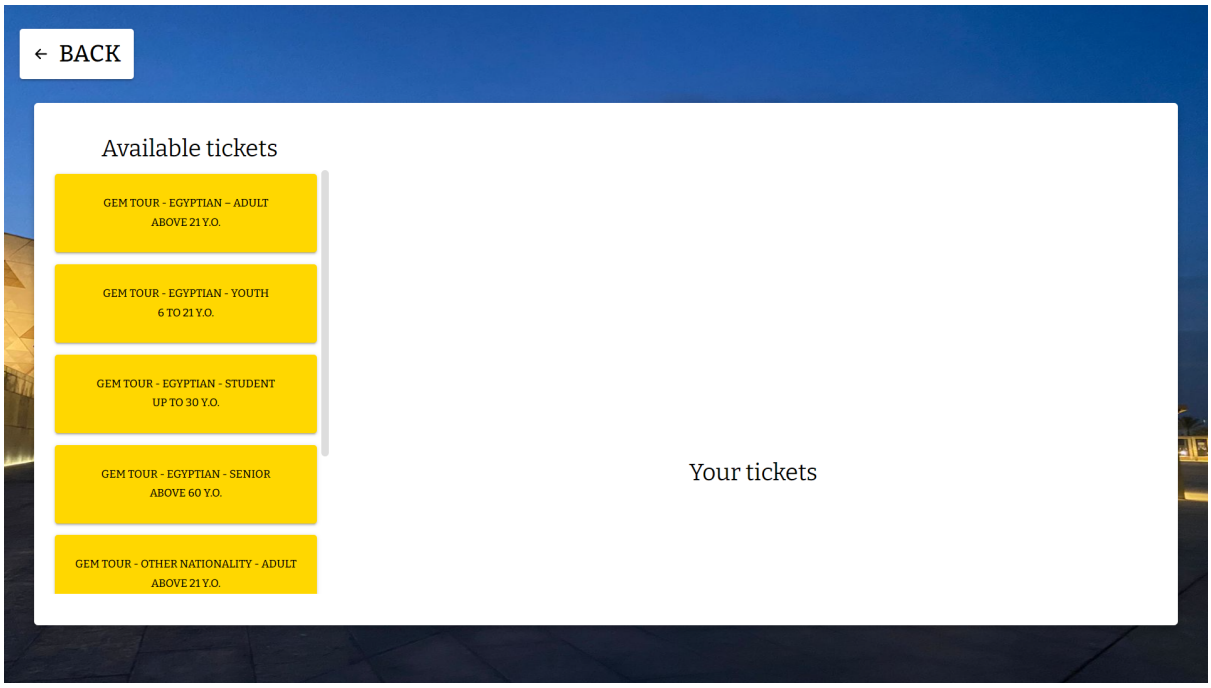


Figure 4.3: The booking screen

Scrolling through available tickets can be done by making a horizontal line with the tips of the fingers (excluding the thumb) and moving them upwards or downwards.

Figure 4.4: hand gesture for scrolling down

4.1.2 Choosing the number of copies of the selected ticket

After choosing the desired ticket, the user can freely choose the number of copies to buy through adding a ticket, removing one or deleting all the copies. Note that the user can only proceed to payment if they have something booked as the proceed button only shows after adding some tickets.

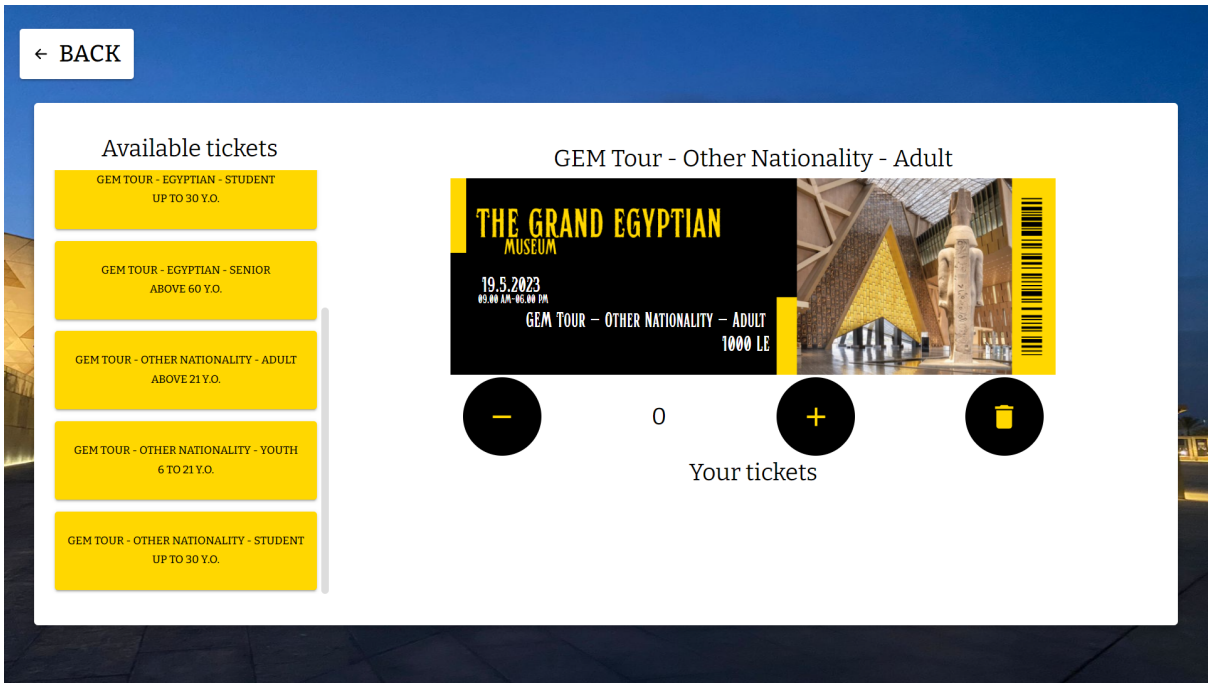


Figure 4.5: The booking screen with the selected ticket info

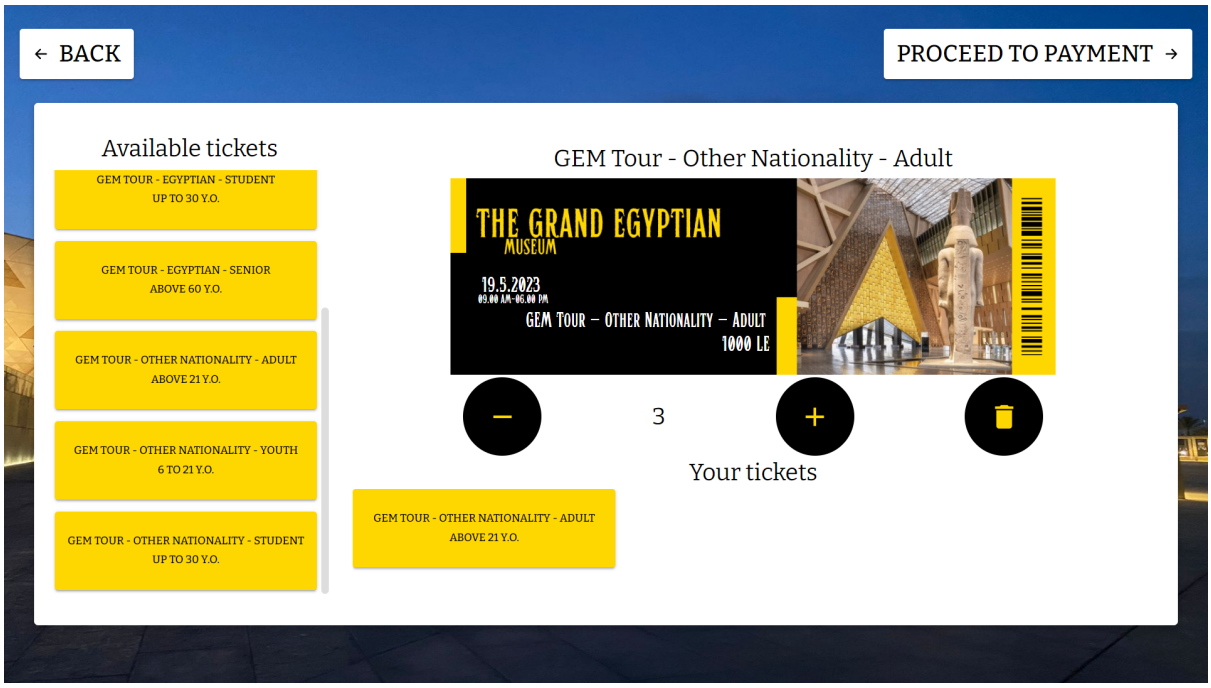


Figure 4.6: The booking screen after adding some copies of a ticket

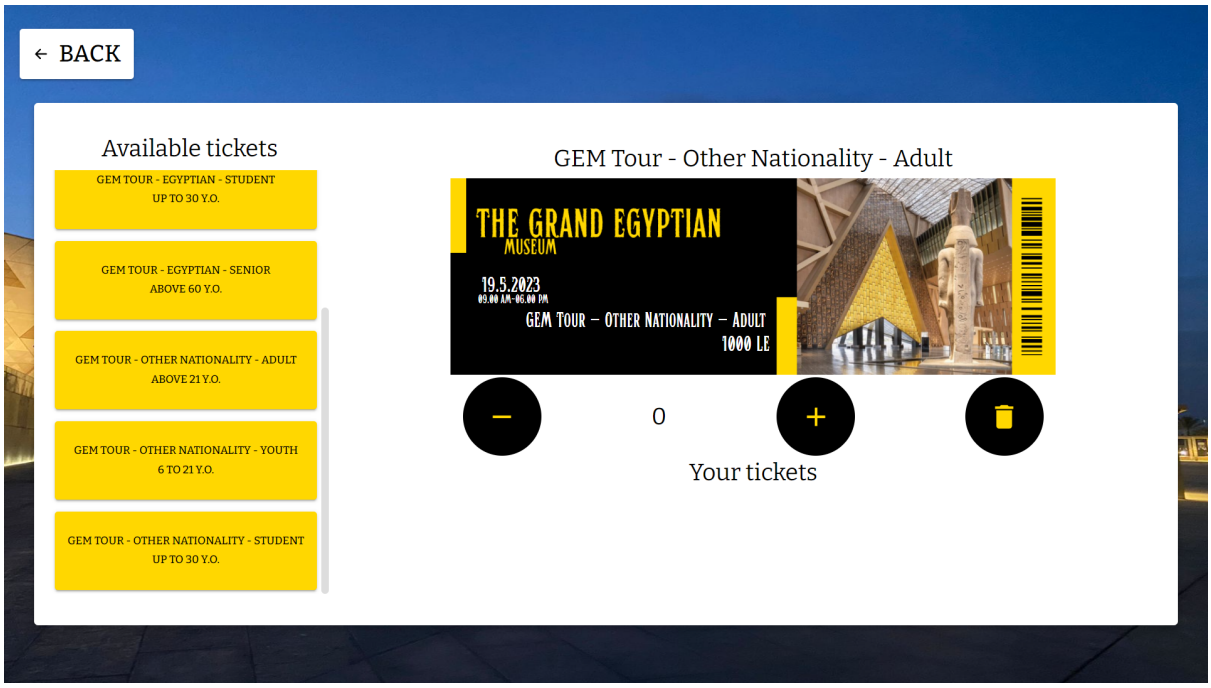


Figure 4.7: The booking screen with after removing the selected ticket

4.1.3 Navigating through booked tickets

After booking multiple tickets, they appear in the "your tickets" section. The user can navigate through them using the horizontal scrolling hand gesture making a vertical line with the tips of their fingers and moving them to the left or to the right.

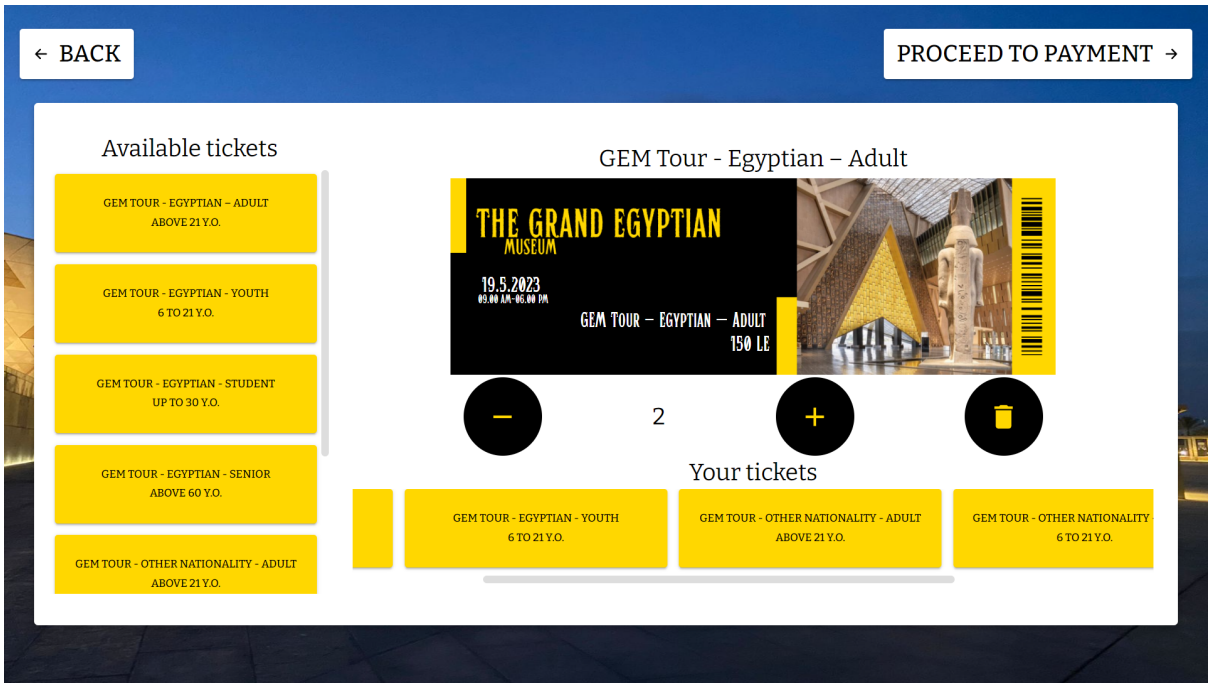


Figure 4.8: The booking screen with multiple booked tickets

Figure 4.9: Hand gesture for scrolling right

4.1.4 Payment

Finally, the user proceeds to the payment screen by clicking the proceed to payment button. This screen show the booked tickets and their total price. Then, the user can pay with their credit card through an external device.

The user can at any time press the back button to return to the booking screen again and add or remove any number of tickets without clearing their previous progress.

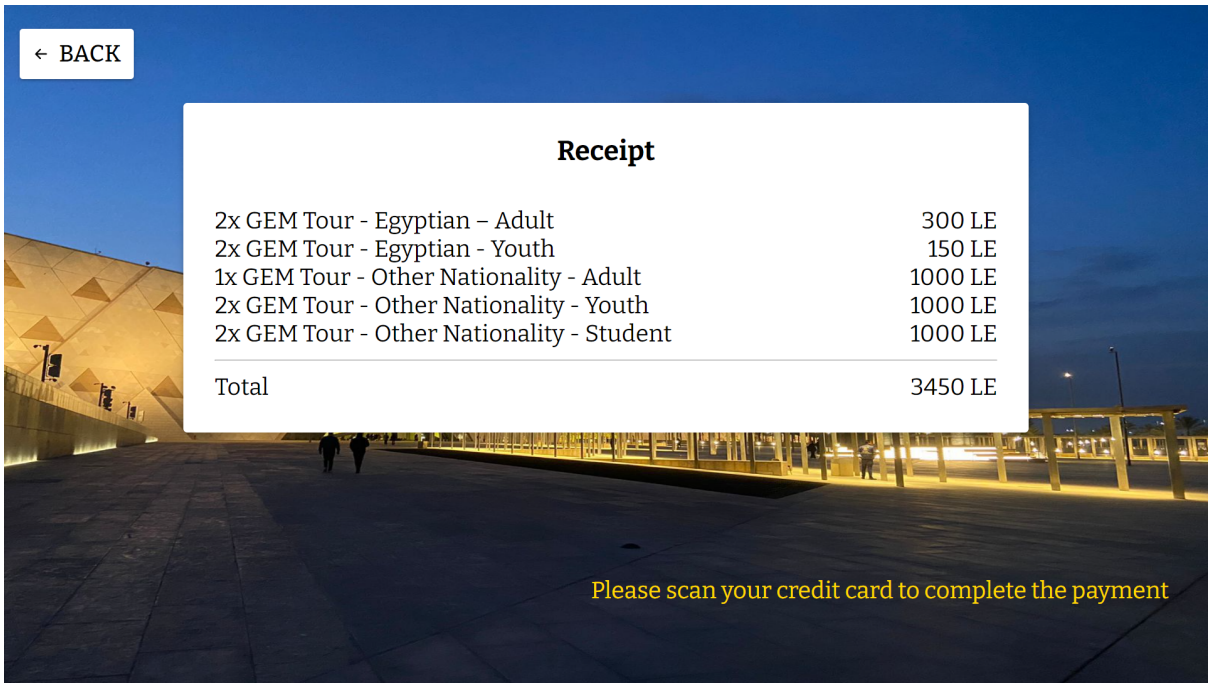


Figure 4.10: The payment screen

Chapter 5

Conclusion and future work

5.1 Conclusion

As the research has demonstrated, touchless technology has become necessary to use after the COVID-19 pandemic for its hygienic benefit. In this paper, A museum tickets screen application was developed using computer vision techniques via Google MediaPipe library and computer automation through RobotJS. Moreover, React was used to create a unique and appealing user experience with high learnability and accessibility. The system effectively translated hand movements into corresponding actions, enabling tasks such as cursor movements, clicking, and scrolling easily with high accuracy. This makes the journey of booking tickets not only full hygienic, but also of much fun.

5.2 Future work

While the touchless screen application demonstrated good performance, there are several areas for future enhancement and exploration such as:

- Optimizing the environmental adaptability and the performance in different lighting conditions, complex backgrounds, and varying camera setups.
- Improving hand recognition in low lighting conditions in night.
- Designing a new JavaScript library for automation and mouse control instead of RobotJs as it has some compatibility issues.
- Adding a narrator to improve accessibility.
- Designing more hand gestures to avoid using clicking so much. This can include adding hand gestures for number detection to make booking alot of tickets faster.

- Implementing an easy to use touchless payment method like QR codes.
- Changing the camera perspective to be able to make gestures that require moving fingers towards the screen without changing the cursor position.
- Adding an option to customize the gestures to accommodate users with different hand sizes, physical capabilities, and preferences.

Appendix

Appendix A

Lists

ML	Machine learning
CAGR	Compound Annual Growth Rate
RFID	Radio frequency identification
NFC	Near field communication
QR	Quick response
DBMs	Deep Boltzmann Machines
CNNs	Convolutional Neural Networks
DBNs	Deep Belief Networks
IoT	Internet of things
API	Application Programming Interface
UI	user interfaces
HGR	hand gesture recognition
CT	computer tomography
MRI	magnetic resonance imagery

List of Figures

Bibliography

[1] Anastasios Doulamis Eftychios Protopapadakis Athanasios Voulodimos, Nikolaos Doulamis. Deep learning for computer vision: A brief review", computational intelligence and neuroscience. 2018.

[2] Taiwo Oladipupo Ayodele. Introduction to machine learning. In Yagang Zhang, editor, *New Advances in Machine Learning*, chapter 1. IntechOpen, Rijeka, 2010.

[3] V. Bazarevsky and F. Zhang. On-device, real-time hand tracking with mediapipe, – google ai blog. 2019.

[4] Yogeshchandra Puranik. Bhupati Venkat Sai Indla. Review on react js. 2021.

[5] Muhammad Zahid Iqbal and Abraham G. Campbell. From luxury to necessity: Progress of touchless interaction technology. *Technology in Society*, 67:101796, 2021.

[6] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for perceiving and processing reality. In *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) 2019*, 2019.

[7] R. Malpass. Touchless technology: Past, present, and future. *Ombori*, 2021.

[8] Lucas Mohimont, François Alin, Marine Rondeau, Nathalie Gaveau, and Luiz Angelo Steffenel. Computer vision and deep learning for precision viticulture. *Agronomy*, 12(10), 2022.

[9] Kenton O'Hara, Gerardo Gonzalez, Abigail Sellen, Graeme Penney, Andreas Varnavas, Helena Mentis, Antonio Criminisi, Robert Corish, Mark Rouncefield, Neville Dastur, and Tom Carrell. Touchless interaction in surgery. *Commun. ACM*, 57(1):70–77, jan 2014.

[10] Aditi Ohol, Suman Goudar, Sukita Shettigar, and Shubhum Anand. Touchless touch screen user interface. 2017.

[11] Anjali Patil and Shilendra Patil. Hand gesture recognition system for controlling vlc media player based on two stream transfer learning. 12 2022.

[12] B. Rommel. BMW gesture control - the next level of Idrive Interaction, BMW gesture control: How it works? How to use it? *BimmerTech*, 2020.