Contents

List of Figures

Abstract

Writing in air has lately become one of the most interesting and difficult research areas in image processing and pattern recognition. A wide range of human-machine interactions benefit from this technology's inclusion. Numerous researches have investigated new methods and tactics for reducing processing time and improving recognition accuracy. In Computer Vision, object tracking is a crucial part of the task. Object tracking systems have become more popular as a result of faster computers, more affordable and higher quality video cameras, and the increasing need for automated video analysis. Typically, video analysis involves three major steps: identifying the object, tracking its movement from frame to frame, and lastly analyzing its behavior. Object tracking takes into account four issues: object representation, tracking feature selection, object identification, and object tracking. Object tracking algorithms are utilized in a variety of real-world applications, including video indexing, autonomous surveillance, and vehicle navigation. Motion-to-text converters for intelligent wearable electronics that can write in the air are the subject of this study. This initiative serves as a time capsule for transitory movements. It will use computer vision to track the path of the finger. Messages, emails, etc. maybe be sent using the text that has been generated. It will be a great help for deaf individuals to communicate. It's a good way to communicate without having to write

Keywords-*Air Writing, Character Recognition, Object Detection, Real-Time Gesture Control System, Smart Wearable, Computer Vision.*

Chapter 1

INTRODUCTION

1.1 Introduction

For most people nowadays, digital art takes the place of traditional writing as their primary means of expression. We call it "digital art" when we use digital medium to convey our artistic expressions and messages. Digital manifestation is distinguished by its reliance on modern science and technology. There were different kinds of art forms before digital art was created. There are four types of art: visual art, audio art, audio-visual, and audio-visual imaginative. This includes everything from literature to painting to sculpture to architecture to music to dance and theatre. However, digital and traditional creative forms are inextricably intertwined. However, human life's fundamental necessities are the driving force behind societal progress. There is a similar phenomenon in art. The link between digital and traditional art is now so intertwined that we need to study both of them meticulously. The old-school method of writing is when you use a pen and paper or a blackboard. Developing a hand motion recognition system that can be used to digitally write has been the fundamental objective of digital art. It is possible to produce digital art by utilizing a keyboard, touch-screen surface, digital pen, stylus or even electronic hand gloves. Machine learning algorithms and python programming, on the other hand, are used to recognize hand gestures in order to facilitate human-machine communication organically. We're seeing a rise in the need for natural 'human-computer interaction (HCI) systems as technology progresses. .

1.2 Purpose

The purpose of an air canvas project is to create a virtual drawing tool that allows users to draw or write in the air using gestures, which are then captured and rendered on a digital canvas. This kind of project involves the use of various technologies such as computer vision, motion sensors, and machine learning to detect and interpret the user's gestures. By doing so, it provides an innovative way to interact with digital devices without the need for traditional input methods like a mouse or touchscreen.

One significant application of air canvas technology is in education. By enabling teachers and students to interact with digital content through gestures, it enhances the learning experience, making it more engaging and interactive. For instance, teachers can write or draw diagrams in the air, which are then displayed on a digital screen for the class to see. This can make complex concepts easier to understand and help maintain student interest.

In the realm of art and creativity, an air canvas offers artists a new medium to express themselves. Artists can create digital art using hand gestures, providing a unique way to experiment with shapes, lines, and forms. This can open up new possibilities for creative expression, allowing artists to create in a more fluid and dynamic manner.

For individuals with disabilities, air canvas technology can significantly improve accessibility. It enables people who may have difficulty using traditional input devices to interact with computers and digital content more easily. By using gestures, they can perform tasks that would otherwise be challenging, thereby promoting greater independence and inclusivity.

The gaming and entertainment industries also benefit from air canvas technology. It allows for the creation of immersive gaming experiences where players can interact with the game environment through natural movements. This can make games more engaging and fun, as players feel more physically involved in the gameplay.

In design and prototyping, air canvas technology provides a valuable tool for designers and engineers. It allows them to sketch and prototype ideas quickly in a virtual space, facilitating rapid iteration and collaboration. This can lead to more efficient design processes and the development of better products.

In the fields of virtual and augmented reality, air canvas technology enhances the user experience by enabling users to draw and manipulate objects in a 3D space using hand gestures. This can make VR and AR applications more intuitive and immersive, providing a more seamless interaction between the user and the virtual environment.

Finally, air canvas projects contribute to user interface innovation. By experimenting with new ways of interacting with digital devices, they can lead to the development of more intuitive and natural user interfaces. This can improve the overall user experience and make technology more accessible to a wider range of people. Overall, the air canvas project aims to bridge the gap between the physical and digital worlds, making interactions with technology more natural and seamless.

1.3 Motivation

The motivation for developing an air canvas using Python stems from the desire to leverage Python's versatility and ease of use to create an innovative and interactive drawing tool. Python, known for its simplicity and readability, is an ideal language for implementing complex algorithms and interfacing with various hardware components, making it perfect for a project that combines computer vision, machine learning, and gesture recognition. This ease of development allows both beginners and experienced developers to contribute to and expand upon the air canvas project.

Python's extensive ecosystem of libraries and frameworks is a significant motivator for using it in an air canvas project. Libraries such as OpenCV for computer vision, Mediapipe for hand tracking, and TensorFlow or PyTorch for machine learning provide robust tools for developing the core functionalities of an air canvas. These libraries facilitate the detection and interpretation of hand gestures, allowing for real-time rendering of drawings on a digital canvas. Python's rich set of tools accelerates development, enabling rapid prototyping and iteration, which is crucial for refining gesture recognition and improving user experience.

The open-source nature of Python fosters a collaborative environment where developers can share code, ideas, and improvements. This collaboration is essential for the ongoing development of an air canvas project, as it encourages community contributions and the sharing of best practices. Python's widespread use in both academia and industry ensures that there is a large pool of talent familiar with the language, which can lead to a diverse and vibrant community of contributors. This collective effort can drive innovation and help overcome technical challenges more effectively.

Python's cross-platform capabilities are another motivating factor. An air canvas developed in Python can run on various operating systems, including Windows, macOS, and Linux, making it accessible to a broad audience. This cross-platform compatibility ensures that the air canvas can be used in different environments, from classrooms and art studios to gaming and design labs. Python's ability to interface with different hardware, such as webcams and motion sensors, further enhances its suitability for an air canvas project.

Lastly, the motivation for using Python in an air canvas project is also driven by the potential for educational and research applications. Python is a popular language for teaching programming and computer science concepts, and an air canvas project can serve as an engaging educational tool. It can be used to teach topics such as computer vision, machine learning, and human-computer interaction in a hands-on and interactive manner. Additionally, researchers can use the air canvas as a platform for exploring new algorithms and techniques in gesture recognition and real-time interaction.

In summary, the motivation for developing an air canvas using Python is rooted in the language's simplicity, extensive ecosystem, collaborative nature, cross-platform capabilities, and potential for educational and research applications. These factors make Python an ideal choice for creating an innovative and interactive tool that bridges the gap between physical gestures and digital art.

Chapter 2

LITERATURE SURVEY

The concept of an air canvas leverages advancements in human-computer interaction (HCI), computer vision, and machine learning to create a system where users can draw or write in the air, with their gestures captured and displayed on a digital canvas. This literature survey explores various aspects of the project, focusing on the technological underpinnings, existing solutions, and relevant research in the field.

2.0.1 Human-Computer Interaction and Gesture Recognition

Gesture recognition is a key component of an air canvas system. Research in HCI has extensively explored the use of gestures as a natural form of interaction with digital systems. Wachs et al. (2011) provide a comprehensive review of gesture-based interfaces, highlighting the potential for natural and intuitive user interactions. The literature emphasizes the importance of accurate gesture detection and interpretation, which can be achieved through computer vision techniques and machine learning algorithms.

2.0.2 Computer Vision Techniques

The use of computer vision to track and interpret hand movements is critical for an air canvas. OpenCV, a widely used open-source computer vision library, offers robust tools for real-time image processing and feature extraction. Literature on hand tracking often references the use of techniques such as background subtraction, contour detection, and color-based segmentation to isolate and track hand movements. For instance, Chen et al. (2007) demonstrated a hand gesture recognition system using contour analysis and skin color detection, which forms the basis for many modern hand tracking systems.

2.0.3 Machine Learning and Deep Learning

Machine learning, particularly deep learning, has revolutionized gesture recognition. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are commonly used for this purpose. Studies like those by Molchanov et al. (2015) have shown the effectiveness of deep learning models in recognizing complex hand gestures with high accuracy. Frameworks such as TensorFlow and PyTorch facilitate the implementation of these models, allowing for real-time performance and adaptability to different users and environments.

2.0.4 Python Ecosystem for Development

Python's popularity in the development community is largely due to its simplicity and the availability of powerful libraries. The literature frequently cites Python's versatility in integrating various libraries and tools needed for an air canvas project. For example, OpenCV for computer vision, Mediapipe for hand tracking, and TensorFlow or PyTorch for machine learning are all well-documented and supported, providing a robust foundation for development. Additionally, Python's ease of use accelerates the development cycle, allowing for rapid prototyping and iterative improvements.

2.0.5 Existing Solutions and Applications

Several existing applications and research projects have explored the concept of air canvas and related technologies. Projects like Google's Teachable Machine and Microsoft's Kinect have paved the way for gesture-based interaction. The literature on these projects provides valuable insights into the challenges and solutions associated with gesture recognition and real-time interaction. These projects often highlight the importance of user experience, latency reduction, and the robustness of gesture detection in varying conditions.

2.0.6 Educational and Research Implications

The educational potential of an air canvas project is significant. Literature on educational technology emphasizes the benefits of interactive and engaging tools in enhancing learning outcomes. An air canvas can serve as a practical application for teaching programming, computer vision, and machine learning. Research by Papert (1980) on constructionist learning theories supports the idea that such hands-on projects can deepen understanding and foster creativity among students.

In summary, the literature survey on the air canvas project reveals a rich field of research and development spanning multiple disciplines. The integration of HCI principles, computer vision techniques, machine learning algorithms, and the Python ecosystem forms the foundation of this innovative project. Existing solutions and research provide valuable lessons and highlight the potential for educational and practical applications, underscoring the relevance and impact of developing an air canvas using Python.

Chapter 3

PROBLEM STATEMENT

3.1 Problem Statement

Current digital drawing and writing tools rely heavily on physical input devices like tablets, styluses, and touchscreens, which can limit the natural flow of creativity and accessibility for users. There is a need for a more intuitive and accessible way to interact with digital canvases that eliminates the dependency on physical devices.

The problem is to create a system that allows users to draw and write in the air using hand gestures, accurately capturing these gestures and rendering them in real-time on a digital canvas. This system must address challenges in gesture recognition accuracy, real-time processing, and user-friendly interaction to provide a seamless and engaging experience.

3.1.1 Explanation

The problem at hand is to develop an innovative ”Air Canvas” system using Python, enabling users to draw and write in the air with gestures rather than physical tools like tablets or styluses. This approach seeks to enhance creativity and accessibility by providing a more natural and intuitive interaction with digital canvases. Key challenges include achieving accurate gesture recognition in various environments and ensuring real-time processing for responsive user interaction.

Chapter 4

EXISTING SYSTEM

Graphics Tablets: These are devices with a flat surface on which users can draw using a stylus or pen. They often include pressure sensitivity and are widely used by artists and designers for precise digital artwork.

Styluses and Touchscreens: Devices like smartphones, tablets, and interactive displays allow users to draw directly on the screen using a stylus or their finger. They offer a more intuitive interface for digital drawing and note-taking.

Computer Mice: Although primarily used for general computer navigation, computer mice with advanced features (like scroll wheels and programmable buttons) can also be used for basic digital drawing tasks.

Graphics Software and Input Devices: Professional graphic design software (e.g., Adobe Photoshop, Illustrator) typically supports input from various devices like graphics tablets, styluses, and mice, providing extensive tools and functionalities for digital artwork creation.

4.0.1 LIMITATIONS OF EXISTING SYSTEM

The traditional systems for digital drawing and interaction, such as graphics tablets, styluses, touchscreens, and computer mice, come with several limitations:

Portability: Graphics tablets and some advanced styluses require a separate physical device, which may not be convenient to carry around compared to a system like Air Canvas, which uses only a colored tip or cap.

Physical Constraints: Devices like graphics tablets and styluses impose physical constraints on users, requiring them to interact with a specific tool or surface. This can limit natural movement and may not be suitable for all users, especially those who prefer a more fluid and natural interaction.

Cost: High-quality graphics tablets and styluses can be expensive, making them less accessible to individuals or organizations with budget constraints.

Learning Curve: Some traditional devices, such as graphics tablets with styluses, have a learning curve that may deter new users from adopting them for digital drawing. This can impact usability and user satisfaction.

Maintenance: Traditional devices like graphics tablets and styluses require maintenance, such as periodic cleaning and calibration, to ensure optimal performance and longevity.

Accessibility: While touchscreens have improved accessibility, they may still pose challenges for users with certain disabilities or those who require alternative input methods.

Hygiene: Especially in shared environments, traditional input devices like styluses and touchscreens may raise hygiene concerns due to frequent physical contact.

Precision and Sensitivity: The precision and sensitivity of traditional devices can vary, affecting the accuracy and quality of digital drawings, especially for intricate designs or detailed artwork.

Dependency on External Software: Many traditional drawing devices rely on external software applications for advanced functionalities, which may require additional learning and integration efforts.

Environmental Limitations: Lighting conditions and background interference can affect the performance of traditional devices, such as styluses and touchscreens, impacting their usability in different environments.

These limitations highlight the potential benefits of innovative systems like Air Canvas, which aim to overcome some of these challenges by offering a more intuitive, touchless, and accessible approach to digital drawing and interaction.

Chapter 5

PROPOSED SYSTEM

This computer vision experiment uses an Air canvas, which allows you to draw on a screen by waving a finger equipped with a colorful tip or a basic colored cap. These computer vision projects would not have been possible without OpenCV's help. There are no keypads, styluses, pens, or gloves needed for character input in the suggested technique.

5.1 FEATURES OF AIR CANVAS

- Can track any specific colored pointer.
- User can draw in four different colors and even change them without any hustle.
- Able to rub the board with a single location at the top of the screen.
- No need to touch the computer once the program is run.

5.2 SCOPE

The scope of computer vision and that of OpenCV is huge.Object (human and non-human) detection in both commercial as well as governmental space is huge and already happens is many ways.

Transportation - with autonomous driving in ADAS (Automated Driver Assist System) in traffic signs detection, pedestrian detection, safety features such driver fatigue detection etc.

Medical imaging - mammography, cardiovascular and microscopic image analysis (I'm not a medicine guy but I am hearing that a whole lot of computer imaging aided decision-making such as automated detection and counting of microorganisms will involve use of OpenCV)

Manufacturing - Ton of computer vision stuff there as well such as rotation invariant detection on a conveyer belt with detection of stoke of robotic gripping.

Public order and security - pedestrian/citizen detection and tracking, mob management, prediction of future events.

5.3 APPLICATIONS

Python may be used to quickly analyze photos and videos and extract meaningful information from them, thanks to the many methods provided in OpenCV. Other frequent uses include,

Image Processing:

There are several ways in which the OpenCV may be used to process and interpret images, such as altering their shape, colour, or extracting important information from the supplied picture and writing it into a new image.

Face Detection:

By employing Haar-Cascade Classifiers, either from locally recorded videos or photos or from live streaming through web camera.

Face Recognition:

In order to identify faces in the films, face identification was performed using OpenCV by generating bounding boxes (rectangles) and subsequently model training using ML methods.

Object Detection:

OpenCV and YOLO, an object identification method, may be used to identify moving or stationary objects in images and videos.

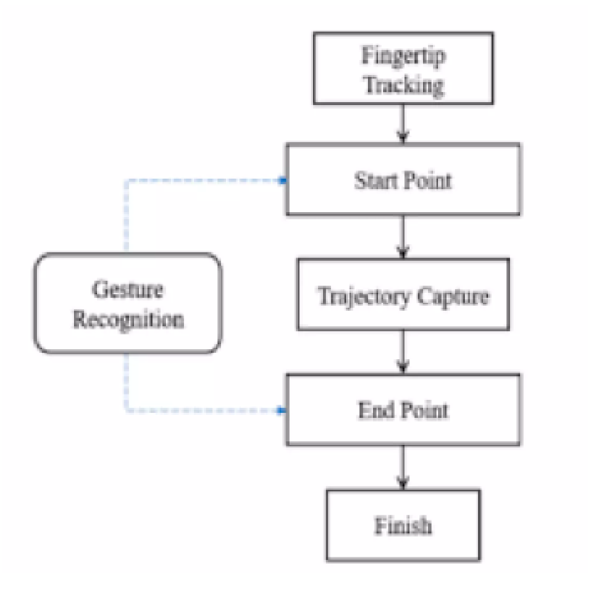


Figure 5.1: flowchart of proposed method

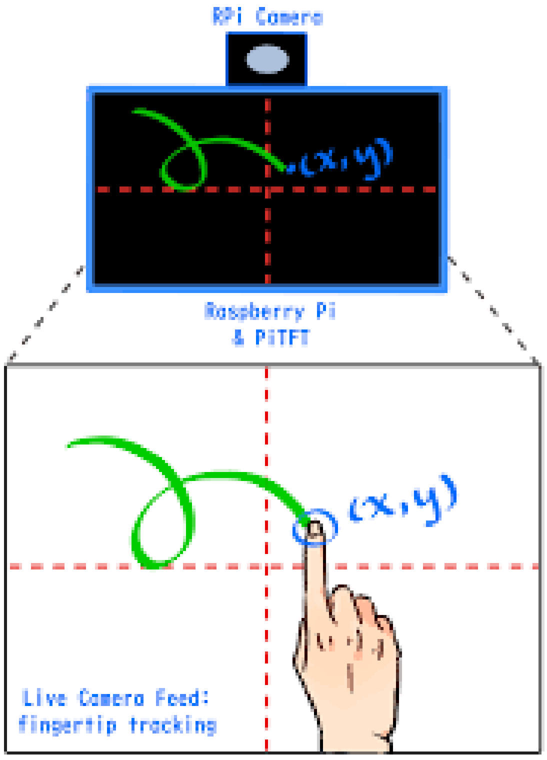


Figure 5.2: project overview

Chapter 6

METHODOLOGY

This system needs a dataset for the Fingertip Detection Model. The Fingertip Model's primary purpose is used to record the motion, i.e., the air character.

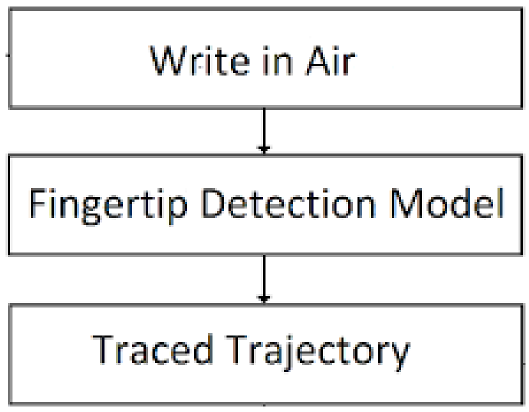


Figure 6.1: Methodology

6.1 Fingertip Detection Model:

For air writing, all you need is a stylus or one of the many colored airpens on the market. But the system relies on the tip of a finger to operate it. We think that individuals should be able to write in the air without having to carry a pen around with them. Every frame has been analyzed using Deep Learning techniques to generate a list of coordinates for each fingertip.

6.2 Techniques of Fingertip recognition Dataset Creation:

1.Video to Images:
Hand motion films were recorded for two seconds and then transferred to a variety of diverse contexts. This footage was sliced up into 30 individual pictures, which can be seen in More than 2000 photos were taken. Data was manually annotated in this dataset. The most accurate model developed using this dataset has a precision of 99.5 percent. Dataset monotony resulted from using a same movie with identical surroundings to produce 30 different photos. It follows that for non-continuous environments like those in the sample, the model failed miserably..

2.Take Pictures in Distinct Backgrounds:
We built a new dataset to solve the problem of the prior method's lack of variety. This time, we were well aware of the fact that gestures were required in order to operate the system. To that end, we gathered the four hand gestures shown in Figure 4. The goal was to create a model that could recognize the tips of all four fingers with equal efficiency. The user would be able to manage the system by showing the number of fingers he has. One index finger is all that's required to start writing, and the user can now - quickly write by showing one index finger, convert this writing motion to e-text using two fingers, add space using three fingers, hit backspace using five fingers, and then show 1,2,3 fingers to select the first, second, and third predictions respectively. Show five fingers to exit the prediction mode. The photos in this collection totaled 1800. An auto-labeled dataset was created using a software that has previously been trained on this dataset Once the photos had been properly renamed, a new model was introduced. At least 94the time, it worked. Different backdrops proved to be suitable for the model, unlike the previous one

Chapter 7

ALGORITHM

Step 1: Start reading the frames and convert the captured frames to HSV colour space. (This makes it easier for colour detection.)

Step 2: Prepare the canvas frame and place the respective ink buttons on it.

Step 3: Adjust the track bar values to find the mask of the coloured marker.

Step 4: Preprocess the mask using morphological operations, such as erosion and dilation.

Step 5: Detect the contours, find the center coordinates of the largest contour, and keep storing them in an array for successive frames. (These arrays are used for drawing points on the canvas.)

Step 6: Finally, draw the points stored in the array on the frames and canvas.

The Air Canvas algorithm effectively enables drawing in mid-air by capturing the motion of a colored marker and translating it onto a digital canvas. By converting frames to the HSV color space, it simplifies color detection, ensuring accurate tracking of the marker. The use of morphological operations helps in refining the mask, and contour detection allows pinpointing the marker's position. The algorithm's ability to store and use these positions for drawing makes it a powerful tool for creating digital art without any physical contact. This innovative approach opens up new possibilities for interactive applications and creative expression.

Chapter 8

SOFTWARE AND HARDWARE REQUIREMENTS

8.1 Software Requirements

Here are the software and hardware requirements with added for LaTeX:
Software Requirements

- **Python**: The primary programming language used for developing the Air Canvas application.
- **OpenCV**: A powerful library for computer vision tasks, used for processing the video frames and handling image operations.
- **NumPy**: A library for numerical computations, essential for handling arrays and performing mathematical operations.
- **Matplotlib** (optional): Useful for visualizing the drawn points and debugging.
- **IDE**: An Integrated Development Environment like PyCharm, VSCode, or Jupyter Notebook for writing and testing the code.
- **Additional Libraries**: Any other libraries for GUI development (e.g., Tkinter) if required for creating a user interface for the application.

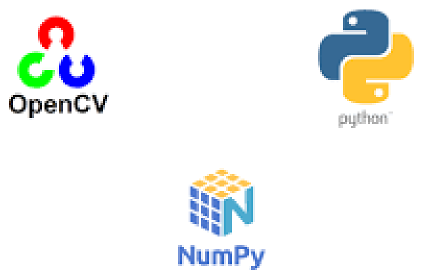


Figure 8.1: Software requirements

8.2 Hardware Requirements

- **Camera**: A webcam or any camera module capable of capturing video frames.
- **Computer**: A computer with a decent processor (e.g., Intel i5 or above) and sufficient RAM (e.g., 4GB or more) to handle real-time video processing.
- **Colored Marker**: Any brightly colored object that can be easily distinguished from the background for tracking.
- **Display Monitor**: To view the output of the Air Canvas and interact with the application.

Chapter 9

SOURCE CODE

```
import numpy as np
    import cv2
    from collections import deque
    ” default called trackbar function ”
    def setValues(x):
    print(””)
    ” Creating the trackbars needed for ”
    ” adjusting the marker colour These ”
    ” trackbars will be used for setting ”
    ” the upper and lower ranges of the”
    ” HSV required for particular colour ”
    cv2.namedWindow(”Color detectors”)
    cv2.createTrackbar(”Upper Hue”, ”Color detectors”, 153, 180, setValues)
    cv2.createTrackbar(”Upper Saturation”, ”Color detectors”, 255, 255, setValues)
    cv2.createTrackbar(”Upper Value”, ”Color detectors”, 255, 255, setValues)
    cv2.createTrackbar(”Lower Hue”, ”Color detectors”, 64, 180, setValues)
    cv2.createTrackbar(”Lower Saturation”, ”Color detectors”, 72, 255, setValues)
    cv2.createTrackbar(”Lower Value”, ”Color detectors”, 49, 255, setValues)
    ” Giving different arrays to handle colour”
    ” points of different colour These arrays ”
    ” will hold the points of a particular colour”
    ” in the array which will further be used to draw on canvas ”
    bpoints = [deque(maxlen = 1024)]
    gpoints = [deque(maxlen = 1024)]
    rpoints = [deque(maxlen = 1024)]
    ypoints = [deque(maxlen = 1024)]
    ” These indexes will be used to mark position of pointers in colour array ”
    blue_index = 0
    green_index = 0
    red_index = 0
    yellow_index = 0
    ” The kernel to be used for dilation purpose ”
    kernel = np.ones((5, 5), np.uint8)
    ” The colours which will be used as ink for the drawing purpose ”
    colors = [(255, 0, 0), (0, 255, 0),
    (0, 0, 255), (0, 255, 255)]
    colorIndex = 0
    ” Here is code for Canvas setup ”
    paintWindow = np.zeros((471, 636, 3)) + 255
    cv2.namedWindow(’Paint’, cv2.WINDOW_AUTOSIZE)
    ” Loading the default webcam of PC. ”
    cap = cv2.VideoCapture(0)
    ” Keep looping ”
    while True:
    ” Reading the frame from the camera ”
    ret, frame = cap.read()
    ” Flipping the frame to see same side of yours ”
    frame = cv2.flip(frame, 1)
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    ” Getting the updated positions of the trackbar and setting the HSV values ”
    u_hue = cv2.getTrackbarPos(”Upper Hue”, ”Color detectors”)
    u_saturation = cv2.getTrackbarPos(”Upper Saturation”, ”Color detectors”)
    u_value = cv2.getTrackbarPos(”Upper Value”, ”Color detectors”)
    l_hue = cv2.getTrackbarPos(”Lower Hue”, ”Color detectors”)
    l_saturation = cv2.getTrackbarPos(”Lower Saturation”, ”Color detectors”)
    l_value = cv2.getTrackbarPos(”Lower Value”, ”Color detectors”)
    Upper_hsv = np.array([u_hue, u_saturation, u_value])
    Lower_hsv = np.array([l_hue, l_saturation, l_value])
    ” Adding the colour buttons to the live frame for colour access ”
    frame = cv2.rectangle(frame, (40, 1), (140, 65), (122, 122, 122), -1)
    frame = cv2.rectangle(frame, (160, 1), (255, 65), colors[0], -1)
    frame = cv2.rectangle(frame, (275, 1), (370, 65), colors[1], -1)
    frame = cv2.rectangle(frame, (390, 1), (485, 65), colors[2], -1)
    frame = cv2.rectangle(frame, (505, 1), (600, 65), colors[3], -1)
    cv2.putText(frame, ”CLEAR ALL”, (49, 33), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, ”BLUE”, (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, ”GREEN”, (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, ”RED”, (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, ”YELLOW”, (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(150, 150, 150), 2, cv2.LINE_AA)
    ” Identifying the pointer by making its mask ”
    Mask = cv2.inRange(hsv, Lower_hsv, Upper_hsv)
    Mask = cv2.erode(Mask, kernel, iterations = 1)
    Mask = cv2.morphologyEx(Mask, cv2.MORPH_OPEN, kernel)
    Mask = cv2.dilate(Mask, kernel, iterations = 1)
    ” Find contours for the pointer after identifying it ”
    cnts, _ = cv2.findContours(Mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPL
    center = None
    ” If the contours are formed ”
```

```
if len(cnts) >0:
" sorting the contours to find biggest "
cnt = sorted(cnts, key = cv2.contourArea, reverse = True)[0]
" Get the radius of the enclosing circle
around the found contour "
((x, y), radius) = cv2.minEnclosingCircle(cnt)
" Draw the circle around the contour "
cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
" Calculating the center of the detected contour "
M = cv2.moments(cnt)
center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))
"Now checking if the user wants to click on any button above the screen "
if center[1] ≤ 65 :
" Clear Button "
if 40 ≤ center[0] ≤ 140 :
bpoints = [deque(maxlen = 512)]
gpoints = [deque(maxlen = 512)]
rpoints = [deque(maxlen = 512)]
ypoints = [deque(maxlen = 512)]
blue_index = 0
green_index = 0
red_index = 0
yellow_index = 0
paintWindow[67:, :, :] = 255
elif 160 ≤ center[0] ≤ 255 :
colorIndex = 0 " Blue "
elif 275 ≤ center[0] ≤ 370 :
colorIndex = 1 " Green "
elif 390 ≤ center[0] ≤ 485 :
colorIndex = 2 " Red "
elif 505 ≤ center[0] ≤ 600 :
colorIndex = 3 " Yellow "
else :
if colorIndex == 0:
bpoints[blue_index].appendleft(center)
elif colorIndex == 1:
gpoints[green_index].appendleft(center)
elif colorIndex == 2:
rpoints[red_index].appendleft(center)
elif colorIndex == 3:
ypoints[yellow_index].appendleft(center)
" Append the next deques when nothing is detected to avoid messing up "
else:
bpoints.append(deque(maxlen = 512))
blue_index += 1
gpoints.append(deque(maxlen = 512))
green_index += 1
rpoints.append(deque(maxlen = 512))
```

```
red_index += 1
ypoints.append(deque(maxlen = 512))
yellow_index += 1
” Draw lines of all the colors on the canvas and frame ”
points = [bpoints, gpoints, rpoints, ypoints]
for i in range(len(points)):
for j in range(len(points[i])):
for k in range(1, len(points[i][j])):
if points[i][j][k - 1] is None or points[i][j][k] is None:
continue
cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)
cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)
” Show all the windows ”
cv2.imshow(”Tracking”, frame)
cv2.imshow(”Paint”, paintWindow)
cv2.imshow(”mask”, Mask)
” If the ’q’ key is pressed then stop the application ”
if cv2.waitKey(1)  0xFF == ord(”q”):
break
” Release the camera and all resources ”
cap.release()
cv2.destroyAllWindows()
```

9.1 FLOWCHART OF THE LOGIC CODE

1. Start: The program starts here.

2. Open Webcam: Opens the default webcam of the PC.

3. Trackbars: Creates trackbars for adjusting color ranges for detection.

4. Canvas: Defines the canvas (paintWindow) for drawing.

5. Read Frame: Reads a frame from the webcam.

6. Flip Frame: Flips the frame horizontally.

7. Convert to HSV: Converts the frame from BGR to HSV color space.

8. Get Trackbar Positions: Gets the current positions of the trackbars (for color ranges).

9. Get ROI Values: Calculates the upper and lower HSV values based on trackbar positions.

10. Create Mask: Creates a mask based on the defined HSV range.

11. (Clear Button Pressed): Checks if the clear button is pressed.

12. Update Mask: If clear button is pressed, resets the mask and clears the canvas.

13. Find Contours: Finds contours in the mask (potential pointer location).

14. Draw Circle: If a contour is found, draws a circle around it.

15. (Center Detected): Checks if the center of the contour is detected.

16. (Any Color Button): Checks if any color button is pressed.

17. Set Color Index: Sets the color index based on the pressed color button.

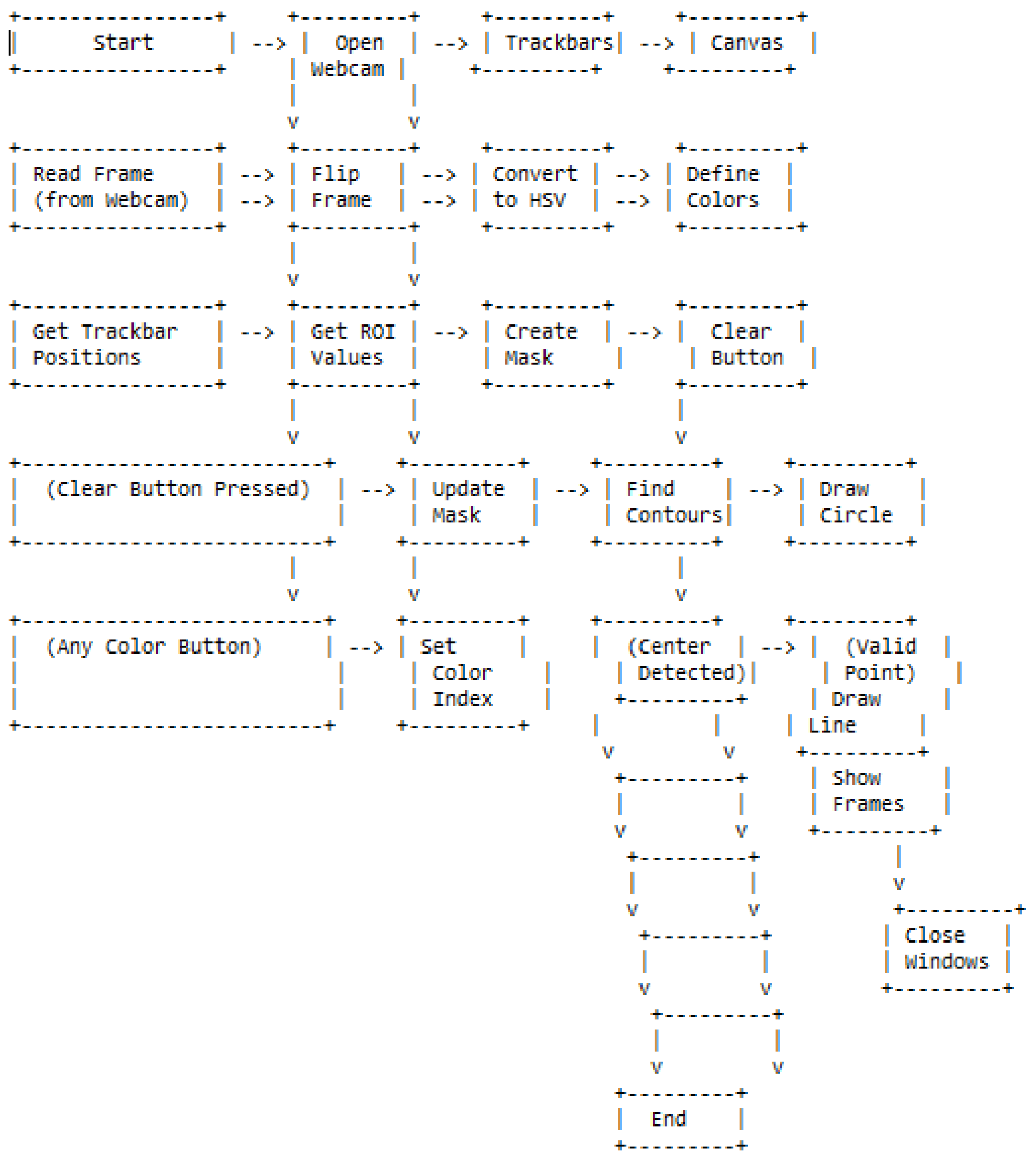
18. (Valid Point): Checks if a valid point (center) is detected.

19. Draw Line: If a valid point is detected and a color is chosen, draws a line on the frame and canvas based on the chosen color.

20. Show Frames: Shows the original frame (”Tracking”), mask (”mask”), and the drawing canvas (”Paint”).

21. (Close Windows): Checks if the ’q’ key is pressed to quit the application.

22. End: The program ends here.
This flowchart represents the main logic of the code. Here’s a brief explanation of each step:



```
+----------------+     +---------+     +---------+     +---------+
|      Start      | --> |  Open   | --> | Trackbars| --> | Canvas  |
+----------------+     | Webcam |     +---------+     +---------+
                       |         |
                       v         v
+----------------+     +---------+     +---------+     +---------+
| Read Frame     | --> | Flip    | --> | Convert | --> | Define  |
| (from Webcam)  | --> | Frame   | --> | to HSV  | --> | Colors  |
+----------------+     +---------+     +---------+     +---------+
                       |         |
                       v         v
+----------------+     +---------+     +---------+     +---------+
| Get Trackbar   | --> | Get ROI | --> | Create  | --> |  Clear  |
| Positions      |     | Values  |     | Mask     |     | Button  |
+----------------+     +---------+     +---------+     +---------+
                       |         |                     |
                       v         v                     v
+------------------------+     +---------+     +---------+     +---------+
|  (Clear Button Pressed) | --> | Update  | --> | Find    | --> | Draw    |
|                        |     | Mask    |     | Contours|     | Circle  |
+------------------------+     +---------+     +---------+     +---------+
                       |         |                 |
                       v         v                 v
+------------------------+     +---------+     +---------+     +---------+
|  (Any Color Button)    | --> | Set     |     |  (Center  | --> |  (Valid  |
|                        |     | Color    |     | Detected)|     | Point)  |
|                        |     | Index    |     +---------+     | Draw    |
+------------------------+     +---------+     |         |     | Line    |
                                                v         v     +---------+
                                                +---------+     | Show    |
                                                |         |     | Frames  |
                                                v         v     +---------+
                                                 +---------+         |
                                                 |         |         v
                                                 v         v      +---------+
                                                  +---------+     | Close   |
                                                  |         |     | Windows |
                                                  v         v     +---------+
                                                   +---------+
                                                   |         |
                                                   v         v
                                                +---------+
                                                |  End    |
                                                +---------+
```

Figure 9.1: flowchart of the logic code

Chapter 10

OUTPUT OF PROJECT

Output of the Project (Air Canvas)

1. Real-Time Drawing: The primary output of the Air Canvas project is the ability to draw in real-time using a colored marker in front of a webcam. The marker's movement is tracked and converted into digital ink on a virtual canvas.

2. Color Selection: Users can select different colors for drawing by pressing virtual buttons displayed on the screen. This allows for a multi-colored drawing experience.

3. Clear Canvas: The project includes a functionality to clear the entire canvas, allowing users to start fresh without restarting the application.

4. Visual Feedback: The application provides visual feedback by showing the original webcam feed, the mask used for detecting the marker, and the resulting drawing canvas. This helps users to see the detection process and the final output simultaneously.

5. User Interaction: The application is interactive, responding to user inputs such as color selection and clearing the canvas in real-time. This interaction is seamless and enhances the user experience.

6. Contour Detection: The output also includes the detection and drawing of contours around the marker, which helps in accurately tracking the marker's position.

7. Line Smoothing: The drawn lines are smoothed out using the detected points, resulting in a more natural drawing experience without jittery lines.

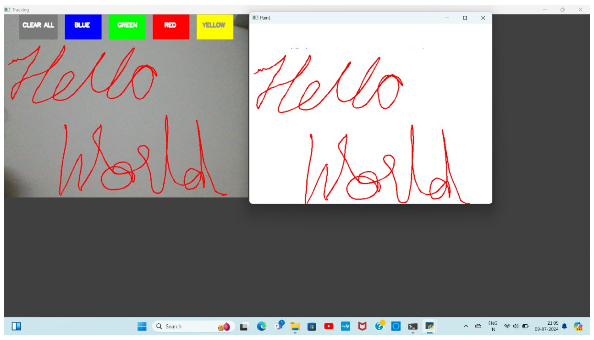


Figure 10.1: ”Hello World”

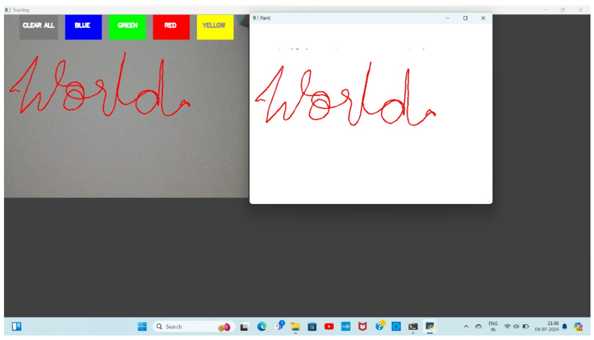


Figure 10.2: ”world”

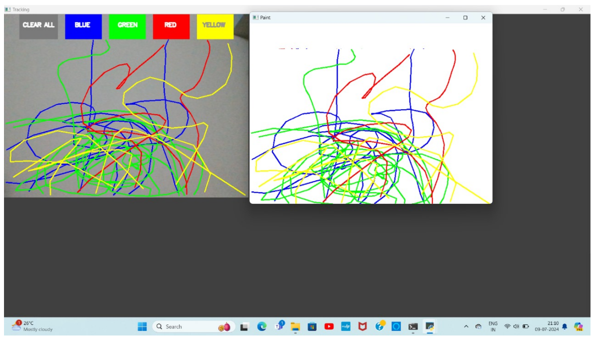


Figure 10.3: Usage of all colours

Chapter 11

IMPLEMENTATION OF PROJECT

Implementation of Air Canvas

1. Import Libraries: The implementation begins by importing the necessary libraries, such as `numpy`, `cv2` (OpenCV), and `deque` from `collections`.

2. Trackbar Function: A default trackbar callback function is created, which is essential for adjusting the marker's color detection range.

3. Create Trackbars: Trackbars are created for setting the upper and lower HSV values. These trackbars help in fine-tuning the color detection for the marker.

4. Define Color Points: Arrays are defined to handle color points of different colors (blue, green, red, yellow). These arrays store the points which will be used for drawing.

5. Initialize Canvas: A white canvas (paintWindow) is initialized where the drawing will take place.

6. Open Webcam: The default webcam of the PC is accessed for capturing the live video feed.

7. Read and Flip Frame: Frames are read from the webcam, and each frame is flipped horizontally to simulate a mirror view.

8. Convert to HSV: The frames are converted from BGR to HSV color space for easier color detection.

9. Get Trackbar Positions: The current positions of the trackbars are retrieved to get the upper and lower HSV values.

10. Create Mask: A mask is created based on the defined HSV range to detect the marker.

11. Preprocess Mask: The mask is preprocessed using erosion, dilation, and morphological opening to reduce noise.

12. Find Contours: Contours are found in the mask to locate the marker's position.

13. Detect and Draw: If contours are detected, the largest contour is identified, and a circle is drawn around it. The center of the contour is calculated for drawing.

14. Check Button Press: The program checks if any virtual button (clear or color) is pressed based on the detected center's position.

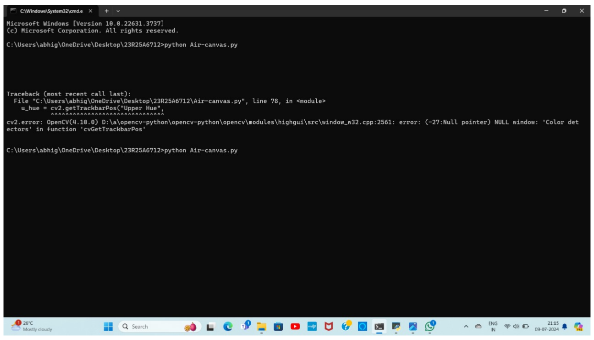
15. Update Points: Based on the detected color and button presses, points are updated in the respective color arrays.

16. Draw on Canvas: Lines are drawn on the canvas and the original frame using the points stored in the color arrays.

17. Show Frames: The original frame, the mask, and the canvas are displayed to the user.

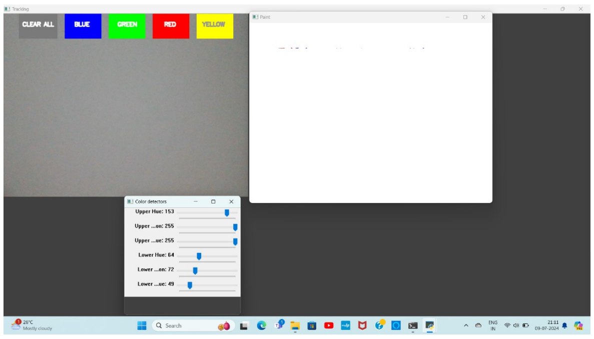
18. Quit Application: The program checks if the 'q' key is pressed to quit the application and release the webcam.

19. Release Resources: Finally, the webcam is released and all OpenCV windows are closed. .



```
Microsoft Windows [Version 10.0.22631.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\abhig\OneDrive\Desktop\23R25A6712>python Air-canvas.py




Traceback (most recent call last):
  File "C:\Users\abhig\OneDrive\Desktop\23R25A6712\Air-canvas.py", line 78, in <module>
    u_hue = cv2.getTrackbarPos("Upper Hue",
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
cv2.error: OpenCV(4.10.0) D:\a\opencv-python\opencv-python\opencv\modules\highgui\src\window_w32.cpp:2561: error: (-27:Null pointer) NULL window: 'Color det
ectors' in function 'cvGetTrackbarPos'


C:\Users\abhig\OneDrive\Desktop\23R25A6712>python Air-canvas.py
```

Figure 11.1: Command prompt



Upper Hue: 153
Upper ...on: 255
Upper ...ue: 255
Lower Hue: 64
Lower ...on: 72
Lower ...ue: 49

Figure 11.2: HSV adjustment

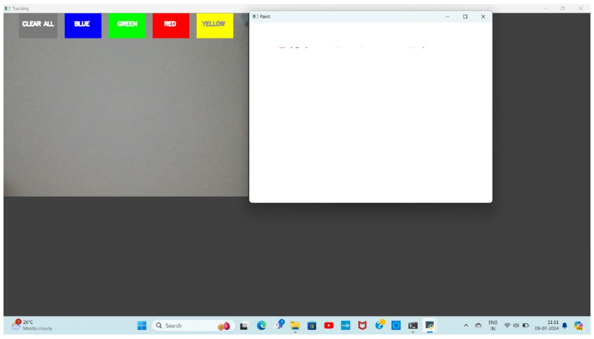


Figure 11.3: air canvasing window

Chapter 12

CONCLUSION

An airwriting approach employing a laptop camera and a video-based pointing mechanism is shown here.. To begin with, the proposed technique tracks the colour of the fingertip in video frames and then applies OCR to the plotted pictures in order to identify the written letters.

It also allows a natural human-system interface that does not need a keypad, pen, or glove for character input. It's all you need is a phone camera and a red hue to reorganize a finger tip. OpenCv and python were used to create an application for the studies. Using the suggested technique, an average accuracy of 92.083 percent may be achieved in the recognition of correct alphabets.

The suggested solution resulted in a 50 ms per character delay in total writing time. Furthermore, the suggested approach may be used to all unconnected languages, but it has one important disadvantage that it is colour sensitive in such a manner that the presence of any red colour in the backdrop before commencing the analysis might lead to erroneous findings.

12.1 FUTURE SCOPE

Computer Vision is the science of helping computers perceive and interpret digital pictures such as photos and movies. It's been a decades-long topic of intense investigation. Computer vision is getting better than the human visual cognitive system at spotting patterns from pictures. Computer vision-based technologies have surpassed human doctors' pattern recognition skills in the healthcare industry.

Let us examine the status of computer vision technology now and in the future.There are several aspects to consider when computer vision expands its effect on the human world. With further study and fine-tuning, computer vision will be able to do more in the future. The system will be simpler to train and can identify more from photos than it does presently.Computer vision will be used in conjunction with other technologies or subsets of AI to generate more attractive applications. Image captioning apps, for example, may use natural language generation (NLG) to understand things in the environment for visually impaired persons. Computer vision may help create artificial general intelligence (AGI) and artificial superintelligence (ASI) by processing information better than the human visual system. Computer vision is a growing sector linked to virtual and augmented reality (VR and AR). Recent market participants have shown a great interest in VR/AR fusion. This significant growth in attention is mirrored in the release of several cutting-edge technology items

Bibliography

[1] G. Bradski, A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, 2008.

[2] T. E. Oliphant, *A Guide to NumPy*, Trelgol Publishing, USA, 2006.

[3] Python Software Foundation, *Python Language Reference, version 3.8*, Available at http://www.python.org

[4] OpenCV, *Open Source Computer Vision Library*, Available at https://opencv.org/

[5] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

[6] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, Prentice Hall, 2007.

[7] Raymond Hettinger, *An introduction to deque*, Available at https://docs.python.org/3/library/collections.htmlcollections.deque