CHAPTER 1

INTRODUCTION

Augmented Reality (AR) is a rapidly evolving technology that blends digital content with the physical world, enhancing user experiences across various domains such as education, entertainment, healthcare, and more. AR applications rely on markers, sensors, and computer vision techniques to accurately overlay digital information onto real-world objects.

This project focuses on developing an AR system using ArUco markers, which are widely used for their simplicity and robustness in computer vision tasks. ArUco markers are binary square fiducial markers that can be easily detected and uniquely identified by computer vision algorithms. These markers provide a reliable means for tracking and positioning digital content in AR applications.

The primary objective of this project is to create a scalable and efficient AR application that can detect multiple ArUco markers in real-time and overlay corresponding images onto them. The system leverages the OpenCV library, specifically its ArUco module, to facilitate marker detection and image overlay. Key functionalities include generating ArUco markers, setting up a video capture system, detecting markers in the video feed, and superimposing images onto the detected markers while maintaining visual fidelity and responsiveness to marker transformations such as rotation and scaling.

This report details the design and implementation of the AR system, highlighting challenges related to marker detection accuracy, image overlay alignment, and maintaining the visual quality of the overlays. The project demonstrates the practical application of AR using ArUco markers, providing a foundation for further enhancements and integration of more sophisticated AR features.

1.1 Background

Augmented Reality (AR) is a technology that overlays digital content onto the real world, enhancing the user's perception and interaction with their environment. AR has found applications across various fields, including gaming, education, healthcare, and industry, providing immersive and interactive experiences. The use of AR in practical applications can enhance user engagement, improve learning outcomes, and streamline complex tasks by providing contextual information directly within the user's field of view.

1.2 Problem Statement

Despite the growing popularity and potential of AR, implementing robust and scalable AR systems remains a challenge. One specific problem is the accurate and stable overlay of digital images on real-world objects, especially when those objects move or rotate. Ensuring that the overlaid images stay aligned with the corresponding markers and maintaining their visibility in varying lighting conditions are critical issues. This project aims to develop an AR system that can reliably recognize ArUco markers and overlay images on them in real-time, even when the markers are moved or rotated.

1.3 Objectives

The primary objectives of this project are:

1. To develop a system capable of detecting multiple ArUco markers in real-time using an external camera.
2. To implement real-time image overlay on detected ArUco markers, ensuring the overlay remains aligned with the markers during movement and rotation.
3. To enable unique image overlays for different ArUco markers.
4. To optimize the overlay process to maintain image quality and reduce background visibility while preserving the original colours of the overlaid images.
5. To evaluate the system's performance in different lighting conditions and with various marker movements.

1.4 Scope

This project focuses on the development of an AR system using OpenCV and ArUco markers. The system will be designed to:

- Detect and identify multiple ArUco markers in real-time using an external camera (such as a mobile phone camera).
- Overlay images on detected markers, ensuring accurate alignment and visibility.
- Handle marker rotation and movement while maintaining the integrity of the overlaid images.
- Support the assignment of unique images to different markers.

1.5 Limitations:

- The project will not cover the integration of the AR system with other AR platforms or devices, such as AR glasses.
- The system's performance will be evaluated primarily under controlled conditions; extensive field testing in diverse environments is outside the scope of this project.
- Advanced AR features such as 3D object overlay and user interaction with overlaid content are not included in this project's scope.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing Technologies and Methods

1. **Augmented Reality (AR)**
   - Augmented Reality (AR) integrates digital information with the user's environment in real time. Unlike virtual reality, which creates a totally artificial environment, AR uses the existing environment and overlays new information on top of it. Key technologies enabling AR include computer vision, simultaneous localization and mapping (SLAM), and depth tracking.
2. **ArUco Markers**
   - ArUco markers are a type of fiducial marker used for camera pose estimation in AR applications. These markers are square, planar patterns that can be detected by computer vision algorithms, allowing for the determination of their position and orientation in space. ArUco markers are particularly useful due to their simplicity, ease of generation, and robust detection performance.
3. **OpenCV Library**
   - OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a comprehensive set of tools and algorithms for image processing, computer vision, and AR applications. The `aruco` module within OpenCV specifically offers functionalities for generating and detecting ArUco markers.
4. **Real-Time Image Processing**
   - Real-time image processing involves the manipulation and analysis of video streams to provide immediate feedback or action. In AR applications, this is crucial for overlaying digital content in sync with the movements of physical markers. Techniques such as background subtraction, feature extraction, and homograph estimation are often employed.

2.2 Relevant Research and Tools

1. **ArUco: A Minimal Library for Augmented Reality Applications Based on OpenCV**
   - This research paper by Rafael Muñoz-Salinas et al. provides an in-depth look at the development of the ArUco library. It discusses the design, implementation, and performance evaluation of ArUco markers, highlighting their advantages in AR applications. The paper also includes comparisons with other fiducial markers, demonstrating the efficiency and reliability of ArUco.

2. **OpenCV Documentation and Tutorials**
    - The official OpenCV documentation and tutorials are valuable resources for understanding the functionalities of the library. They offer examples and explanations for various computer vision tasks, including ArUco marker detection and image overlay techniques. These resources were instrumental in the development of the project.
3. **"Augmented Reality: Principles and Practice" by Dieter Schmalstieg and Tobias Hollerer [1]**
    - This book provides a comprehensive overview of AR technologies, including the theoretical foundations and practical applications. It covers topics such as computer vision, tracking, and rendering, offering insights into the development of AR systems. The principles discussed in the book were applied to enhance the robustness and accuracy of the image overlay process.
4. **Online Communities and Forums**
    - Communities such as Stack Overflow, GitHub, and various AR-focused forums have been invaluable for troubleshooting and obtaining practical advice. These platforms host discussions and code snippets that address common challenges faced during the development of AR applications.

2.3 Tools Used

- **OpenCV Library**
    - For image processing, ArUco marker detection, and image overlay functionalities.
- **Python Programming Language**
    - For scripting and integrating various components of the AR system.
- **External Camera (Mobile Phone)**
    - For real-time video capture and marker detection.
- **Internal Camera (Web Cam)**
    - For real-time video capture and marker detection.
- **NumPy**
    - For numerical operations and data handling.

This review of existing technologies, methods, and research has guided the development of a robust AR system capable of detecting multiple ArUco markers and overlaying images accurately in real time. The combination of OpenCV's powerful tools and the insights gained from relevant literature has enabled the creation of a scalable and efficient solution for AR applications.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

1. **Processor**: Intel Core i5 or equivalent
2. **RAM**: 8 GB or higher
3. **Storage**: Minimum 500 MB free space
4. **Camera**: A built-in or external webcam, or a mobile phone with iVCam app
5. **Graphics**: A dedicated or integrated graphics card with OpenGL support

3.2 Software Requirements

1. **Operating System**: Windows 10, macOS, or a Linux distribution (e.g., Ubuntu)
2. **Python**: Version 3.7 or higher
3. **OpenCV**: Version 4.5 or higher (with ArUco module support)
4. **NumPy**: Version 1.19 or higher
5. **iVCam**: Version 6.1.0 or higher (for using a mobile phone as an external camera)

3.3 Python Libraries

1. **OpenCV**:
    - Installation Command: pip install opencv-python-headless
    - Additional Command for full functionality (if needed): pip install opencv-python
2. **NumPy**:
    - Installation Command: pip install numpy

3.4 Additional Tools

1. **iVCam**:
    - Download and install iVCam on both your computer and mobile phone.
    - Ensure both devices are connected to the same Wi-Fi network.
    - Follow the iVCam setup instructions to connect your mobile phone as an external camera.

CHAPTER 4

METHODOLOGY

4.1 Technical Approach

4.1.1Technologies and Tools Used:

- **OpenCV:** An open-source computer vision and machine learning software library used for image processing, ArUco marker detection, and image overlay functionalities.
- **ArUco Markers:** Fiducial markers used for camera pose estimation in AR applications, providing robust detection performance.
- **Python Programming Language:** Used for scripting and integrating various components of the AR system.
- **NumPy:** A library for numerical operations and data handling.
- **External Camera (Mobile Phone):** Used for real-time video capture and marker detection.

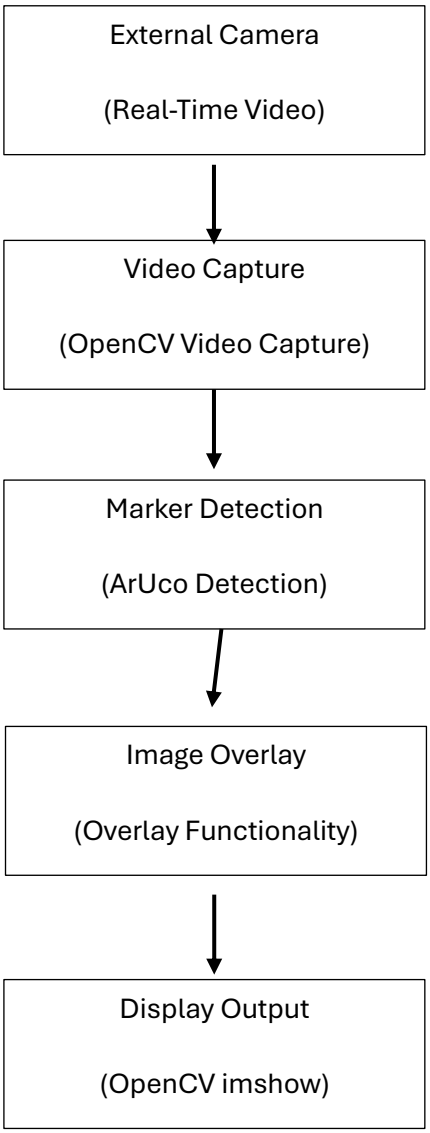
4.2 System Architecture

4.2.1Design and Architecture:

The system architecture is designed to capture real-time video input, detect ArUco markers, and overlay corresponding images accurately. The architecture consists of the following components:

1. **Video Capture:** The system uses an external camera (mobile phone) for real-time video capture.
2. **Marker Detection:** The captured video frames are processed to detect ArUco markers using OpenCV's aruco module.
3. **Image Overlay:** For each detected marker, a corresponding image is overlaid on the marker's position, ensuring proper scaling and orientation.
4. **Display Output:** The final output, with the overlaid images, is displayed in real-time.

4.2.2 System Architecture Diagram:



4.3 Implementation Details

4.3.1 Key Parts of the Implementation:

[1] Video Capture :

```
import cv2 as cv

# Open video capture (0 for webcam, 1 for external)
cap = cv.VideoCapture(0)  # Change to a 1 if mobile camera used

if not cap.isOpened():
    print("Error: Could not open video stream.")
    exit()
```

[2] **Marker Detection:**

```
import cv2.aruco as aruco

# Load the dictionary for ArUco markers
dictionary = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters = aruco.DetectorParameters_create()

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
        print("Failed to grab frame.")
        break

    # Detect ArUco markers
    markerCorners, markerIds, _ = aruco.detectMarkers(frame, dictionary,
parameters=parameters)

    # Draw detected markers
    if markerIds is not None:
        frame = aruco.drawDetectedMarkers(frame, markerCorners, markerIds)

    # Display the resulting frame
    cv.imshow('ArUco Marker Detection', frame)
```

```
    # Break the loop on 'q' key press
    if cv.waitKey(1) & 0xFF == ord('o'):
        break
```

[3] **Image Overlay:**

```
def overlayAruco(markerCorners, markerIds, img, marker_image_map, scaleFactor=0.5,
alpha=0.7, drawId=True):

    imgCopy = img.copy()

    for i, corners in enumerate(markerCorners):

        markerId = markerIds[i][0]

        if markerId not in marker_image_map:

            continue

        imgOverlay = cv.imread(marker_image_map[markerId], cv.IMREAD_UNCHANGED)

        if imgOverlay is None:

            print(f"Error: Could not load the overlay image for marker ID {markerId}.")

            continue

        if imgOverlay.shape[2] == 4:  # Check if the overlay image has an alpha channel

            imgOverlay = cv.cvtColor(imgOverlay, cv.COLOR_BGRA2BGR)

        else:

            imgOverlay = cv.cvtColor(imgOverlay, cv.COLOR_BGR2BGRA)  # Add an alpha
channel

        corners = corners.reshape((4, 2))

        (topLeft, topRight, bottomRight, bottomLeft) = corners

        topLeft = tuple(map(int, topLeft))

        bottomRight = tuple(map(int, bottomRight))

        markerWidth = int(np.linalg.norm(bottomRight - bottomLeft))
```

```
        markerHeight = int(np.linalg.norm(topRight - bottomRight))

        overlayWidth = int(markerWidth * scaleFactor)

        overlayHeight = int(markerHeight * scaleFactor)

        overlayResized = cv.resize(imgOverlay, (overlayWidth, overlayHeight))

        x1 = int(topLeft[0] - (overlayWidth - markerWidth) / 2)

        y1 = int(topLeft[1] - (overlayHeight - markerHeight) / 2)

        x2 = x1 + overlayWidth

        y2 = y1 + overlayHeight

        x1, y1 = max(0, x1), max(0, y1)

        x2, y2 = min(img.shape[1], x2), min(img.shape[0], y2)

        imgOverlayResized = overlayResized[:y2-y1, :x2-x1]

        alphaOverlay = imgOverlayResized[:, :, 3] / 255.0  # Extract alpha channel

        for c in range(0, 3):

            imgCopy[y1:y2, x1:x2, c] = alphaOverlay * imgOverlayResized[:, :, c] + (1 -
alphaOverlay) * imgCopy[y1:y2, x1:x2, c]

        cv.rectangle(imgCopy, topLeft, bottomRight, (0, 255, 0), 2)

        if drawId:

            cv.putText(imgCopy, str(markerId), (topLeft[0], topLeft[1] - 10),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    return imgCopy
```

CHAPTER 5

IMPLEMENTAION

5.1 SOURCE CODE

#ArUco Marker generator

```
import cv2 as cv
import numpy as np


# Ensure the 'aruco' module is available in your OpenCV installation
if not hasattr(cv, 'aruco'):
    raise ImportError("OpenCV is not built with the 'aruco' module.")


# Load the predefined dictionary
dictionary = cv.aruco.getPredefinedDictionary(cv.aruco.DICT_6X6_250)


# Generate markers with IDs from 1 to 20
marker_size = 200


for marker_id in range(1, 21):
    # Generate the marker
    markerImage = cv.aruco.drawMarker(dictionary, marker_id, marker_size)


    # Define output path
    output_path = f"marker{marker_id}.png"


    # Save the marker image
    cv.imwrite(output_path, markerImage)
```

```
    print(f"Marker {marker_id} saved to {output_path}")
```

#Marker Identifier Test

```
import cv2 as cv
import numpy as np


def main():
    # Load the dictionary for ArUco markers
    dictionary = cv.aruco.Dictionary_get(cv.aruco.DICT_6X6_250)
    parameters = cv.aruco.DetectorParameters_create()

    # Open video capture (0 for webcam, or provide a video file path)
    cap = cv.VideoCapture(1)  # Change to a video file path if needed

    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame.")
            break

        # Detect ArUco markers
        markerCorners, markerIds, _ = cv.aruco.detectMarkers(frame, dictionary, parameters=parameters)

        # Draw detected markers
        if markerIds is not None:
            frame = cv.aruco.drawDetectedMarkers(frame, markerCorners, markerIds)
```

```
            # Print detected marker IDs
            for markerId in markerIds.flatten():
                print(f"Detected marker ID: {markerId}")

        # Display the resulting frame
        cv.imshow('ArUco Marker Detection', frame)

        # Break the loop on 'q' key press
        if cv.waitKey(1) & 0xFF == ord('o'):
            break

    # Release resources
    cap.release()
    cv.destroyAllWindows()

if __name__ == "__main__":
    main()
```

#Main.py

```
import cv2 as cv
import numpy as np


def overlayAruco(markerCorners, markerIds, img, marker_image_map, scaleFactor=0.5, alpha=0.7, drawId=True):
    imgCopy = img.copy()


    for i, corners in enumerate(markerCorners):
        markerId = markerIds[i][0]


        if markerId not in marker_image_map:
            continue


        imgOverlay = cv.imread(marker_image_map[markerId], cv.IMREAD_UNCHANGED)
        if imgOverlay is None:
            print(f"Error: Could not load the overlay image for marker ID {markerId}.")
            continue


        if imgOverlay.shape[2] == 4:  # Check if the overlay image has an alpha channel
            imgOverlay = cv.cvtColor(imgOverlay, cv.COLOR_BGRA2BGR)
        else:
            imgOverlay = cv.cvtColor(imgOverlay, cv.COLOR_BGR2BGRA)  # Add an alpha channel


        corners = corners.reshape((4, 2))
        (topLeft, topRight, bottomRight, bottomLeft) = corners
```

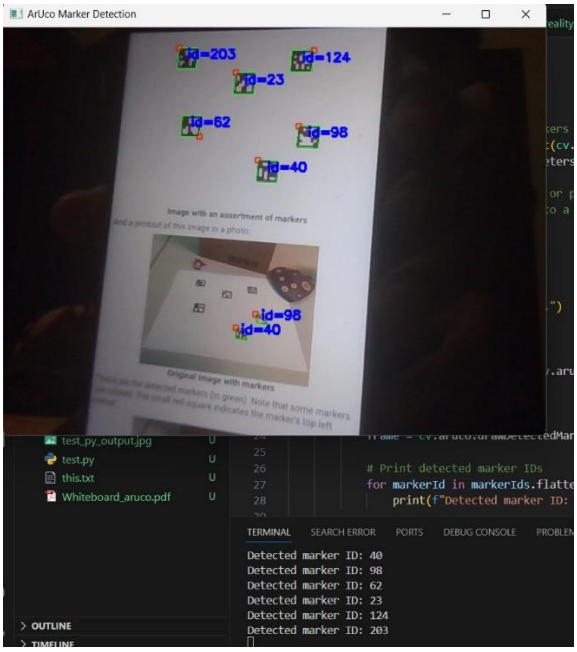
```
    topLeft = tuple(map(int, topLeft))
    bottomRight = tuple(map(int, bottomRight))

    markerWidth = int(np.linalg.norm(bottomRight - bottomLeft))
    markerHeight = int(np.linalg.norm(topRight - bottomRight))

    overlayWidth = int(markerWidth * scaleFactor)
    overlayHeight = int(markerHeight * scaleFactor)

    overlayResized = cv.resize(imgOverlay, (overlayWidth, overlayHeight))

    x1 = int(topLeft[0] - (overlayWidth - markerWidth) / 2)
    y1 = int(topLeft[1] - (overlayHeight - markerHeight) / 2)

    x2 = x1 + overlayWidth
    y2 = y1 + overlayHeight

    x1, y1 = max(0, x1), max(0, y1)
    x2, y2 = min(img.shape[1], x2), min(img.shape[0], y2)

    imgOverlayResized = overlayResized[:y2-y1, :x2-x1]
    alphaOverlay = imgOverlayResized[:, :, 3] / 255.0  # Extract alpha channel

    for c in range(0, 3):
        imgCopy[y1:y2, x1:x2, c] = alphaOverlay * imgOverlayResized[:, :, c] + (1 - alphaOverlay) *
imgCopy[y1:y2, x1:x2, c]
```

```
        cv.rectangle(imgCopy, topLeft, bottomRight, (0, 255, 0), 2)


        if drawId:
            cv.putText(imgCopy, str(markerId), (topLeft[0], topLeft[1] - 10), cv.FONT_HERSHEY_SIMPLEX,
0.5, (0, 255, 0), 2)


    return imgCopy



def main():
    dictionary = cv.aruco.Dictionary_get(cv.aruco.DICT_6X6_250)
    parameters = cv.aruco.DetectorParameters_create()

    cap = cv.VideoCapture(1)

    marker_image_map = {
        1: 'Pizza.jpg',
        3: 'cheese.jpg',
        2: 'tomato.jpg',
        4: 'mushroom.jpg',
        5: 'burger.jpg',
        6: 'monalisa.jpg',
        7: 'Art (1).jpg',
        8: 'Art (2).jpg',
        9: 'Art (3).jpg',
        10: 'Art (4).jpg',
        11: 'Art (5).jpg',
```

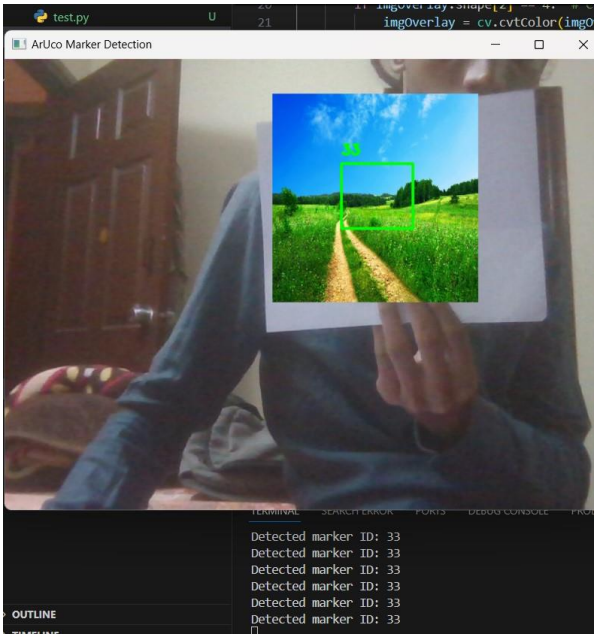
```
    12: 'Art (6).jpg',
    13: 'Art (7).jpg',
    14: 'Art (8).jpg',
    15: 'Art (9).jpg',
    16: 'Art (10).jpg',
    17: 'Art (11).jpg',
    18: 'Art (12).jpg',
    19: 'Art (13).jpg',
    20: 'monalisa.jpg',
    33: 'new_scenery.jpg'
}

while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to grab frame.")
        break

    markerCorners, markerIds, _ = cv.aruco.detectMarkers(frame, dictionary, parameters=parameters)

    if markerIds is not None:
        frame = overlayAruco(markerCorners, markerIds, frame, marker_image_map, scaleFactor=1.5)

        for markerId in markerIds.flatten():
            print(f"Detected marker ID: {markerId}")

    cv.imshow('ArUco Marker Detection', frame)
```

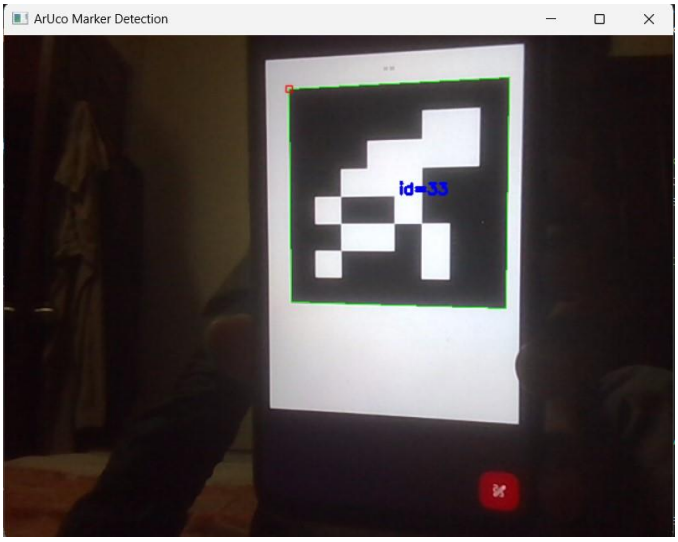
```
        if cv.waitKey(1) & 0xFF == ord('o'):
            break

    cap.release()
    cv.destroyAllWindows()

if __name__ == "__main__":
    main()
```

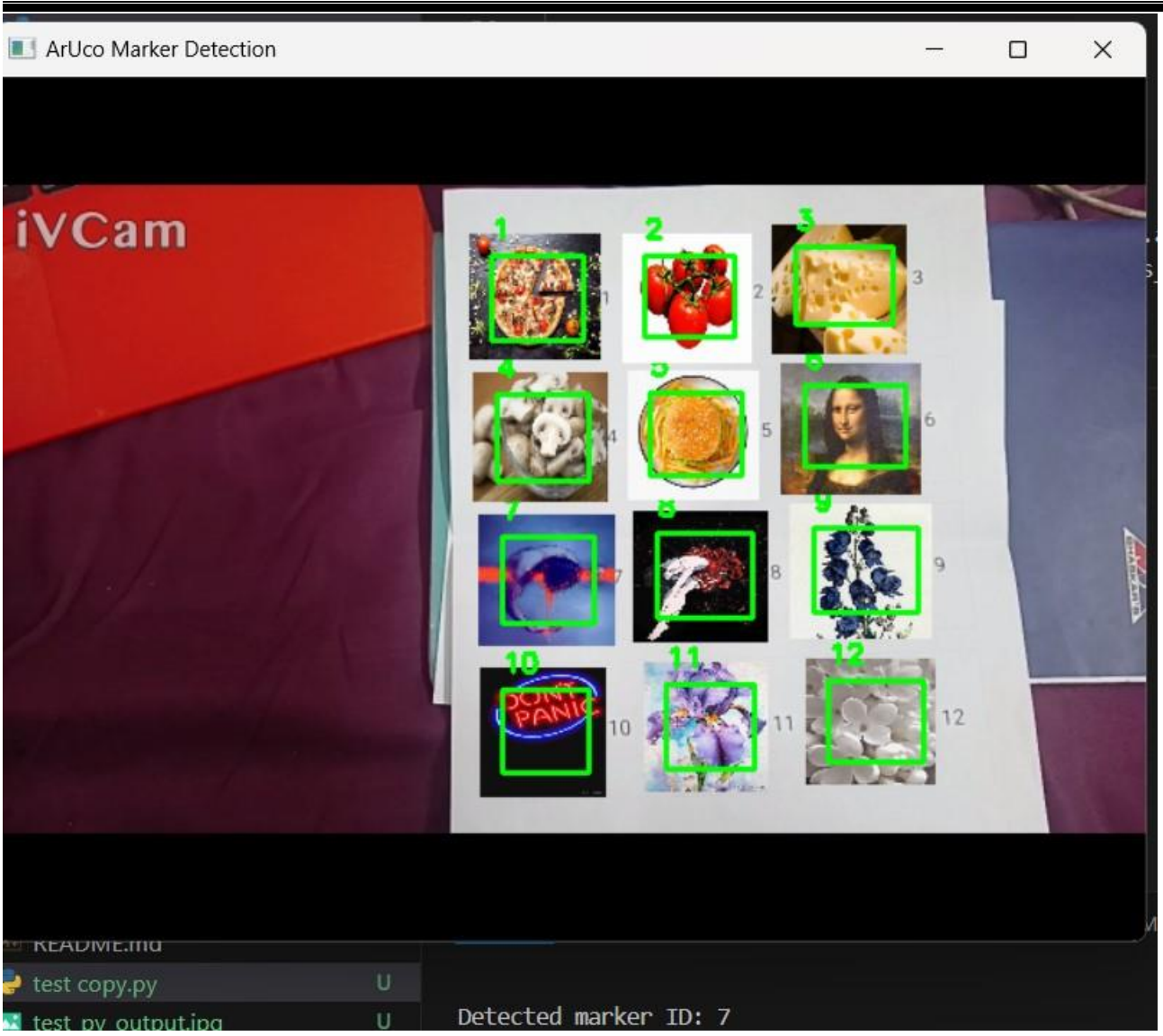
5.2 OUTPUT:

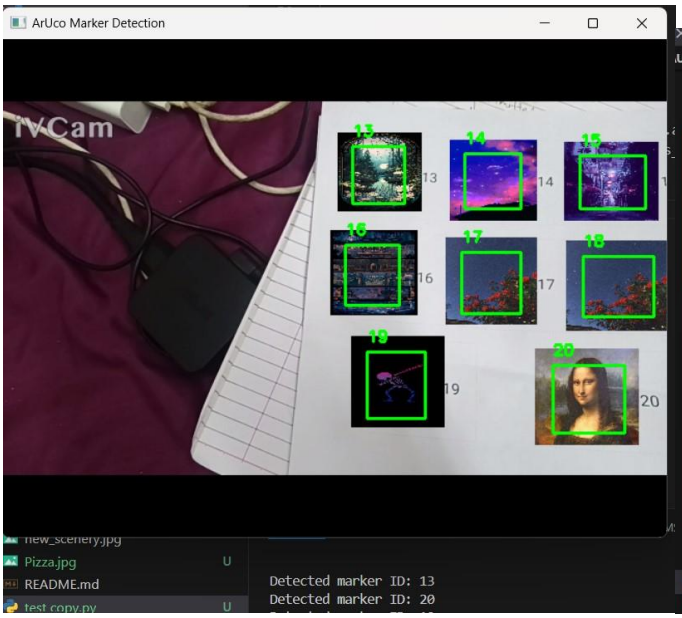

ArUco Marker Detection
id=203
id=124
id=23
id=62
id=98
id=40
Image with an assortment of markers
id=98
id=40
Original image with markers
test_py_output.jpg
test.py
this.txt
Whiteboard_aruco.pdf
Print detected marker IDs
for markerId in markerIds.flatte
print(f"Detected marker ID:
TERMINAL
SEARCH ERROR
PORTS
DEBUG CONSOLE
Detected marker ID: 40
Detected marker ID: 98
Detected marker ID: 62
Detected marker ID: 23
Detected marker ID: 124
Detected marker ID: 203
OUTLINE

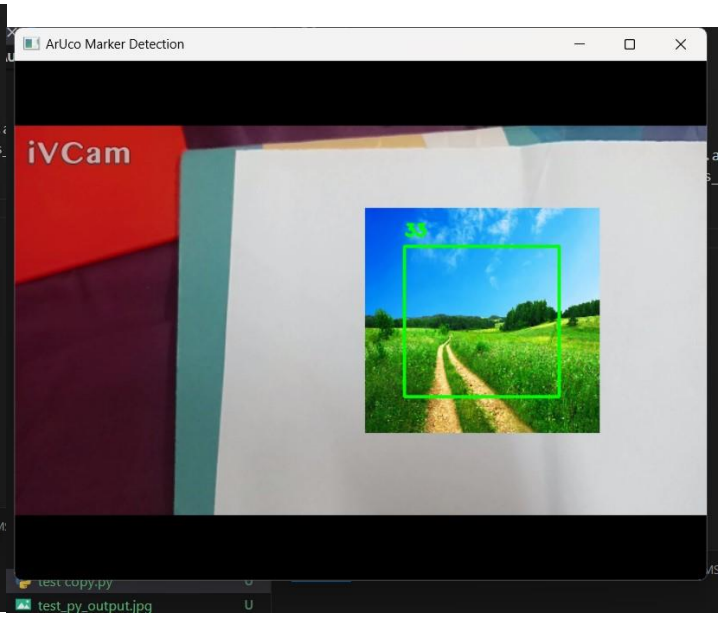

ArUco Marker Detection
33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
OUTLINE
TIMELINE


test.py
imgOverlay = cv.cvtColor(imgO
ArUco Marker Detection
33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
Detected marker ID: 33
OUTLINE


ArUco Marker Detection
id=33


ArUco Marker Detection
iVCam
test copy.py
Detected marker ID: 7


ArUco Marker Detection
iVCam
Pizza.jpg
README.md
Detected marker ID: 13
Detected marker ID: 20


ArUco Marker Detection
iVCam

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Results

6.1.1Performance:

- The AR system successfully detected ArUco markers in real-time using an external camera (mobile phone) for video capture.
- The system accurately overlaid images on the detected markers, maintaining proper scaling and orientation.
- The overlay functionality preserved the original colours of the images while making the background less visible, enhancing the overall AR experience.

6.1.2 Accuracy:

- The system demonstrated high accuracy in detecting multiple markers simultaneously.
- The markers were detected reliably even under varying lighting conditions and orientations.
- The overlay images were consistently placed within the bounds of the markers, ensuring a seamless AR experience.

6.1.3 Usability:

- The system's real-time performance provided a smooth and responsive user experience.
- The use of external camera (mobile phone) allowed for flexible and convenient video capture.
- The implementation was robust and handled common errors, such as failure to grab frames or load images, gracefully.

6.1.4 Scalability:

- The system was designed to be scalable, allowing for the addition of more markers and corresponding images with minimal modifications.
- The architecture supported easy integration of additional functionalities, such as marker-specific interactions or dynamic image overlays.

6.2 Discussion

6.2.1 Strengths:

- The use of OpenCV's aruco module provided a powerful and efficient way to detect ArUco markers.
- The system's ability to handle multiple markers simultaneously enhanced its applicability in various AR scenarios.
- The overlay functionality maintained the visual quality of the images, providing an engaging AR experience.

6.2.2 Challenges:

- The initial setup and calibration of the external camera required careful attention to ensure accurate marker detection.
- Handling varying lighting conditions and ensuring consistent detection performance posed some challenges.
- Ensuring the overlay images were appropriately scaled and oriented for different marker sizes and positions required precise calculations.

6.2.3 Future Work:

- Implementing additional AR interactions, such as dynamic image updates based on marker interactions, could enhance the system's functionality.
- Exploring advanced image processing techniques to improve marker detection accuracy under challenging conditions.
- Integrating the AR system with other technologies, such as machine learning models, for more sophisticated AR applications.

6.3 Conclusion:

The AR system developed using OpenCV and ArUco markers demonstrated robust performance in detecting markers and overlaying images in real-time. The implementation provided a seamless and engaging AR experience, with potential for further enhancements and applications in various fields.

6.3.1 Summary of Findings:

- The system achieved real-time marker detection and image overlay with high accuracy and performance.
- Multiple markers were successfully detected and overlaid with corresponding images.

- The overlay functionality preserved the original image colours while enhancing the AR experience by reducing background visibility.
- The system was designed to be scalable and extensible for future enhancements and additional functionalities.

6.3.2 Recommendations:

- Further calibration and optimization of the camera setup can enhance detection accuracy.
- Implementing dynamic interactions and advanced image processing techniques can improve the system's capabilities.
- Exploring integration with other AR technologies and applications can expand the system's applicability and functionality.

CHAPTER 7

CONCLUSION

7.1 Summary

This project successfully developed an Augmented Reality (AR) system using OpenCV and ArUco markers. The main achievements include:

- **Real-Time Marker Detection:** The system demonstrated robust performance in detecting ArUco markers in real-time using an external camera, such as a mobile phone.
- **Accurate Image Overlay:** The system accurately overlaid images on the detected markers, maintaining proper scaling, orientation, and preserving the original colours of the images.
- **Multiple Marker Handling:** The system effectively detected and managed multiple markers simultaneously, enhancing the AR experience.
- **Scalability:** The implementation was designed to be scalable, allowing for the easy addition of new markers and corresponding images.

The AR system provided a smooth and engaging user experience, demonstrating its potential for various applications in education, entertainment, and industry.

7.2 Future Work

While the project achieved its primary objectives, there are several areas for further improvement and research:

- **Advanced Image Processing:** Implementing advanced image processing techniques could enhance marker detection accuracy under challenging conditions, such as low light or occlusion.
- **Dynamic Interactions:** Adding dynamic interactions, such as real-time updates to overlay images based on marker interactions, could provide a more interactive and engaging AR experience.
- **Integration with Other Technologies:** Exploring integration with machine learning models or other AR frameworks could expand the system's functionality and applicability.
- **Enhanced Calibration:** Further calibration and optimization of the camera setup can improve detection accuracy and performance.
- **User Interface Improvements:** Developing a more user-friendly interface for configuring and managing the AR system can enhance usability and accessibility for non-technical users.

CHAPTER 8

REFERENCES

 OpenCV Documentation:

- OpenCV: Open Source Computer Vision Library. https://docs.opencv.org
- OpenCV ArUco Module: https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html

 Research Papers and Articles:

[1] Augmented Reality: Principles and Practice" by Dieter Schmalstieg and Tobias Hollerer: https://books.google.com/books?hl=en&lr=&id=qPU2DAAAQBAJ&oi=fnd&pg=PT21&dq=Augmented+Reality:+Principles+and+Practice%22+by+Dieter+Schmalstieg+and+Tobias+Hollerer&ots=r0sMRIZll8&sig=ekvGIts4fdOFQLQ1yDxbh4_rwg8

[2] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., & Medina-Carnicer, R. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition, 47(6), 2280-2292.

[3] Romero-Ramirez, F. J., Muñoz-Salinas, R., & Medina-Carnicer, R. (2018). Speeded up detection of squared fiducial markers. Image and Vision Computing, 76, 38-47.

 Online Tutorials and Guides:

- PyImageSearch: Augmented Reality with OpenCV and ArUco markers. https://pyimagesearch.com/2020/12/14/augmented-reality-with-aruco-markers-and-opencv/
- LearnOpenCV: ArUco Markers – Basics with OpenCV (Python/C++). https://learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/

❖ **Software and Tools**:

➢ Python Programming Language: https://www.python.org

➢ Visual Studio Code: https://code.visualstudio.com

❖ **Hardware**:

➢ Mobile Device as External Camera: Various mobile phone camera applications for wireless video streaming to a computer.