

TABLE OF CONTENTS

List of Figures.....	1
CHAPTER 1. INTRODUCTION	2
1.1. Introduction to Project.....	2
1.2. Identification of Problem.....	2
CHAPTER 2. BACKGROUND STUDY	3
2.1. Existing solutions	3
2.2. Problem Definition	5
2.3. Goals/Objectives	5
CHAPTER 3. DESIGN FLOW/PROCESS	6
3.1. Evaluation & Selection of Specifications/Features	6
3.2. Analysis of Features and finalization subject to constraints	7
3.3. Design Flow.....	7
CHAPTER 4. RESULTS ANALYSIS AND VALIDATION.....	8
4.1. Implementation of solution.....	8
4.2. Output and Findings	9
CHAPTER 5. CONCLUSION AND FUTURE WORK	10
5.1. Conclusion	10
5.2. Future work.....	11

List of Figures:

Sr. No	Figure Name	Page Number
1	OBS Studio Interface	3
2	Bandicam Interface	4
3	Output of the Project	9
4	Performance of the Project	10

CHAPTER 1. INTRODUCTION

1.1. Introduction to Project

Our project aims to develop a screen recording application using Python, leveraging libraries such as OpenCV, PIL (Pillow), NumPy, and screeninfo. Screen recording applications are essential tools in various fields, including education, software development, gaming, and technical support. We designed this project to create a straightforward yet functional screen recorder that captures screen activity and stores it as a video file. The main focus is to provide a solution that balances ease of use, resource efficiency, and versatility, making it suitable for creating tutorials, recording online lectures, conducting software demos, or troubleshooting issues.

We chose Python due to its popularity, simplicity, and extensive multimedia library support. OpenCV is employed for video manipulation and real-time processing, PIL facilitates the capture of screenshots, and NumPy manages image processing operations. Additionally, screeninfo retrieves screen dimension information, ensuring that the program dynamically adapts to different display settings.

1.2. Identification of Problem

Despite the availability of numerous screen recording tools, we identified several limitations and challenges faced by users:

- **High Resource Consumption:** Many popular screen recording software programs consume significant CPU and memory resources, leading to poor performance on lower-end systems.
- **Licensing and Costs:** Tools such as Camtasia and Bandicam charge for premium features, which may not be affordable for all users.

- **Complexity and Learning Curve:** Advanced screen recording solutions like OBS Studio have a steep learning curve, making them less accessible to non-technical users.
- **Lack of Customization:** Some screen recorders do not offer sufficient options for customizing recording settings, such as frame rate, resolution, or video encoding format.

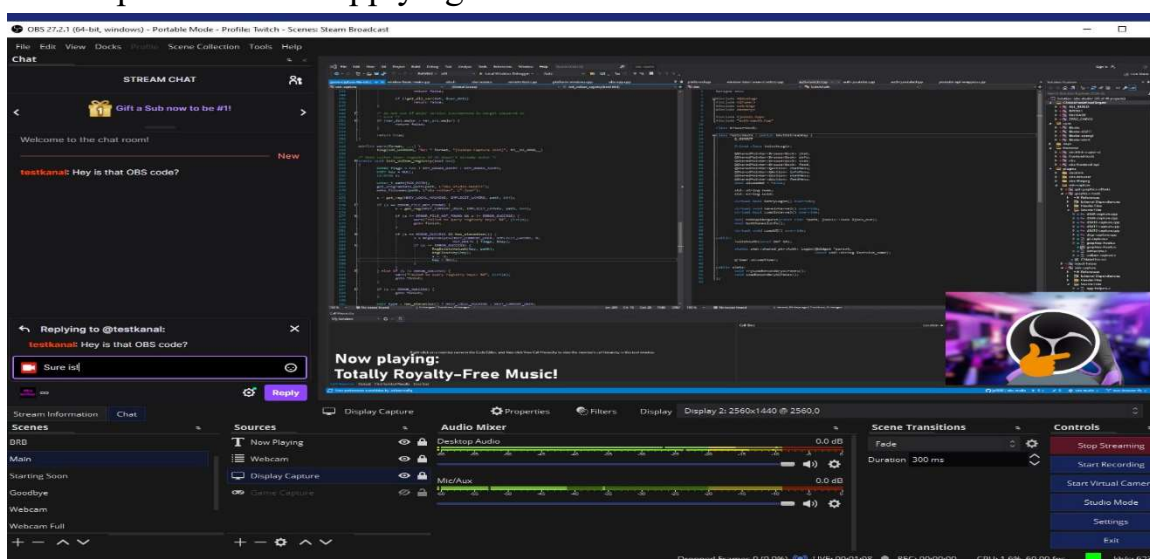
Our project addresses these issues by developing a lightweight, customizable screen recording application that is open-source, accessible, and efficient for everyday use.

CHAPTER 2. BACKGROUND STUDY

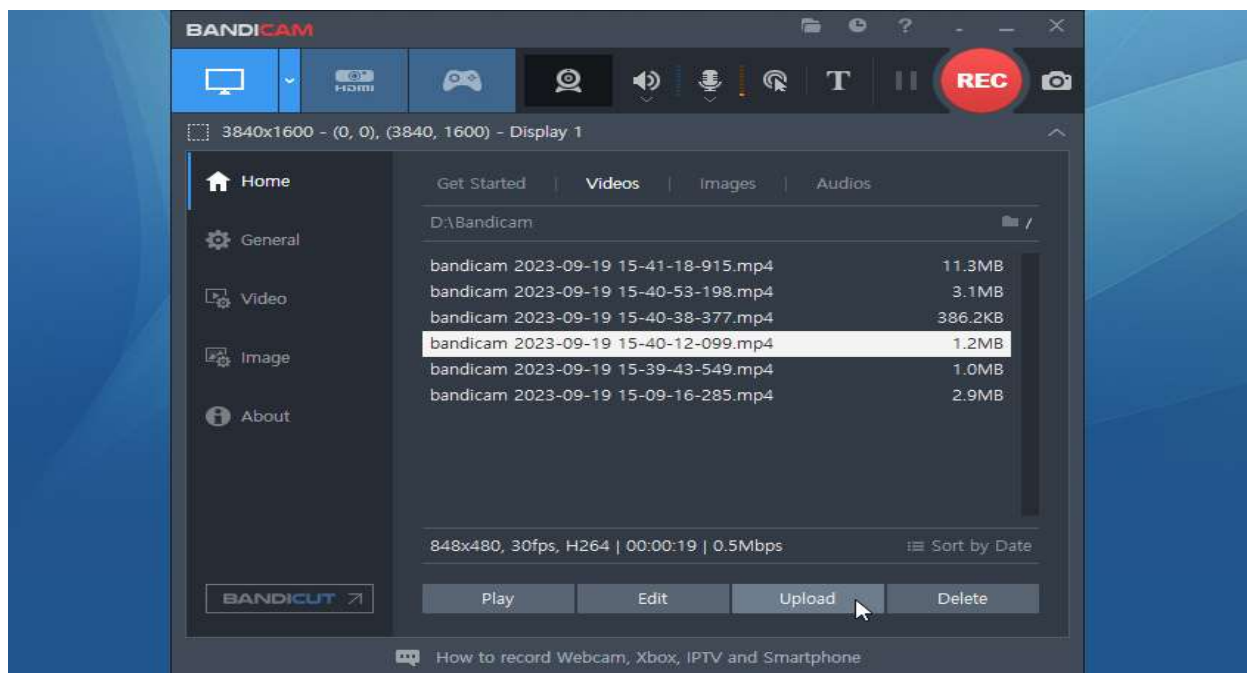
2.1. Existing Solutions

We researched several screen recording tools to understand their strengths and weaknesses:

- **OBS Studio:** An open-source, feature-rich solution widely used for live streaming and recording. OBS allows for custom video and audio configurations, multiple source capture, and streaming to various platforms. However, its complexity can overwhelm users seeking a simple recording solution. Additionally, OBS can be resource-intensive, especially when using multiple sources or applying real-time filters and transitions.



- **Camtasia:** Known for its user-friendly interface and powerful editing capabilities, Camtasia enables seamless recording and editing. It includes advanced features such as cursor effects, callouts, and interactive quizzes. However, Camtasia is a paid software with a relatively high cost, making it less accessible to individuals or small businesses on a budget.
- **Bandicam:** This tool provides efficient screen recording and high compression video quality, making it ideal for recording gameplay. It supports a range of codecs and offers customizable settings for frame rates and bitrates. Despite its advantages, Bandicam's free version has limitations on recording time, and its watermarking of recorded videos can be a drawback for some users.



Bandicam Interface.

- **Free and Open-Source Tools:** Options such as SimpleScreenRecorder and Kazam cater to Linux users, while ShareX offers extensive functionality for Windows. However, some of these tools lack advanced customization options or have limited support for different platforms.

2.2. Problem Definition

The main problem with existing solutions is the trade-off between functionality, cost, and ease of use. Most tools either charge for advanced features, have a steep learning curve, or consume significant system resources. Users seeking a straightforward, efficient, and customizable screen recording solution often struggle to meet their needs without compromising on one or more aspects.

Key Issues:

- High cost of premium screen recorders like Camtasia.
- Resource-heavy operations in tools like OBS Studio, impacting system performance.
- Limited customization options in free or open-source software.
- Challenges in cross-platform compatibility, especially for Linux and macOS users.

2.3. Goals/Objectives

The objectives of our project are:

- To develop a lightweight, customizable screen recorder capable of real-time video capture with minimal resource consumption.
- To provide a free and open-source solution suitable for personal, educational, and professional purposes.
- To enable real-time video processing and support standard video formats (e.g., MP4) for compatibility with most media players.
- To maintain a balance between quality and performance, ensuring that the tool runs smoothly on basic hardware.
- To allow customization of recording parameters such as frame rate and resolution for different user needs.

CHAPTER 3. DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

We carefully evaluated libraries and specifications to achieve our project goals:

- **OpenCV:** Provides extensive functionalities for video manipulation, including reading, writing, and displaying video frames in real-time. It is efficient for handling large arrays of pixel data and supports various codecs.
- **PIL (Pillow):** A Python Imaging Library that facilitates capturing the screen as an image. PIL simplifies operations like image cropping, resizing, and format conversion, which are essential for screen recording tasks.
- **NumPy:** Manages the conversion of image data to arrays, allowing for mathematical operations and efficient data manipulation. This library is crucial for converting captured frames to formats compatible with OpenCV.
- **screeninfo:** Ensures the program automatically adapts to different screen resolutions, making it suitable for a wide range of monitors and display setups.

Considered Features:

- **Frame Rate Adjustment:** Allows users to customize the smoothness of the recorded video. A default frame rate of 5 FPS is selected to minimize CPU usage while providing acceptable visual clarity.
- **Resolution Detection:** Dynamically retrieves the screen resolution to support multiple monitors and various screen sizes.
- **Video Codec Selection:** The MP4 format is selected for its balance between quality and file size. The codec 'mp4v' ensures compatibility across different systems and media players.

3.2. Analysis of Features and Finalization Subject to Constraints

We identified the following main constraints:

- **System Resources:** The application should not consume excessive CPU or memory, particularly on lower-end hardware. Real-time processing must be optimized to maintain smooth performance.
- **User Simplicity vs. Customizability:** While the tool should offer basic settings customization, the interface and usage should remain simple enough for non-technical users. Advanced features should not complicate the core functionality.
- **Compatibility:** The solution should be developed with cross-platform compatibility in mind, even if initial support is focused on Windows systems.

Finalized Features:

- Real-time screen capture with adjustable frame rates.
- Automatic screen resolution detection.
- Output format as MP4, ensuring wide compatibility.
- A minimal interface focused on essential functionalities.

3.3. Design Flow

The design flow follows these steps:

1. **Retrieve Screen Dimensions:** The program uses `screeninfo` to detect connected monitors and their dimensions, setting the appropriate capture area for each screen.
2. **Initialize Video Writer:** The `cv2.VideoWriter` object is set up with the codec, file name, frame rate, and resolution.
3. **Capture Screen Continuously:** The program enters a loop where it captures the screen using `ImageGrab.grab()`, converts the screenshot to a NumPy array, and processes it with OpenCV functions.

4. **Process Frame Data:** The captured image is converted from RGB to BGR for OpenCV compatibility and then displayed in a preview window.
5. **Save Frame to File:** Each processed frame is written to the video file. The program continuously listens for the 'Esc' key, which ends the recording and releases all resources.
6. **Release Resources and Save File:** The VideoWriter object is released, and the output video is saved to disk.

CHAPTER 4. RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of Solution

The screen recording application was implemented using Python, with the necessary libraries installed and configured. A virtual environment was created to manage dependencies and maintain a clean development setup. The application was tested on multiple systems to ensure consistent performance across different hardware configurations.

The code of the project is:

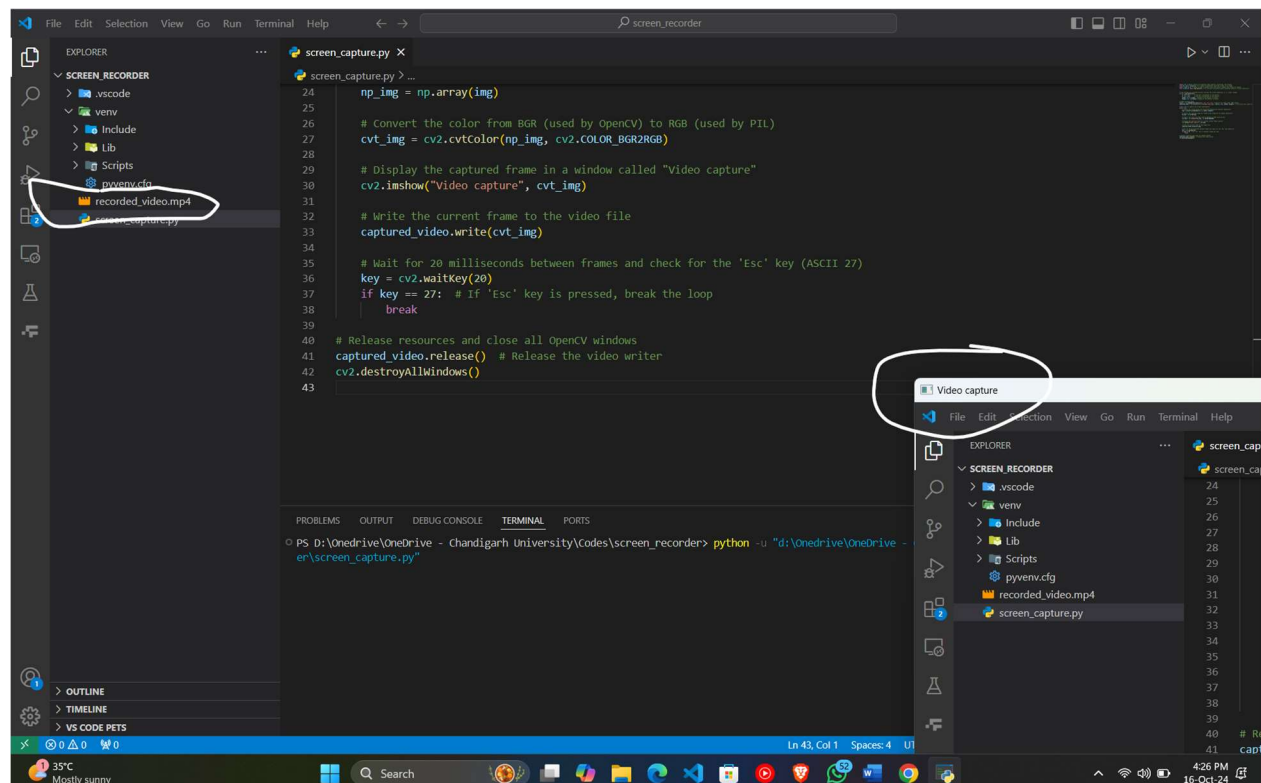
```
import cv2 # OpenCV library for handling video capture, processing, and display
from PIL import ImageGrab # To capture screenshots (grab the screen) as an image
import numpy as np # NumPy is used to handle arrays and convert images into arrays
for OpenCV
from screeninfo import get_monitors # To retrieve information about connected
monitors (screen dimensions)
# Loop through all connected monitors and get the screen dimensions (x, y, width,
height)
for m in get_monitors():
    x: int = m.x # Top-left x-coordinate of the monitor
    y: int = m.y # Top-left y-coordinate of the monitor
    width: int = m.width # Width of the monitor in pixels
    height: int = m.height # Height of the monitor in pixels
# Video encoding setup
fourcc = cv2.VideoWriter_fourcc("m", "p", "4", "v") # Specify the video codec (MP4
format)
```

```

captured_video = cv2.VideoWriter("recorded_video.mp4", fourcc, 5.0, (width,
height)) # Initialize the video writer
# Main loop to capture the screen continuously
while True:
    # Capture the screen within the specified bounding box (monitor dimensions)
    img = ImageGrab.grab(bbox=(x, y, width, height))
    # Convert the captured image to a NumPy array (required for OpenCV operations)
    np_img = np.array(img)
    # Convert the color from BGR (used by OpenCV) to RGB (used by PIL)
    cvt_img = cv2.cvtColor(np_img, cv2.COLOR_BGR2RGB)
    # Display the captured frame in a window called "Video capture"
    cv2.imshow("Video capture", cvt_img)
    # Write the current frame to the video file
    captured_video.write(cvt_img)
    # Wait for 20 milliseconds between frames and check for the 'Esc' key (ASCII
27)
    key = cv2.waitKey(20)
    if key == 27: # If 'Esc' key is pressed, break the loop
        break
# Release resources and close all OpenCV windows
captured_video.release() # Release the video writer
cv2.destroyAllWindows()

```

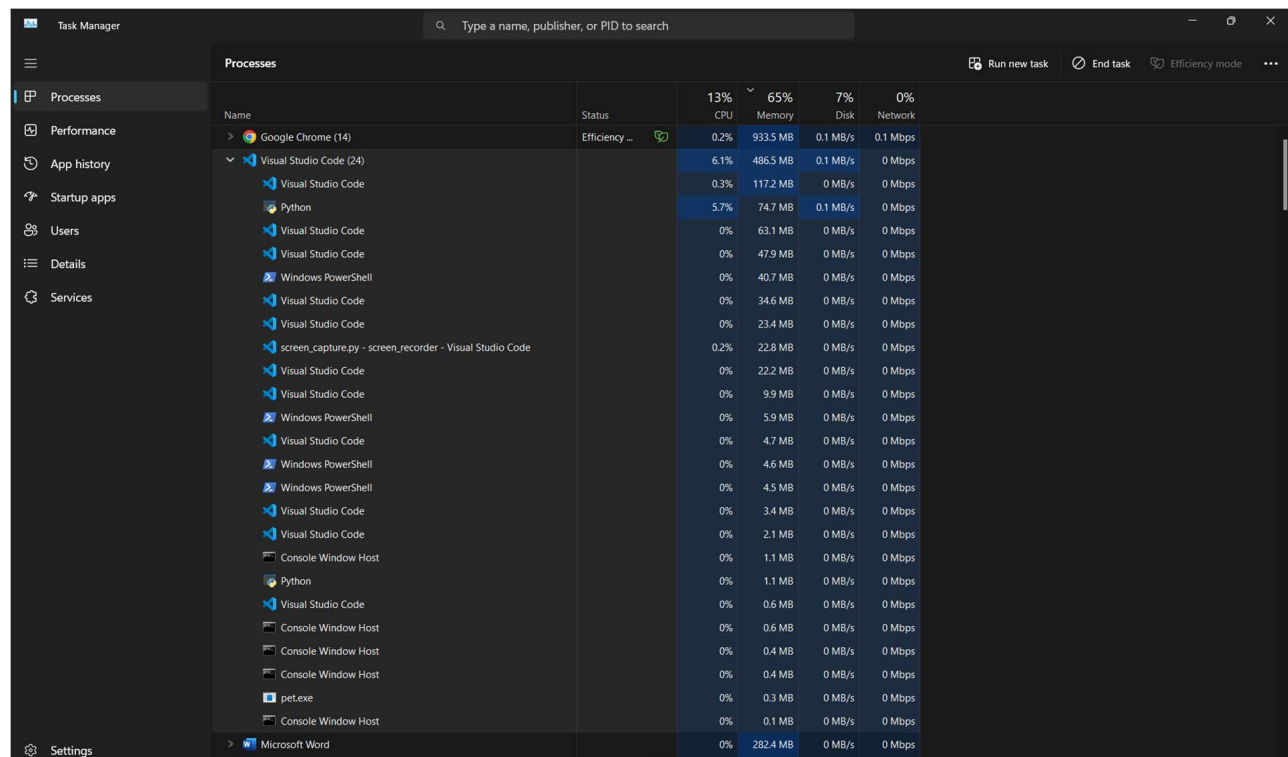
4.2 Output and Findings:



In the picture you can see the output the program is named as video capture and recorded_video.mp4 is being highlighted.

Video capture window shows us the output and the screen that is being recorded. And in the recorded_video.mp4 we can see the screen video that has been captured.

The project takes only 6% of the CPU while its alternatives takes about 25 to 30 percentage of CPU power.



Name	Status	CPU	Memory	Disk	Network
Google Chrome (14)	Efficiency ...	0.2%	933.5 MB	0.1 MB/s	0.1 Mbps
Visual Studio Code (24)		6.1%	486.5 MB	0.1 MB/s	0.1 Mbps
Visual Studio Code		0.3%	117.2 MB	0 MB/s	0 Mbps
Python		5.7%	74.7 MB	0.1 MB/s	0 Mbps
Visual Studio Code		0%	63.1 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	47.9 MB	0 MB/s	0 Mbps
Windows PowerShell		0%	40.7 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	34.6 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	23.4 MB	0 MB/s	0 Mbps
screen_capture.py - screen_recorder - Visual Studio Code		0.2%	22.8 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	22.2 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	9.9 MB	0 MB/s	0 Mbps
Windows PowerShell		0%	5.9 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	4.7 MB	0 MB/s	0 Mbps
Windows PowerShell		0%	4.6 MB	0 MB/s	0 Mbps
Windows PowerShell		0%	4.5 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	3.4 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	2.1 MB	0 MB/s	0 Mbps
Console Window Host		0%	1.1 MB	0 MB/s	0 Mbps
Python		0%	1.1 MB	0 MB/s	0 Mbps
Visual Studio Code		0%	0.6 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.6 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.4 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.4 MB	0 MB/s	0 Mbps
pet.exe		0%	0.3 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.1 MB	0 MB/s	0 Mbps
Microsoft Word		0%	282.4 MB	0 MB/s	0 Mbps

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

The project successfully achieved its goal of creating a simple screen recording tool. The software provides real-time screen capture with video encoding and displays the captured content as it is recorded. The solution is effective for basic use cases, such as recording tutorials or troubleshooting sessions.

5.2. Future Work

- **Feature Enhancements:** Adding audio recording capabilities to the screen recorder.
- **Performance Optimization:** Improve frame rates for smoother video recording.
- **Graphical User Interface (GUI):** Developing a user-friendly GUI for easier use.
- **Cross-Platform Compatibility:** Extending support for macOS and Linux systems.
- **Audio Recording:** Adding audio capture to synchronize video and sound.