Fluid Motion Analysis

A PROJECT REPORT SUBMITTED BY

G. T. MADUGODAGE
(S / 19 / 435)

in partial fulfillment of the requirement for the course of

CSC 3141 Image Processing Laboratory

to the
Department of Statistics and Computer Science

of the

FACULTY OF SCIENCE
UNIVERSITY OF PERADENIYA
SRI LANKA
2024

DECLARATION

I do hereby declare that the work reported in this project report was exclusively carried out by me under the supervision of Prof. Amalka Pinidiyaarachchi . It describes the results of my own independent work except where due reference has been made in the text. No part of this project report has been submitted earlier or concurrently for the same or any other degree.

Date: ……. ……

Signature of the Candidate

Certified by:

1. Instructor: **Mr. Kansaji Kotuwage**

   Date: ……. Signature:…….

2. Supervisor: **Prof. Amalka Pinidiyaarachchi**

   Date: ……. Signature:…….

3. Head of the Department: **Dr. Sachith Abeysundara**

   Date: ……. Signature:…….

ABSTRACT

In fluid dynamics, understanding the behavior of fluid flow is critical for applications in engineering, physics, and various scientific fields. This project aims to develop a real-time system to measure fluid flow velocity and calculate shear stress using image processing techniques. By utilizing computer vision, the system analyzes particle movement within a fluid from video footage to determine velocity and compute the corresponding shear stress at various points. The methodology involves frame extraction, contour detection, particle tracking, and velocity calculation using tools such as OpenCV, NumPy, and Matplotlib. The system also provides an interactive calibration feature to ensure accurate measurements based on real-world dimensions.

The results of this project demonstrate the ability to track single-particle motion with high accuracy, although challenges were noted when tracking multiple particles simultaneously. The system's performance can be further enhanced by integrating advanced tracking algorithms, such as Kalman filters, and machine learning techniques for improved accuracy and real-time analysis. This project lays the groundwork for further research in fluid dynamics, with potential applications in various industrial and scientific domains.

ACKNOWLEDGEMENTS

I would like to extend my sincere gratitude to the following individuals, without whose support and guidance, this project would not have been possible.

First and foremost, I would like to thank Dr. Amalka Pinidiyaarachchi for providing invaluable theoretical and practical knowledge in the field of digital image processing.Your expertise and guidance have been instrumental in shaping the foundation of this project.

I am also deeply grateful to Mr. Kansaji Kotuwage for his continuous support, practical insights, and guidance throughout the entire project. Your contributions were vital in overcoming various challenges during the implementation process.

Furthermore, I would like to express my heartfelt appreciation to Dr. Sachith Abeysundara, Head of the Department of Statistics and Computer Science, for providing the necessary facilities and resources that contributed significantly to the success of this project.

TABLE OF CONTENT

DECLARATION 2
ABSTRACT 3
TABLE OF CONTENT 5
LIST OF FIGURES 5
INTRODUCTION 1
1.1 Introduction 1
1.2 Problem statement 1
1.3 solution 1
RELATED WORK 4
METHODOLOGY 4
3.1 Image processing pipeline 5
3.2 Used Functions and Their Applications 6
3.3. Used Tools, Technologies, and Libraries 11
3.4. Procedure 12
RESULTS AND DISCUSSION 16
4.1 Major Resulting Steps and Some Tested Image Results 16
4.2 Discussion 17
CONCLUSIONS 18
5.1 Advantages of the Implemented System 18
5.2 Issues 19
5.3 Conclusion 19
REFERENCES 20

LIST OF FIGURES

Figure 1: Camera Setup Diagram
Figure 2: setup used for calibration
Figure 3: Flow chart diagram of the mythology used
Figure 4: pipeline for determining velocity
Figure 5: pipeline for image calibration
Figure 6: Code snippet for initialize log file
Figure 7: Code snippet for generate fluid id
Figure 8: Code snippet for calibration
Figure 9: Code snippet for process the video
Figure 10: Code snippet for process individual frame and calculate velocity
Figure 11: Code snippet for display velocity
Figure 12: Code snippet for Draw contour centroids
Figure 13: Code snippet for plot
Figure 14: Code snippet for plot paths
Figure 15: Code snippet for take individual particle paths
Figure 16: Code snippet for add fluid details into log file
Figure 17: Code snippet for calculate shear stress
Figure 18: Code snippet for main function
Figure 19: Environment setup and used variables
Figure 20: Calibrating an image containing known object(rectangle)
Figure 21: video processing procedure
Figure 21: velocity calculating procedure
Figure 21: Shear stress calculation used function
Figure 22: calibration result
Figure 22: detected particle and it’s centroid with velocity
Figure 23: Output plots
Figure 24: results saving in a log file

CHAPTER 01

INTRODUCTION

1.1 Introduction

Fluid dynamics plays a vital role in various engineering and scientific fields, including mechanical, civil, and environmental engineering. Understanding the flow behavior of fluids is crucial for applications ranging from industrial fluid transport systems to natural phenomena like river flows. One of the key parameters in fluid flow analysis is velocity, which helps in determining the movement and dynamics of the fluid. Additionally, the shear stress, which refers to the internal forces within the fluid layers, is another important factor, especially in assessing the fluid's interaction with solid boundaries.

The advancement in image processing techniques provides new ways to analyze fluid behavior by extracting meaningful information from visual data. The ability to track the movement of particles in a fluid, calculate their velocities, and estimate the shear stress using image sequences presents a powerful, non-invasive method of fluid flow analysis.

In this project, a video processing tool was developed to analyze the motion of particles in a fluid from video footage. By utilizing techniques like background subtraction, contour detection, and velocity estimation, the system calculates the average velocity of the fluid. Additionally, it estimates the shear stress, which is crucial in understanding the fluid's response to forces. The tool allows users to process video data, visualize particle movement, and log the calculated results for further analysis.

This project aims to bridge the gap between theoretical fluid dynamics and practical video-based fluid analysis, providing a cost-effective and efficient tool for velocity and shear stress calculation in fluid systems.

1.2 Problem statement

Accurate measurement of fluid velocity and shear stress is crucial in fluid dynamics but often requires expensive, time-consuming methods. Traditional techniques can be intrusive and complex, making real-time analysis difficult. This project aims to develop an efficient, non-invasive system to calculate velocity and shear stress from video footage of fluid flow, offering a simpler, automated solution for fluid analysis.

1.3 solution

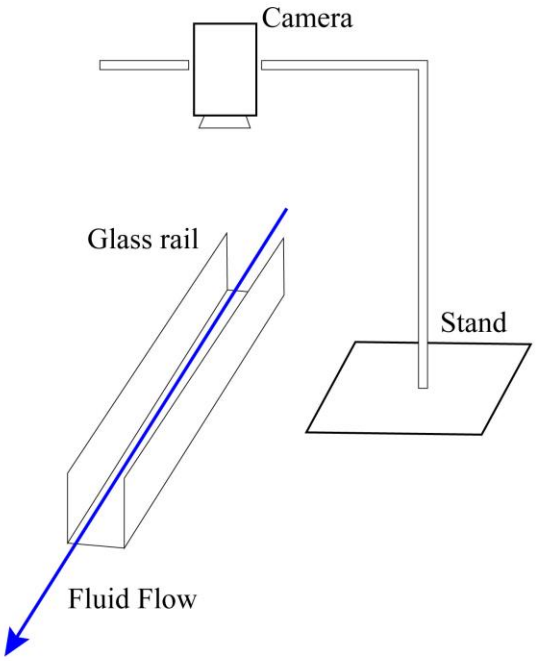
As illustrated in Figure 1, the setup was designed to measure fluid velocity by introducing traceable particles into the flow. In this experiment, tiny regiform balls were used to track the movement of fluid. The velocity was determined by measuring the speed of these particles as they flowed through the fluid.

To calculate shear stress, it was assumed that the fluid flow was laminar. The shear stress was computed using the formula:

$$\Gamma = \mu\left(\frac{\Delta v}{\Delta y}\right)$$

Where:
- T= Shear stress (Pa)
- μ= Dynamic viscosity of the fluid (Pa·s)
- Δv= Change in velocity between fluid layers (m/s)
- Δy = Distance between the fluid layers (m)

For accurate real-world measurements, the experimental setup, shown in Figure 2, incorporated known dimensions. A reference object with known width was used to calibrate the system, allowing us to determine how many pixels corresponded to 1 cm. Using this calibration, the measured velocities were converted from pixel units to real-world measurements, ensuring accuracy in the velocity and shear stress calculations.



i

Figure 1: Camera Setup Diagram

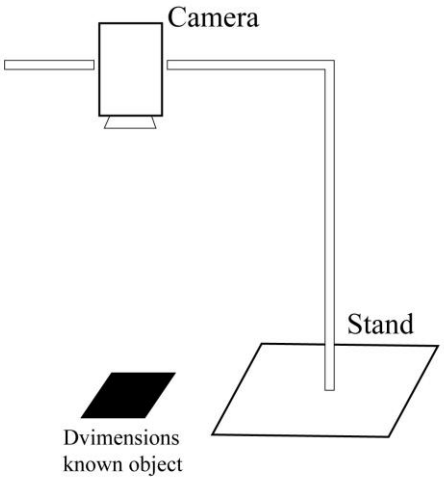


Figure 2: setup used for calibration

CHAPTER 02

RELATED WORK

In the field of fluid dynamics, various methods have been employed for measuring flow properties, including velocity and shear stress. Traditional approaches, such as Particle Image Velocimetry (PIV) and Laser Doppler Velocimetry (LDV), have been widely used in experimental fluid mechanics. These techniques offer precise measurements but often require expensive equipment and complex setups.

In recent years, advancements in digital image processing and machine learning have introduced more accessible alternatives. Techniques like optical flow estimation and contour detection have been applied to video footage to track fluid particles and compute their velocity. Research has also explored the use of Kalman filters for better tracking of particle movement and noise reduction in flow measurements.

However, most of these approaches are either highly specialized or difficult to implement without extensive technical knowledge. This project leverages basic image processing techniques and real-time video analysis to offer an easier-to-use, cost-effective solution for measuring fluid velocity and shear stress.

CHAPTER 03

METHODOLOGY

3.1 Image processing pipeline

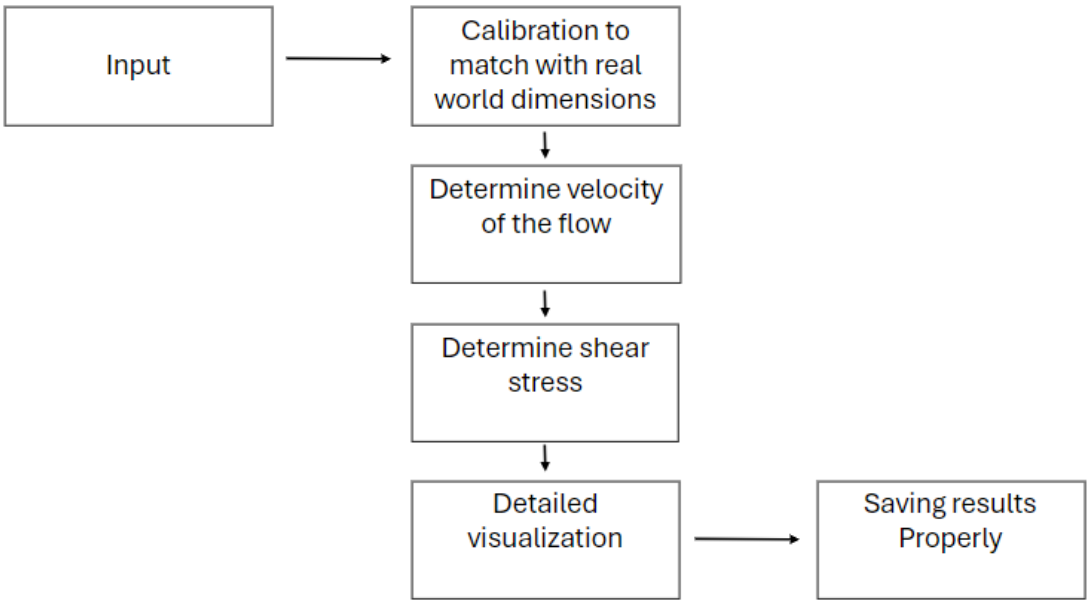


Figure 3: Flow chart diagram of the mythology used

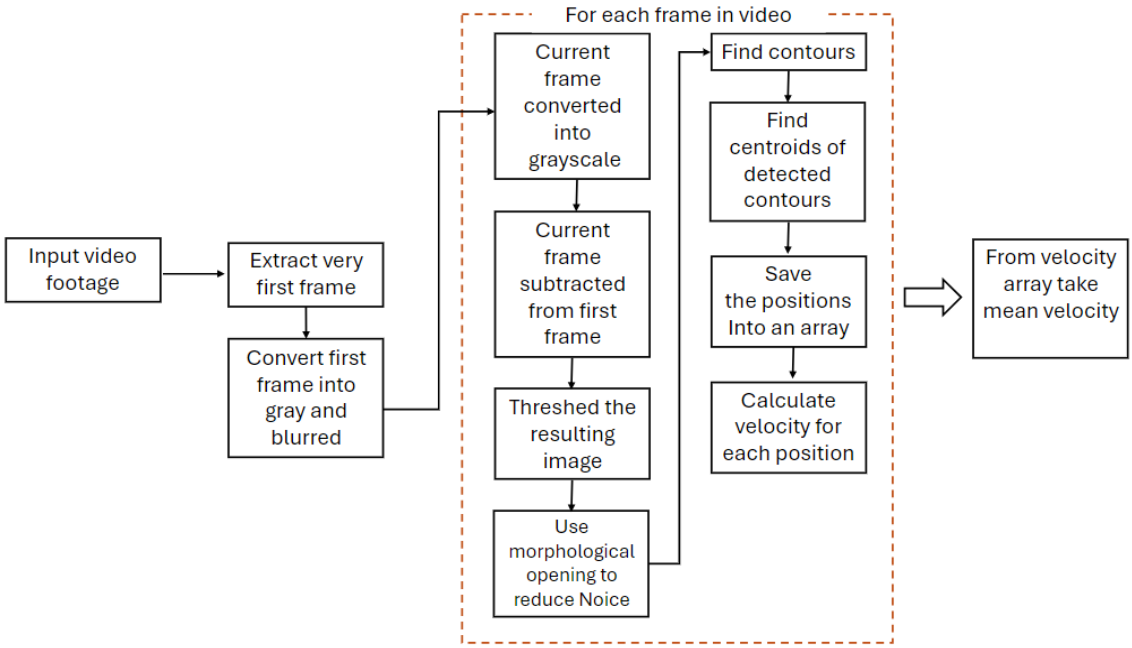


Figure 4: pipeline for determining velocity

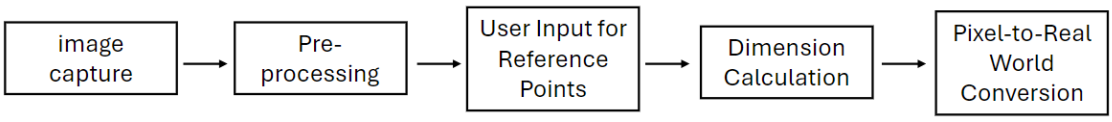


Figure 5: pipeline for image calibration

3.2 Used Functions and Their Applications

Initialize log file

Ensures that the log file for recording data exists, and initializes it with appropriate headers for fluid ID, date-time, average velocity, and average shear stress

```
# Ensure log file exists
def initialize_log_file(log_file):
    # Ensure the directory for the log file exists
    log_dir = os.path.dirname(log_file)
    if log_dir and not os.path.exists(log_dir):
        os.makedirs(log_dir)

    # Create and initialize the log file with headers if it doesn't exist
    if not os.path.exists(log_file):
        with open(log_file, 'w') as f:
            header = f"{'Fluid ID':<30} | {'Date-Time':<20} | {'Avg Velocity (m/s)':<20} | {'Avg Shear Stress (Pa)':<20}\n"
            f.write(header)
            f.write('-' * len(header) + '\n')
```

Figure 6: Code snippet for initialize log file

Generate fluid id

Generates a unique identifier for each fluid experiment using the current date and time.

```
# Generate unique fluid ID
def generate_fluid_id():
    return f"fluid_{datetime.now().strftime('%Y%m%d_%H%M%S')}"
```

Figure 7: Code snippet for generate fluid id

Calibrate pixels per cm

Enables the user to calibrate the conversion factor from pixels to centimeters by selecting known dimensions on a reference image. This factor is later used for real-world unit conversions.

```python
# Calibration part
def calibrate_pixels_per_cm():
    image = cv2.imread(CALIBRATION_IMG)
    drawing = False
    ix, iy = -1, -1
    pixels_per_cm_x = PIXELS_PER_CM_DEFAULT
    pixels_per_cm_y = PIXELS_PER_CM_DEFAULT

    def draw(event, x, y, flags, param):
        nonlocal ix, iy, drawing, pixels_per_cm_x, pixels_per_cm_y

        if event == cv2.EVENT_LBUTTONDOWN:
            drawing = True
            ix, iy = x, y
        elif event == cv2.EVENT_MOUSEMOVE and drawing:
            img_copy = image.copy()
            cv2.rectangle(img_copy, (ix, iy), (x, y), (0, 255, 0), 2)
            cv2.imshow('image', img_copy)
        elif event == cv2.EVENT_LBUTTONUP:
            drawing = False
            pixels_per_cm_x, pixels_per_cm_y = calculate_conversion_factor(ix, iy, x, y)

    def calculate_conversion_factor(ix, iy, x, y):
        width_pixels = abs(x - ix)
        height_pixels = abs(y - iy)
        return width_pixels / KNOWN_WIDTH_CM, height_pixels / KNOWN_HEIGHT_CM

    cv2.namedWindow('image')
    cv2.setMouseCallback('image', draw)

    while True:
        cv2.imshow('image', image)
        if cv2.waitKey(1) & 0xFF == 27:  # Press 'ESC' to exit
            break

    cv2.destroyAllWindows()
    # Return the calculated pixels per cm values
    return pixels_per_cm_x, pixels_per_cm_y
```

Figure 8: Code snippet for calibration

Video processing function

Processes the video footage to track particles and compute velocities. It reads frames, processes each frame to detect contours, and calculates particle movement between frames.

```python
# Base video processing function
def process_video(pixels_per_cm_x, pixels_per_cm_y):
    cap = cv2.VideoCapture(VIDEO_FILE)
    fps = cap.get(cv2.CAP_PROP_FPS)
    frame_width, frame_height = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    delay = int(1000 / fps)

    ret, first_frame = cap.read()
    if not ret:
        print("Error: Unable to read the video file.")
        return

    first_frame_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

    pos = [[0, 0]]
    velocity = [[0, 0]]
    velocity_magnitude = []
    i = 1

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame_sub = cv2.absdiff(frame_gray, first_frame_gray)
        _, frame_bin = cv2.threshold(frame_sub, THRESH_VALUE, 255, cv2.THRESH_BINARY)
        frame_bin = cv2.morphologyEx(frame_bin, cv2.MORPH_OPEN, KERNEL)

        contours, _ = cv2.findContours(frame_bin, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        process_frame(frame, contours, pos, velocity, velocity_magnitude, i, fps, pixels_per_cm_x, pixels_per_cm_y)
        cv2.imshow('Frame', frame)
        i += 1

        if cv2.waitKey(delay) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    return pos, velocity_magnitude, frame_width, frame_height
```

Figure 9: Code snippet for process the video

Process individual frames and calculate velocity

Handles the detection of the largest particle (using contours), computes its centroid, and calculates velocity based on particle movement across frames.

```python
# Process individual frames and calculate velocity
def process_frame(frame, contours, pos, velocity, velocity_magnitude, i, fps, pixels_per_cm_x, pixels_per_cm_y):
    if contours:
        contour = max(contours, key=cv2.contourArea)
        M = cv2.moments(contour)

        if M["m00"] != 0:
            cx, cy = int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"])
            pos.append([cx, cy])

            if i != 0:
                Vx_pixels = (pos[i][0] - pos[i - 1][0]) * fps
                Vy_pixels = (pos[i][1] - pos[i - 1][1]) * fps
                Vx_meters = Vx_pixels / pixels_per_cm_x / 100
                Vy_meters = Vy_pixels / pixels_per_cm_y / 100
                velocity.append([Vx_meters, Vy_meters])
                magnitude = np.sqrt(Vx_meters**2 + Vy_meters**2)
                velocity_magnitude.append(magnitude)
                display_text(frame, f"Velocity: {magnitude:.2f} m/s")
            else:
                velocity.append([0, 0])
                velocity_magnitude.append(0)
        draw_contour(frame, contour, cx, cy)
    else:
        pos.append([0, 0])
```

Figure 10: Code snippet for process individual frame and calculate velocity

Display velocity on frame

Overlays velocity information on the video frame for real-time display.

```python
# Display velocity on frame
def display_text(frame, text):
    cv2.putText(frame, text, TEXT_POSITION, cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

Figure 11: Code snippet for display velocity

Draw contour and centroid on frame

Draws the detected particle contour and its centroid on the frame for visualization.

```python
# Draw contour and centroid on frame
def draw_contour(frame, contour, cx, cy):
    cv2.drawContours(frame, [contour], -1, (0, 255, 0), 2)
    cv2.circle(frame, (cx, cy), 7, (255, 0, 0), -1)
    cv2.putText(frame, "centroid", (cx - 25, cy - 25), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
```

Figure 12: Code snippet for Draw contour centroids

Plot particle paths and individual traces

Visualizes particle movement paths with color coding based on velocity and individual particle traces in a matplotlib plot.

```python
def plot_paths(pos, velocity_magnitude, average_velocity, frame_width, frame_height, sub_arrays):
    plt.figure(figsize=(10, 5))

    # Particle Paths Plot
    plt.subplot(1, 2, 1)
    plot_particle_paths(pos, velocity_magnitude, average_velocity, frame_width, frame_height)

    # Individual Traces Plot
    plt.subplot(1, 2, 2)
    plot_individual_traces(sub_arrays, frame_width, frame_height)

    plt.savefig("plots")
    plt.show()
```

Figure 13: Code snippet for plot

Particle paths with velocity color coding

Plots the particle paths with colors indicating velocity deviations from the average.

```python
def plot_particle_paths(pos, velocity_magnitude, average_velocity, frame_width, frame_height):
    valid_positions = [p for p in pos if p != [0, 0]]
    max_deviation = max(abs(mag - average_velocity) for mag in velocity_magnitude)

    for idx, (x, y) in enumerate(valid_positions):
        magnitude = velocity_magnitude[idx] if idx < len(velocity_magnitude) else 0
        deviation = magnitude - average_velocity
        intensity = min(abs(deviation) / max_deviation, 1.0)
        color = (intensity, 0, 0) if deviation > 0 else (0, 0, intensity)
        plt.plot(x, y, 'o', color=color)

    plt.xlim(0, frame_width)
    plt.ylim(frame_height, 0)
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.title("Particle Paths with Velocity Color Coding")
```

Figure 14: Code snippet for plot paths

Individual traces from sub-arrays

Plots the individual traces of particles' movement across the video frames.

```python
def plot_individual_traces(sub_arrays, frame_width, frame_height):
    valid_points = {k: v for k, v in sub_arrays.items() if v and all(isinstance(item, list) for item in v)}
    for key, point_sets in valid_points.items():
        x, y = zip(*point_sets)
        plt.plot(x, y, label=f"{key + 1}", color=(random.random(), random.random(), random.random()))

    plt.xlim(0, frame_width)
    plt.ylim(frame_height, 0)
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.title("Individual Paths")
    plt.legend()
```

Figure 15: Code snippet for take individual particle paths

Log velocity and shear stress data

Logs the calculated average velocity and shear stress into the log file in a structured format.

```python
def log_velocity_data_shear_stress(log_file, fluid_id, average_velocity, average_shear_stress):
    with open(log_file, 'a') as log:
        log_entry = f"{fluid_id:<30} | {datetime.now().strftime('%Y-%m-%d %H:%M:%S'):<20} | {average_velocity:<20.2f} | {average_shear_stress:<20.2f}\n"
        log.write(log_entry)
```

Figure 16: Code snippet for add fluid details into log file

calculate shear stress

Computes shear stress using the velocity gradient across layers of fluid, applying the formula

$$\Gamma = \mu\left(\frac{\Delta v}{\Delta y}\right)$$

```python
def calculate_shear_stress(velocity_magnitude, pixels_per_cm_y, viscosity):
    shear_stress = []
    for i in range(1, len(velocity_magnitude)):
        # Approximate velocity gradient (du/dy)
        du_dy = (velocity_magnitude[i] - velocity_magnitude[i - 1]) / (1 / pixels_per_cm_y)  # Change in velocity over distance
        shear = viscosity * du_dy
        shear_stress.append(shear)
    return np.mean(shear_stress)
```

Figure 17: Code snippet for calculate shear stress

Main function

The main driver function that initializes logging, handles calibration, processes the video, computes average velocity and shear stress, and logs the results while also providing visualization plots of particle movement.

```python
def main():
    initialize_log_file(LOG_FILE)
    fluid_id = generate_fluid_id()

    # Calibrate if needed
    if input("Do you want to perform calibration? (y/n): ").lower() == 'y':
        pixels_per_cm_x, pixels_per_cm_y = calibrate_pixels_per_cm()
    else:
        pixels_per_cm_x, pixels_per_cm_y = PIXELS_PER_CM_DEFAULT, PIXELS_PER_CM_DEFAULT

    #visocity inputable
    viscosity = float(input("Enter the fluid's dynamic viscosity (in Pa·s): "))

    pos, velocity_magnitude, frame_width, frame_height = process_video(pixels_per_cm_x, pixels_per_cm_y)


    # Calculate average velocity
    average_velocity = np.mean(velocity_magnitude)
    print(f"Average Velocity: {average_velocity:.2f} m/s")

    # calculate average shear stress
    average_shear_stress = calculate_shear_stress(velocity_magnitude, pixels_per_cm_y, viscosity)
    print(f"Average Shear Stress: {average_shear_stress:.2f} Pa")

    log_velocity_data_shear_stress(LOG_FILE, fluid_id, average_velocity,average_shear_stress)

    sub_arrays = {}
    current_sub_array = []
    for element in pos:
        if element == [0, 0]:
            if current_sub_array:
                sub_arrays[len(sub_arrays)] = current_sub_array
                current_sub_array = []
        else:
            current_sub_array.append(element)
    if current_sub_array:
        sub_arrays[len(sub_arrays)] = current_sub_array

    plot_paths(pos, velocity_magnitude, average_velocity, frame_width, frame_height, sub_arrays)


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

Figure 18: Code snippet for main function

3.3. Used Tools, Technologies, and Libraries

1. OpenCV (cv2)

A computer vision library used for video and image processing tasks such as contour detection, frame processing, and object tracking. OpenCV provides the core functionalities for detecting particle movements and calculating velocities.

2. NumPy

A fundamental library for scientific computing in Python, used for handling arrays, performing mathematical operations, and calculating statistics like velocity magnitude and shear stress.

3. Matplotlib

A plotting library used to create visual representations of particle movement paths and individual traces. It generates graphical plots for velocity color

coding and particle paths.

4. Python Standard Libraries

os: Used for file and directory operations, such as creating the log file and checking for the existence of directories.

datetime: Utilized to generate timestamps for logging the fluid ID and the time of measurement.

random: Used for assigning random colors to individual particle traces in the visualization.

5. Video Processing Tools:

A video file (`test_footage_4.mp4`) is processed to track the movement of particles, from which velocities are calculated. The user can interact with the video using real-time frame display.

6. Log File (log.txt):

A custom log file used to store the results of each experiment, including fluid ID, timestamp, average velocity, and shear stress values.

7. Hardware (Camera/Video):

A video capturing device is used to record footage of particle movement, which is then processed to extract velocity and shear stress information.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random
import os
from datetime import datetime


# Constants
PIXELS_PER_CM_DEFAULT = 20.0  # Default pixels per cm (can be calibrated)
THRESH_VALUE = 30
KERNEL = np.ones((7, 7), np.uint8)
TEXT_POSITION = (20, 20)
LOG_FILE = 'log.txt'
KNOWN_WIDTH_CM = 5.0
KNOWN_HEIGHT_CM = 5.0
CALIBRATION_IMG = 'callibration/img/calibrating_img.JPG'
VIDEO_FILE = 'video_application/res/test_footage_4.mp4'
viscosity = 0.01
```

Figure 19: Environment setup and used variables

3.4. Procedure

The procedure for the fluid flow analysis system involves several steps, from calibration to image processing and finally to calculating velocity and shear stress. Below is a step-by-

step breakdown of the process:

1.Setup and Calibration

- A camera or video recording device is set up to capture footage of fluid flow with traceable particles (e.g., tiny regiform balls).
- Before processing the video, the system must be calibrated to determine the real-world dimensions from the pixels in the video.
- The calibration process involves selecting known reference points in an image (e.g., an object with known dimensions) and using them to calculate the number of pixels per centimeter (pixels/cm). This allows the system to translate the distance in the video to real-world units.

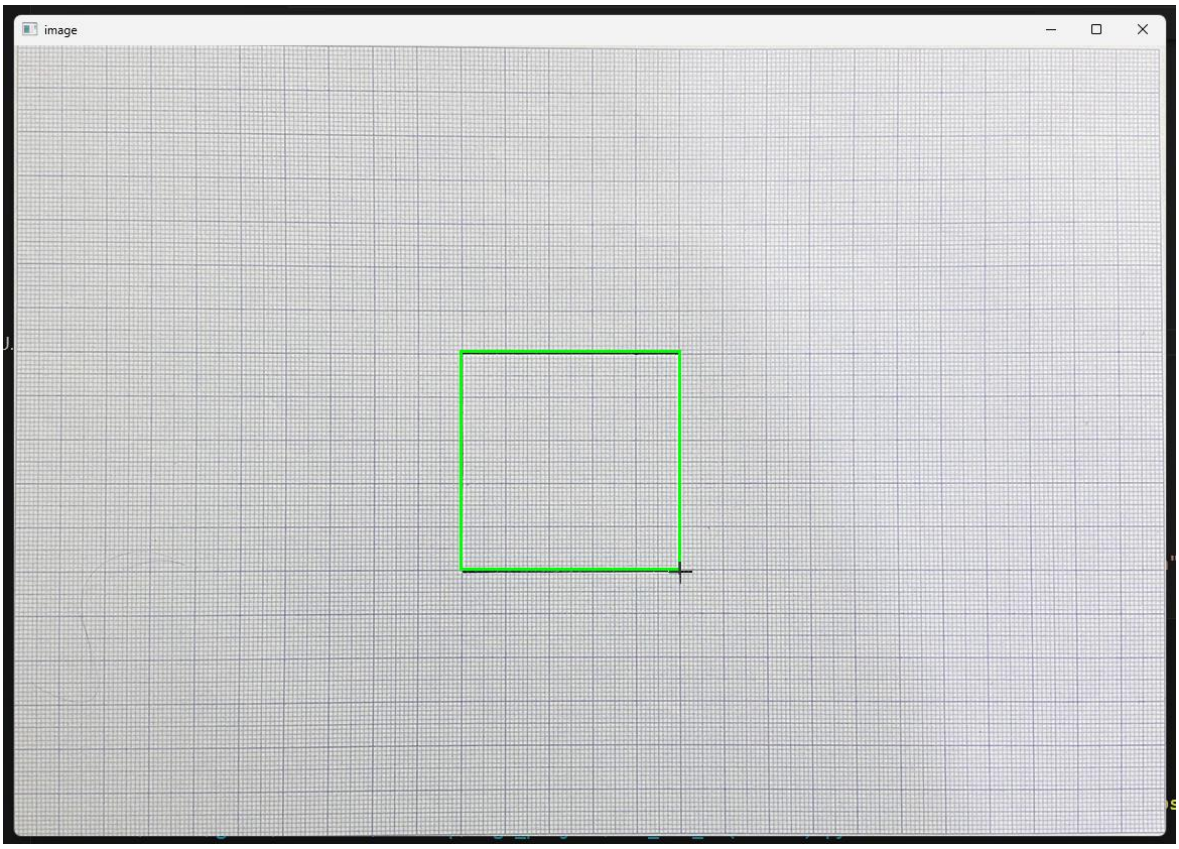


Figure 20: Calibrating an image containing known object(rectangle)

2. Video Processing:

- Once calibration is completed, the program processes the input video footage.
- Each frame of the video is converted to grayscale to simplify processing and to enhance contrast between the moving particles and the background.
- A background subtraction technique is applied to isolate moving objects (the particles) from the static background, followed by thresholding and morphological operations to remove noise and improve detection accuracy.
- Contours are detected in the processed video frames, and the centroid of each detected particle is calculated.

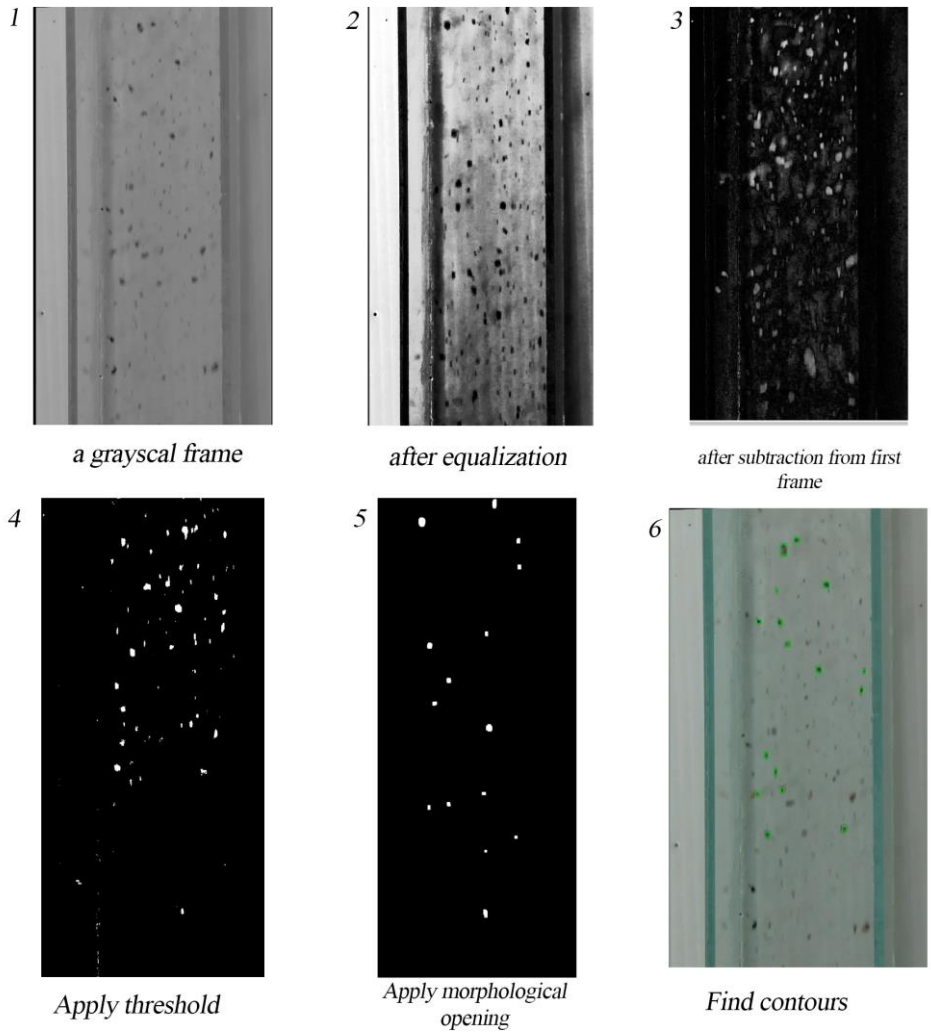


Figure 21: video processing procedure

3. Tracking Particle Movement:

- The system tracks the detected particles across frames by calculating the position of the centroids of the particles.
- It computes the velocity of the particles by measuring the change in position between consecutive frames.
- The velocity is converted from pixels per frame to meters per second using the calibration data (pixels/cm).

4. Velocity Calculation:

- For each detected particle, the program calculates the velocity in the X and Y directions by comparing the positions of the particle in consecutive frames.
- The average velocity of all detected particles is computed and logged.

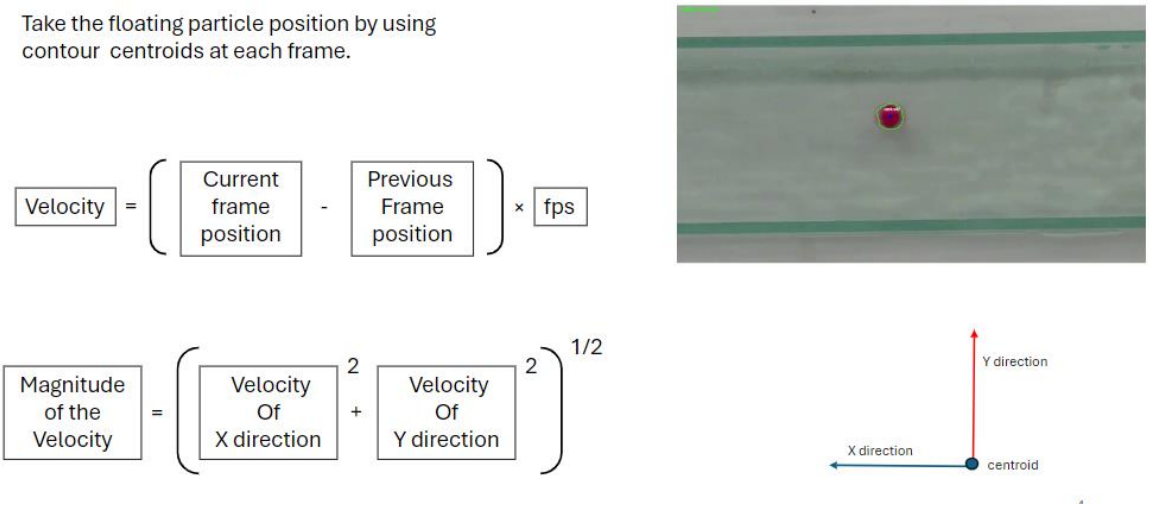


Figure 21: velocity calculating procedure

5.Shear Stress Calculation:

$$T = \mu (\Delta y / \Delta v)$$

Where:

- T = Shear stress (Pa)
- μ\muμ = Dynamic viscosity of the fluid (Pa·s)
- Δv\Delta vΔv = Change in velocity between fluid layers (m/s)
- Δy\Delta yΔy = Distance between layers (m)

```
du_dy = (velocity_magnitude[i] - velocity_magnitude[i - 1]) / (1 / pixels_per_cm_y)
shear = viscosity * du_dy
```

Figure 21: Shear stress calculation used function

6. Data Logging:

- The results of the experiment, including the fluid ID, timestamp, average velocity, and shear stress, are logged in a structured format in a log file (`log.txt`).

7. Visualization:

- The program generates visual plots showing the particle movement paths and the corresponding velocities.
- A particle path plot with velocity color-coding is created, highlighting the speed of particles as they move through the fluid.
- Individual particle traces are plotted, showing the paths followed by particles across the video frames.

CHAPTER 04

RESULTS AND DISCUSSION

4.1 Major Resulting Steps and Some Tested Image Results

Calibration of Pixels to Real-World Dimensions:

- The system used a known object with defined dimensions to calibrate the conversion factor between pixels and centimeters.
- This allowed for accurate conversion of particle movements in the video to real-world distances.

```
Pixels per cm (Width): 22.6
Pixels per cm (Height): 20.8
```

Figure 22: calibration result

Detection and Tracking of Particles and Velocity Calculation

- Contours representing the particles were detected in each video frame.
- The centroids of the particles were tracked across consecutive frames, allowing the system to compute their movement over time.
- The velocity of each particle was calculated by measuring the change in position between consecutive frames and converting it to real-world units using the calibrated pixels-per-centimeter ratio.
- The program logged the average velocity of the particles across all frames.

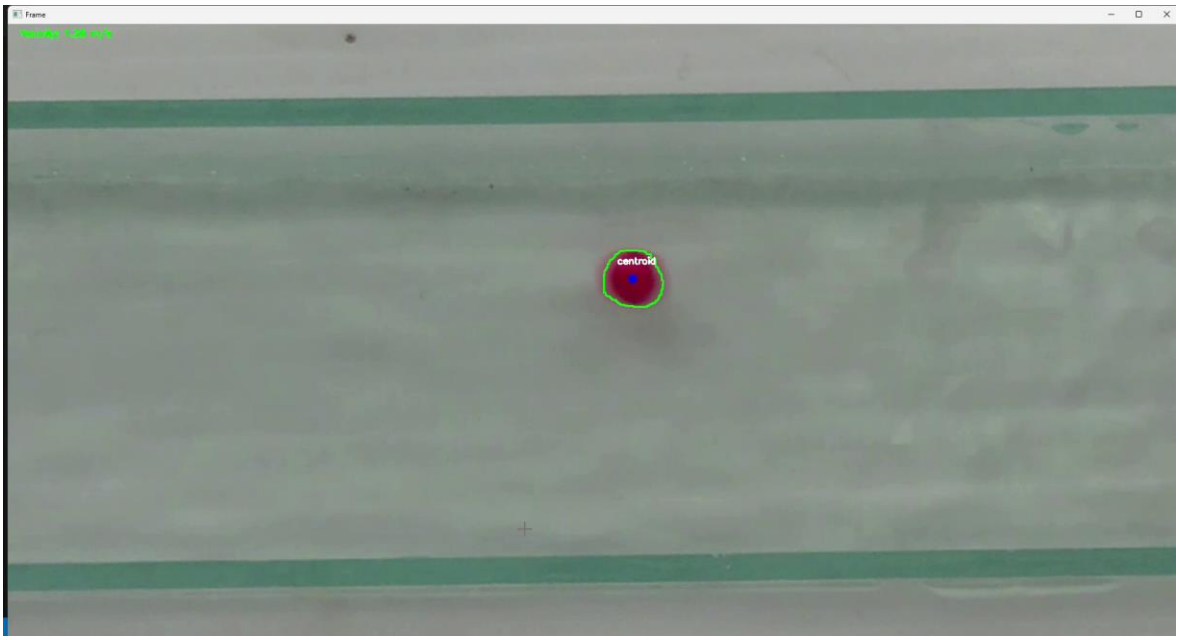


Figure 22: detected particle and it's centroid with velocity

Visualization and Output

- The system generated visual plots, showing particle paths and color-coded velocity magnitudes.
- The average velocity and shear stress values were logged in a text file for future reference.

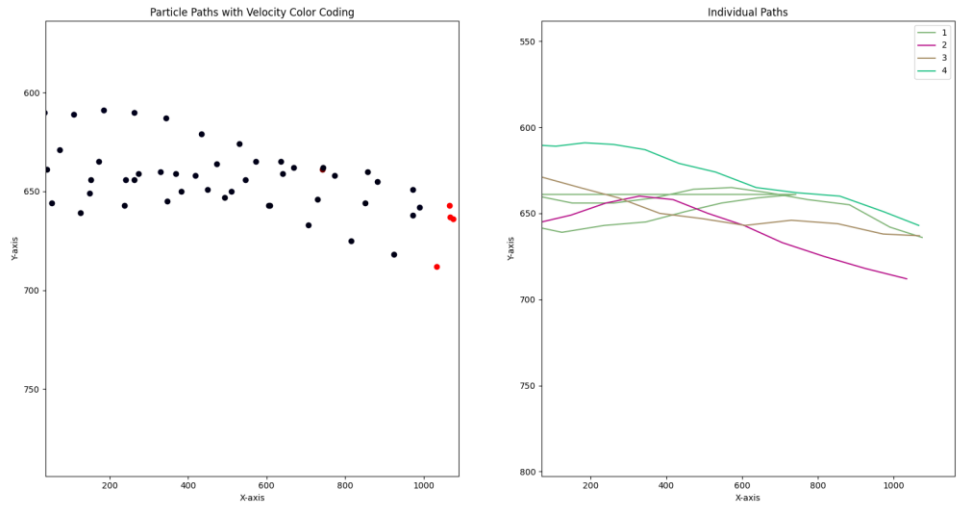


Figure 23: Output plots

Log file

```
Fluid ID                    | Date-Time           | Avg Velocity (m/s)   | Avg Shear Stress (Pa)
-------------------------------------------------------------------------------------------------
fluid_20240915_190601       | 2024-09-15 19:06:26 | 2.96                 | -0.07
fluid_20240915_190647       | 2024-09-15 19:07:28 | 1.40                 | -1.72
```

Figure 24: results saving in a log file

(WATCH A SAMPLE VIDEO DEMONSTRATING HOW THE PROGRAM WORKS.)

4.2 Discussion

The fluid flow analysis system developed in this project demonstrated several key functionalities, including calibration, particle tracking, and velocity and shear stress calculation. Below are the main points of discussion:

1. **Efficiency of the System**:
    - The system performed effectively for tracking individual particles in laminar flow conditions. The background subtraction and noise removal techniques proved useful for detecting the particles with a reasonable degree of accuracy.
    - The system managed to calculate and log average velocity and shear stress values based on the particle movements, providing reliable results when the flow was consistent and uniform.
2. **Challenges Faced**:
    - **Complex Flow Patterns**: When multiple particles overlapped or the flow became turbulent, the system struggled to maintain accurate tracking. Particle detection and velocity calculation became less reliable in these conditions.

- **Noise and Variability in Video Frames**: Depending on the video quality and lighting conditions, additional noise and variability in the frames sometimes interfered with particle detection.
- **Manual Calibration**: The manual calibration process could introduce errors if the selected reference points were not accurate, affecting the pixel-to-centimeter conversion.

3. **Accuracy of Results**:
   - The results for velocity and shear stress were accurate for simple, laminar flow situations. However, the precision decreased when the flow became more complex or when multiple particles were present.
   - The shear stress calculation depended heavily on the accuracy of the velocity gradient, which could be influenced by variations in particle detection.
4. **Observations Regarding Code Performance**:
   - The program performed well in terms of processing speed, efficiently handling video footage of typical resolution.
   - Future improvements could include better handling of complex flows, enhanced particle detection techniques, and automation of the calibration process to reduce user input errors. Additionally, employing advanced filtering techniques like Kalman filters or machine learning could improve particle tracking accuracy in challenging scenarios.

CHAPTER 05

CONCLUSIONS

5.1 Advantages of the Implemented System

The fluid flow analysis system developed in this project demonstrated several strengths,

including:

- **Real-Time Performance**: The system efficiently processes video footage in real-time, detecting and tracking the movement of particles in a fluid. This allows for immediate feedback on velocity and shear stress values during experimentation.
- **Accurate Velocity and Shear Stress Calculations**: Under laminar flow conditions, the system provides reasonably accurate velocity measurements based on particle tracking. Shear stress calculations are also derived accurately using the input viscosity and observed velocity gradients.
- **Adaptability to Different Fluids**: The system is flexible enough to analyze a variety of fluids. By adjusting parameters such as viscosity, users can adapt the program to different fluid types, allowing for a broad range of experiments.
- **User-Friendly Calibration**: The manual calibration process allows users to easily adapt the system to real-world dimensions, ensuring accurate conversion from pixels to physical units.

5.2 Issues

Despite its advantages, the system also has some limitations:

- **Performance with Multiple Particles**: When multiple particles move within the fluid, the program struggles to maintain accurate tracking and velocity calculation, particularly when particles overlap or move in unpredictable ways.
- **Noise Sensitivity**: The system's performance is affected by video quality and lighting conditions. Excessive noise or variability in the frames can interfere with particle detection, leading to errors in velocity and shear stress calculations.
- **Manual Calibration**: The manual calibration process, though flexible, relies on accurate user input. Small errors in selecting calibration points can affect the overall accuracy of the system.
- **Viscosity Input by the User**: The system assumes that the viscosity of the fluid is provided by the user. This introduces an external factor that could affect the accuracy of the shear stress calculations if the input viscosity is incorrect or unknown.
- **Assumption of Laminar Flow**: The system assumes that the fluid flow is laminar, which may not always be the case. In more turbulent conditions, the accuracy of the results can degrade significantly.

5.3 Conclusion

Overall, the project successfully implemented a functional fluid flow analysis system capable of calculating velocity and shear stress based on particle tracking in video footage. The system performs well under controlled conditions, offering reliable results when the flow is laminar and the video is clear.

The system's real-time processing capabilities and adaptability to different fluids make it a useful tool for fluid dynamics experiments. However, certain limitations, such as performance with multiple particles, sensitivity to video noise, and reliance on user-provided viscosity values, suggest areas for further improvement.

Future enhancements could include more advanced particle tracking algorithms, automated calibration, and the integration of machine learning techniques to handle complex flow patterns. These improvements would enhance the system's accuracy, usability, and ability to handle a broader range of fluid dynamics scenarios.

REFERENCES

[1] Gonzalez, R.C., Woods, R.E. (2018). Digital Image Processing (4th ed.).

[2] Bradski, G., Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, Inc.

[3] Matplotlib Development Team. (2023). Matplotlib: Visualization with Python.

[4] Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer.

i