Ship hull water level detection using optical flow and OCR

ENPM 673 Final Project

1st Prathinav Karnala Venkata
M.Eng Robotics
University of Maryland
College Park, Maryland, USA
pratkv@umd.edu

2nd Pranav ANV
M.Eng Robotics
University of Maryland
College Park, Maryland, USA
anvpran@umd.edu

3rd Sarang Shibu
M.Eng Robotics
University of Maryland
College Park, Maryland, USA
sarang@umd.edu

4th Vien-Phuong T. Le
M.S Electrical and Computer Engineering
University of Maryland
College Park, Maryland, USA
vple2159@umd.edu

***Abstract*—Water level measurements on ship hulls must be accurate in order to guarantee both operational effectiveness and maritime safety. Manually reading draft marks using traditional methods takes time and is prone to human error. With optical flow and optical character recognition (OCR) on pre-recorded video footage, this project, "Water Level Detections from Ship Hull," demonstrates an automated water level detection method. The system reads the draft marks using OCR, tracks waterlines using optical flow, and corrects perspective distortions in video frames. When compared to manual readings, the suggested method shows notable improvements in automation and accuracy, making it a dependable and efficient way to measure water levels.**

Keywords: Water Level Detection, Optical Flow, Optical Character Recognition (OCR), Computer Vision, Image Processing, Homography Transformation, Draft Marks, Maritime Operations, Video Analysis

I. Introduction

In maritime operations, measuring the water levels on ship hulls is a crucial task. It is essential for maintaining ship stability, load control, and adherence to safety rules. This procedure has historically involved manually reading draft marks that have been painted on the hull. Due to human error, external factors, and physical obstacles, these manual methods are prone to errors.

The development of automated systems to improve and expedite this process is becoming more and more popular as a result of developments in computer vision and image processing. Water level detection that is automated can be more accurate and efficient, lowering the need for human operators and error-proneness.

The goal of this project, "Water Level Detections from Ship Hull," is to create a system that can detect water levels from pre-recorded video footage of ship hulls using optical flow and optical character recognition (OCR) techniques. The system tackles a number of significant issues:

- **Perspective Distortion:** Draft marks may appear distorted in video footage taken from different perspectives, making accurate reading challenging. To fix these distortions and align the draft marks horizontally for easier reading, the system uses homography transformation.
- **Changing Lighting Conditions:** The visibility of the draft marks can be impacted by environmental elements like lighting and reflections. Accurate OCR is made possible by the system's image enhancement techniques, which enhance the contrast and clarity of the marks.

We outline our project's methodology, specifics of its implementation, and outcomes in this report. We demonstrate the system's potential for useful applications in maritime operations and highlight the efficacy of combining optical flow and OCR for water level detection.

II. Methodology

The process of detecting water level from a ship's hull includes several stages, including optical flow computation, video processing, homography transformation, and OCR. The following are the specific steps:

A. Processing of Videos

The first step in detecting water levels from a ship's hull involves processing the video frames. This step is crucial for preparing each frame for further examination. The detailed process includes:

1) Frame Capture:

- **Video Frames**: Frames are captured from a pre-recorded video of a ship's hull. This involves reading the video file using the `cv2.VideoCapture` function in OpenCV. The function initializes the video capture object with the video file.

- **Frame-by-Frame Analysis**: The video is processed frame by frame to allow for detailed analysis of each frame individually. This is achieved by using a loop that reads each frame until the end of the video is reached.

2) Grayscale Conversion:

- **Convert to Grayscale**: Each captured frame is converted from color (BGR) to grayscale using `cv2.cvtColor`. Grayscale conversion simplifies the image by reducing the number of color channels from three to one, which reduces computational complexity and highlights the structural features of the image.

3) Thresholding:

- **Binary Thresholding**: A binary threshold is applied to the grayscale image using `cv2.threshold`. This step separates the hull from the background by setting pixel values above a certain threshold to white and those below it to black. The threshold value is chosen based on the lighting conditions and the contrast between the hull and the background.

Fig. 1. Sample Frame

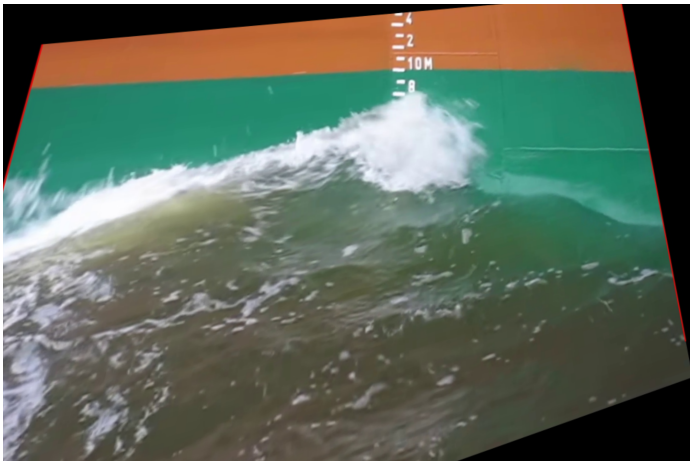

Fig. 2. Warped Image

B. Conversion of Homography

The second step in detecting water levels from a ship's hull involves applying a homography transformation. This step ensures that the hull's draft marks are correctly positioned for Optical Character Recognition (OCR) by correcting perspective distortion caused by the camera angle. The detailed process includes:

1) Region of Interest (ROI) Definition:

- **Define ROI**: The region of interest (ROI) is defined to focus on the part of the frame where the hull's draft marks are located. This step isolates the area of the frame that contains useful information, excluding irrelevant parts. The coordinates of the ROI are set based on prior knowledge or initial inspection of the frames.

2) Mask Creation:

- **Create Mask**: A mask is created to isolate the ROI. The mask is a binary image where the pixels within the ROI are set to white (255), and all other pixels are set to black (0). This mask is applied to the frame to retain only the ROI for further processing using `cv2.rectangle` to draw the ROI and `cv2.bitwise_and` to apply the mask.

3) Color Thresholding:

- **Highlight Draft Marks**: The draft marks are highlighted using color thresholding. Threshold values for the color

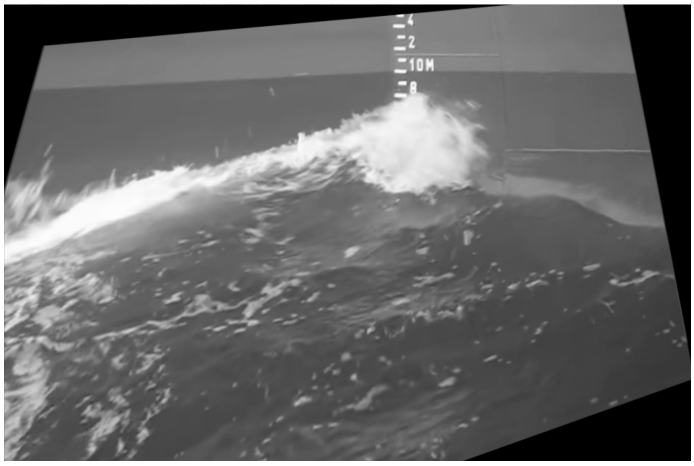

Fig. 3. Gray scale image

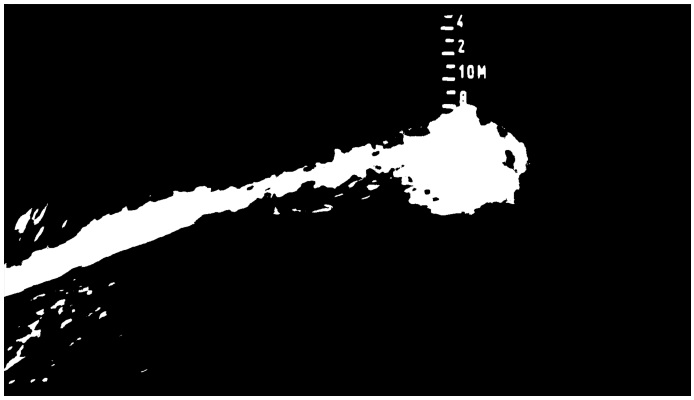

Fig. 4. Sharpened Binary Thresholded image used for OCR

of the draft marks (e.g., shades of orange) are set to create a binary mask that isolates the marks from the rest of the frame. The `cv2.inRange` function [3] is used for this purpose, where the lower and upper bounds of the color are defined to detect the desired marks.

4) Gaussian Blurring:

- **Reduce Noise**: Gaussian blurring is applied to the color-thresholded image to reduce noise and smooth the image. This step helps in minimizing small variations that could interfere with edge detection. The `cv2.GaussianBlur` function with a kernel size (e.g., 7x7) is used to achieve this smoothing effect.

5) Edge Detection:

- **Detect Edges**: Edges in the blurred image are detected using the Canny edge detection algorithm. This step highlights the boundaries of the draft marks, making it easier to detect them as lines in the next step. The `cv2.Canny` function is used with defined thresholds for edge detection.

6) Hough Line Transform:

- **Detect Lines**: The Hough Line Transform is applied to the edge-detected image to find lines that correspond to the draft marks. This method detects lines by transforming points in the image space to the parameter space and identifying the lines that accumulate the most votes. The `cv2.HoughLinesP` function is used with parameters such as rho, theta, and thresholds to detect the lines.

7) Homography Matrix Calculation:

- **Compute Homography Matrix**: The points corresponding to the detected lines are used to calculate the homography matrix. The homography matrix is a transformation matrix that maps the points from the original perspective to a new perspective where the draft marks are aligned horizontally. The source points (detected line points) and destination points (desired alignment) are used in the `cv2.findHomography` function to compute the matrix.

8) Frame Warping:

- **Apply Homography**: The computed homography matrix is applied to the frame to warp it. This warping adjusts the perspective so that the draft marks are aligned horizontally, making them suitable for OCR. The `cv2.warpPerspective` function is used with the homography matrix to transform the frame.

C. Computation of Optical Flow

The third step involves calculating the motion of objects between frames using optical flow. Optical flow is a technique used to estimate the apparent motion of objects between consecutive frames in a video sequence. For this project, the Farneback method [1] is employed due to its effectiveness in computing dense optical flow, providing a comprehensive representation of motion across the entire frame. Here's a detailed explanation of this process:

1) Optical Flow Calculation:

- **Farneback Method**: The Farneback method is chosen for its ability to compute dense optical flow, which means it estimates the flow vector for every pixel in the frame. This contrasts with sparse methods like Lucas-Kanade, which only compute flow for a subset of pixels (features) in the frame. Dense optical flow is particularly useful for identifying changes in the waterline, as it provides detailed motion information across the entire image.
- **Flow Vector Calculation**: The optical flow between the initial grayscale frame and the current grayscale frame is computed using the Farneback method. This method approximates the polynomial expansion of the pixel neighborhood to estimate motion. The `cv2.calcOpticalFlowFarneback` function is used with parameters such as the pyramid scale, number of pyramid levels, window size, iterations, polynomial expansion size, and standard deviation. These parameters influence the accuracy and computational cost of the flow calculation.

2) Magnitude and Angle of Flow Vectors:

- **Calculate Magnitude and Angle**: Once the flow vectors are computed, their magnitude and angle are calculated. The magnitude represents the speed of motion, while the angle indicates the direction of motion. The `cv2.cartToPolar` function is used to convert the Cartesian coordinates (flow vectors) to polar coordinates (magnitude and angle).

3) Thresholding Magnitude:

- **Identify Significant Motion Regions**: To focus on significant motion, the magnitude of the flow vectors is thresholded. Pixels with a magnitude above a certain threshold (e.g., greater than 1) are considered to be in regions of significant motion. This thresholding helps to isolate areas where noticeable movement has occurred, such as changes in the waterline.
- **Binary Mask Creation**: A binary mask is created based on the thresholded magnitude. Pixels with significant motion are set to white (255), and all other pixels are set to black (0). This mask highlights regions of the frame where substantial movement is detected.

4) Contour Detection:

- **Find Contours**: The contours in the thresholded magnitude image are found using `cv2.findContours`. Contours represent the boundaries of regions with significant motion. This step helps to isolate the areas where the waterline has moved between frames.
- **Select Maximum Area Contour**: Among the detected contours, the one with the maximum area is selected. This contour is assumed to correspond to the waterline, as it typically represents the largest moving region in the frame.

5) Bounding Rectangle:

- **Bounding Rectangle Calculation**: A bounding rectangle is drawn around the selected contour to determine

its position and dimensions. The `cv2.boundingRect` function is used to compute the rectangle, which provides the coordinates (x, y) and size (width, height) of the contour.

- **Draw Bounding Rectangle**: The bounding rectangle is drawn on the current frame using `cv2.rectangle`. This visual representation helps to identify the location of the waterline within the frame.

6) Visualize Optical Flow:

- **Create HSV Representation**: To visualize the optical flow, an HSV (Hue, Saturation, Value) representation is created. The angle of the flow vectors is mapped to the hue channel, and the magnitude is mapped to the value channel. The saturation is set to maximum (255) to enhance visibility.
- **Convert HSV to BGR**: The HSV representation is converted to BGR (Blue, Green, Red) color space using `cv2.cvtColor`. This conversion makes it easier to display the optical flow visualization alongside the original frame.
- **Resize for Viewing**: The optical flow visualization is resized for easier viewing using the `resize_image_for_viewing` function. This step scales down the image to fit better on the display.

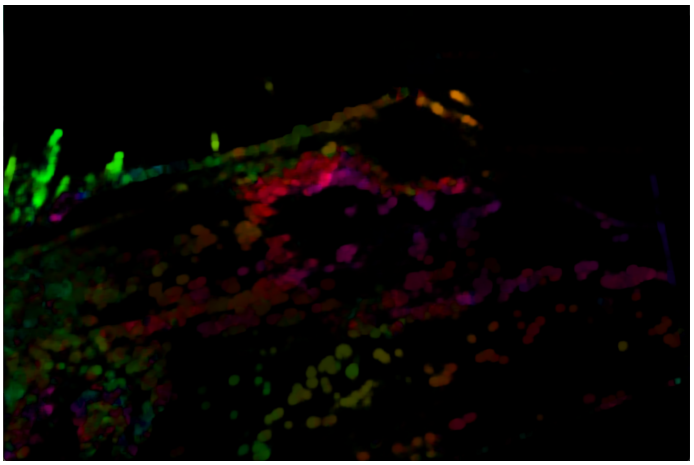

Fig. 5. Optical Flow Image.

D. Measurement of Water Level

The fourth step involves measuring the water level by detecting contours in the thresholded magnitude image. Here's a detailed explanation of the process:

1) Contour Detection:

- **Threshold Magnitude Image**: The magnitude image, which highlights regions of significant motion, is thresholded to create a binary image. Pixels with motion above a certain threshold are set to white (255), while all other pixels are set to black (0). This binary image highlights the areas where the waterline has moved.
- **Find Contours**: Using the `cv2.findContours` function, contours are detected in the thresholded magnitude image. Contours are continuous curves that bound or cover the full boundary of an object in an image. In this case, they represent the boundaries of moving regions in the waterline.

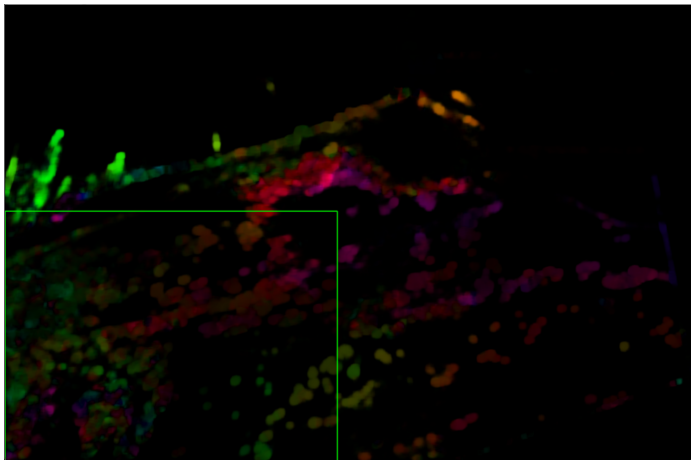

Fig. 6. Optical Flow Image with a bounding box around largest contour

2) Select the Largest Contour:

- **Identify Maximum Area Contour**: Among the detected contours, the one with the largest area is selected. This is based on the assumption that the largest contour corresponds to the waterline, as it usually represents the most significant and continuous movement in the frame.

3) Calculate Bounding Rectangle:

- **Bounding Rectangle Calculation**: A bounding rectangle is drawn around the selected contour using the `cv2.boundingRect` function. This function calculates the smallest upright rectangle that can fully contain the contour. The bounding rectangle provides the coordinates (x, y) of the top-left corner, as well as the width and height of the rectangle.

4) Determine Water Level in Pixels:

- **Extract Y-Coordinate**: The y-coordinate of the top-left corner of the bounding rectangle is extracted. This y-coordinate represents the vertical position of the highest point of the contour, which is used as the water level in pixels.

5) Estimate Water Level:

- **Contour Area**: By analyzing the area and the bounding rectangle of the contour, the height of the water level is estimated in pixels. This height is then converted to real-world units (meters) in subsequent steps.

6) Integration with Draft Marks:

- **Draft Marks OCR**: Optical Character Recognition (OCR) is used to read the numerical draft marks on the hull. Tesseract OCR is applied to sharpened and inverted frames to enhance character recognition accuracy. The detected characters are processed to determine the values of the digits and the 'M' markers, which indicate meters.

- **Calculate Pixel-to-Centimeter Ratio**: The distance in pixels between draft marks is used to calculate a pixel-to-centimeter ratio. This ratio is crucial for converting the pixel height of the water level into real-world units.

7) Final Water Level Estimation:

- **Mapping Pixels to Meters**: The y-coordinates of the detected draft marks are mapped to their corresponding heights in meters using a dictionary. The closest draft mark below the water level is identified, and the remaining distance in pixels is converted to meters using the pixel-to-centimeter ratio. This gives an accurate estimate of the water level.

8) Visualization:

- **Display Results**: The estimated water level is displayed on the frame using `cv2.putText`. This visual feedback helps in verifying the accuracy of the measurements and provides a clear indication of the water level in each frame.

E. Draft Marks OCR

The fifth step in detecting the water level involves using Optical Character Recognition (OCR) to read the numerical draft marks on the ship's hull. Here's a detailed explanation of the process:

1) Image Preparation:

- **Sharpening the Frame**: To enhance the clarity of the numerical draft marks, a sharpening filter is applied to the frame. This is done using a sharpening kernel that emphasizes the edges and details of the draft marks, making them more distinguishable from the background.
- **Inverting the Frame**: The sharpened frame is then inverted to convert the white text on a dark background to black text on a white background. This inversion is crucial for improving OCR accuracy, as Tesseract performs better on dark text against a light background.

2) Applying OCR with Tesseract:

- **OCR Configuration**: Tesseract [2] is configured to focus on numerical digits and the letter 'M', which is commonly used to denote meters in draft marks. Specific configurations are set to optimize the OCR process for these characters.
- **Text Extraction**: Tesseract OCR is applied to the inverted frame to extract the textual information. This includes reading the numerical values and the 'M' markers that indicate the height in meters.

3) Processing OCR Results:

- **Extracting Digit Values**: The text extracted by Tesseract is processed to identify numerical values and 'M' markers. Regular expressions are used to find patterns that match digits and 'M' markers.
- **Identifying Locations of 'M' Markers**: The bounding boxes returned by Tesseract provide the coordinates of the detected characters. The locations of the 'M' markers are identified and stored for further processing.

4) Handling Multiple 'M' Markers:

- **Selecting the Relevant 'M' Marker**: In cases where multiple 'M' markers are detected, the one corresponding to the lowest digit value is selected. This helps in ensuring the correct interpretation of the water level.

5) Estimating Water Level from OCR:

- **Final Water Level Calculation**: The numerical draft marks and 'M' markers are combined to form the final water level estimate. This involves interpreting the digit values correctly and ensuring they represent the height in meters accurately.

6) Visualization:

- **Displaying Results**: The estimated water level is displayed on the frame using `cv2.putText`. This visual feedback helps in verifying the accuracy of the OCR process and provides a clear indication of the water level in each frame.

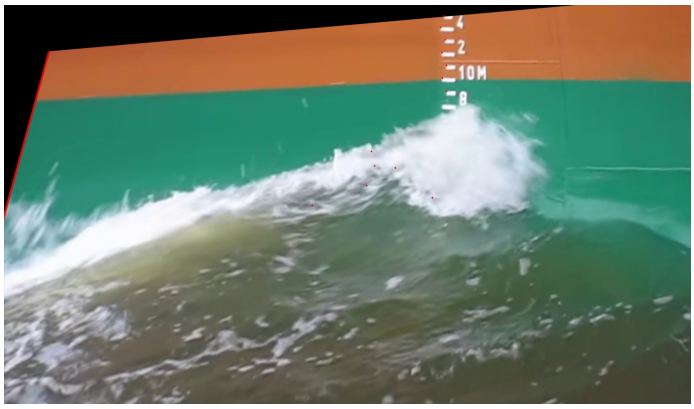

Fig. 7. Markings of all the OCR characters recognized

F. Calculating the Water Level

The final step involves using the detected draft marks and the measured waterline height to calculate an accurate estimate of the water level. Here's a detailed breakdown of this process:

1) Mapping Draft Marks to Heights:

- **Dictionary Initialization**: A dictionary is created to map the y-coordinates of the draft marks to their corresponding height values in meters. This mapping helps in converting the pixel coordinates to real-world measurements.
- **Generating Height Values**: Height values are generated starting from 10.5 meters, decrementing by 0.1 meters for each subsequent draft mark. These values represent the actual water levels marked on the hull.
- **Mapping Heights to y-Coordinates**: The generated height values are assigned to the corresponding y-coordinates of the draft marks. This creates a direct mapping between the pixel locations and the actual water levels.

2) Estimating the Water Level:

- **Extracting y-Coordinates**: The y-coordinates of the draft marks are extracted into a list for easy comparison.
- **Finding the Nearest Draft Mark Below the Waterline**: The closest draft mark below the current waterline is found by comparing the waterline height with the y-coordinates of the draft marks.
- **Calculating the Remaining Distance**: The remaining distance from the waterline to the nearest draft mark is calculated in pixels. This distance is then converted to meters using the average pixel-to-centimeter conversion factor (`pixels_to_cm`).
- **Estimating the Height**: The height of the water level is estimated by adding the converted distance to the height value of the nearest draft mark. This provides an accurate measurement of the water level in meters.

G. Visualization and Post-processing

The final stage involves processing the estimated water levels to eliminate anomalies and visualizing the results. Here's a detailed explanation of the steps involved:

1) Storing Estimated Water Levels:

- **Storing Values**: Throughout the frame-by-frame processing, estimated water levels are stored in a list for further analysis.

2) Post-processing to Eliminate Anomalies:

- **Calculating Mean and Standard Deviation**: The mean and standard deviation of the estimated water levels are calculated. These statistics help identify and remove outliers.
- **Defining Bounds**: Boundaries for valid water levels are set to be within two standard deviations from the mean. Values outside these bounds are considered anomalies and are discarded.
- **Filtering Outliers**: The estimated water levels are filtered to remove values that fall outside the defined bounds. This ensures that only reliable measurements are used in the final analysis.

3) Calculating the Final Water Level:

- **Averaging the Final Water Levels**: The remaining valid water levels are averaged to provide a single, reliable measurement of the water level.

4) Visualizing the Results:

- **Creating the Plot**: Matplotlib is used to create a scatter plot that shows the estimated water levels over time. The x-axis represents the frame number, and the y-axis represents the water level in meters.
- **Displaying the Plot**: The plot is displayed to visualize how the water level changes over time. This graphical representation helps in understanding the consistency and reliability of the estimated water levels.

Because every step of the procedure is carried out on pre-recorded video frames, the method is reliable and accurate in a variety of scenarios. This method offers a dependable and automated way to detect water levels from ship hulls by utilizing optical flow and OCR.

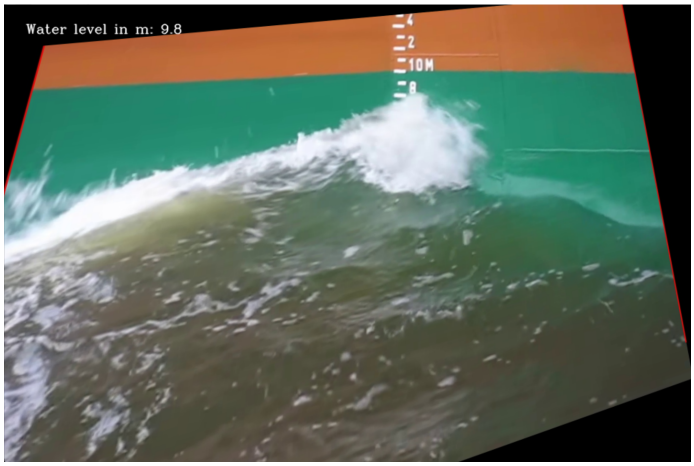

Fig. 8. Final water level computed from frame = 9.8 m

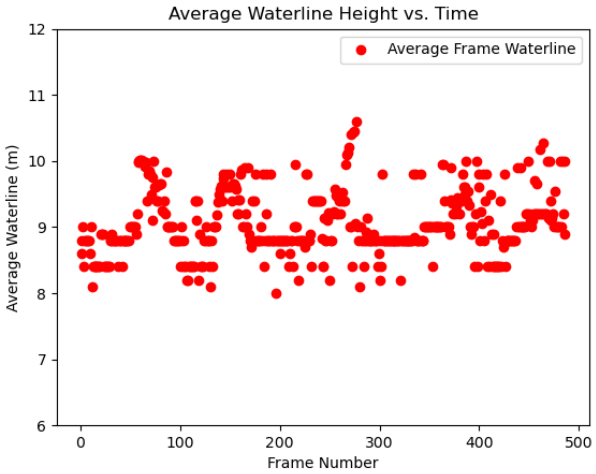

Fig. 9. Plot Average Water Line Height vs Frame Number.

III. Conclusion

The suggested approach of combining optical flow and OCR techniques to detect water levels from ship hulls turned out to be highly successful. We were able to precisely track the movements of the waterlines throughout the video frames by utilizing the Farneback method for dense optical flow. The Homography transformation made the OCR process more dependable for extracting numerical values by guaranteeing that the draft marks were consistently aligned.

The method produced a reliable water level estimate even under variable lighting and ambient conditions. Nevertheless, there were times when we missed a frame and the water level wasn't detected for that frame. This emphasizes how crucial high-quality video is to guaranteeing reliable and consistent measurements. An error margin of about 0.1 meters was noticed in some of the frames, which is attributed to things like motion blur or dim lighting.

The accuracy of our measurements was further improved by the post-processing step that eliminated outliers. The water

level readings over time are displayed in the graph that our results produced, highlighting the accuracy and consistency of our approach.

All things considered, the integration of optical flow, image processing, and OCR produced an accurate and efficient system for water level detection from previously recorded ship hull videos. There are numerous uses for this approach in maritime monitoring and analysis that can be expanded upon and customized.

REFERENCES

[1] Färneback, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Scandinavian Conference on Image Analysis* (pp. 363-370). Springer, Berlin, Heidelberg.

[2] Smith, R., Antonova, D., & Lee, A. (2009). Adapting the Tesseract open-source OCR engine for multilingual OCR. In *Proceedings of the International Workshop on Multilingual OCR* (pp. 1-8).

[3] Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc.