ENPM 673 Spring 2022
Project 4

Water Level Detection for a Ship Hull

Sri Sai Charan V (UID: 117509755)
Vaishanth Ramaraj (UID: 118154237)
Pranav Limbekar (UID: 118393711)
Yash Kulkarni (UID: 117386967)
Jerry Pittman (UID: 17707120)

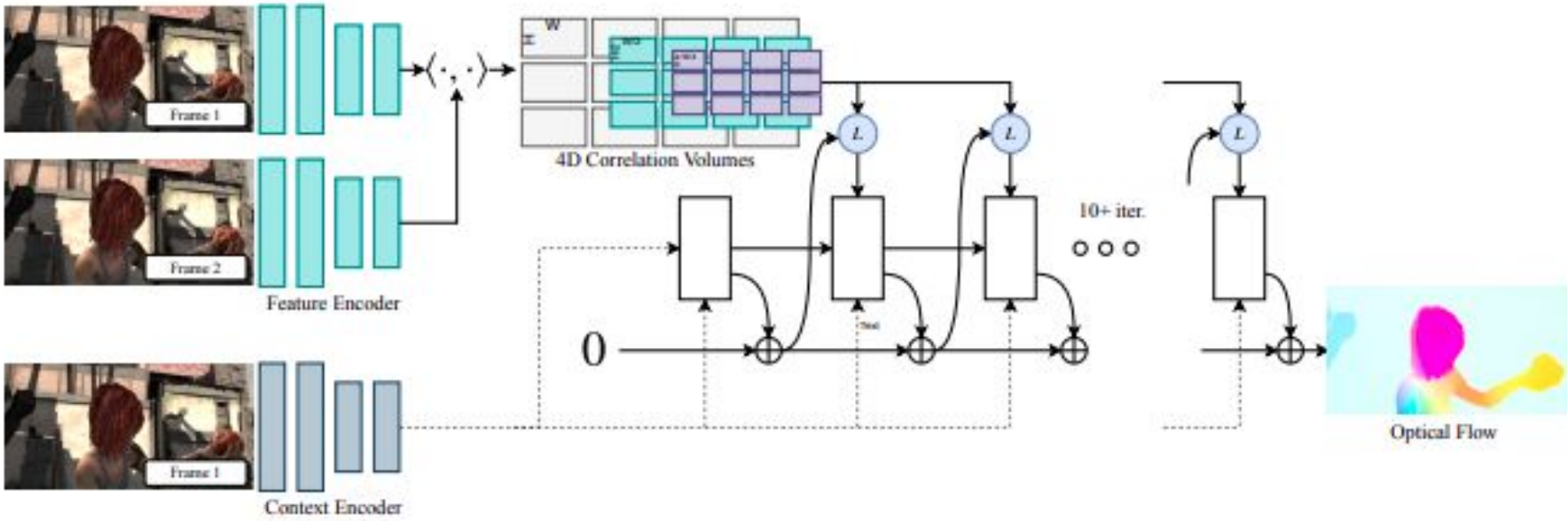
Problem to Solve:

- The traditional ship draft detection method uses an artificial observation method. This method is to obtain the ship's draft value by observing the waterline floating situation for a period of time. Although this method is simple, it is greatly affected by objective factors such as visibility, weather, and water waves, so the measurement error is large, and the accuracy and reliability are reduced. In order to improve the accuracy of ship waterline detection, there have been many ship waterline detection methods, such as sensor-based methods and machine vision-based methods.
- We will be using a machine vision based method for the detection of water level for the hull.

Background Material/ Literature Review

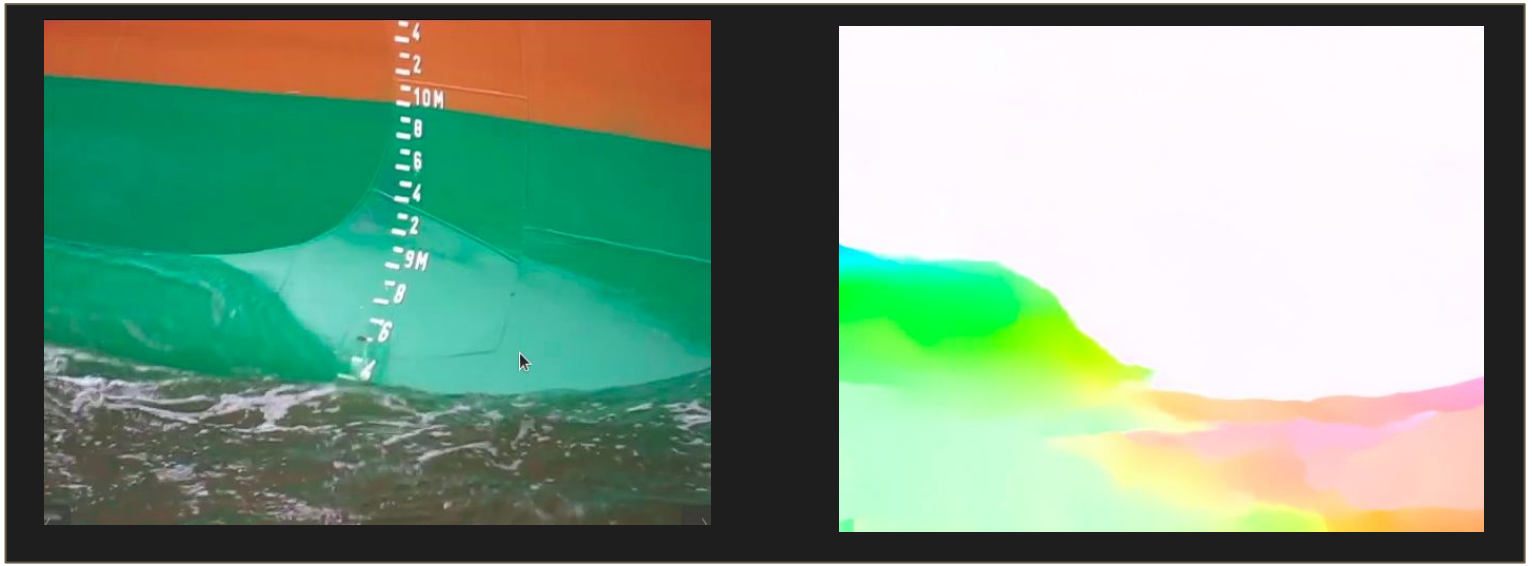
- Wang, Zhong Shi, Peibei Wu, Chao. (2020). A Ship Draft Line Detection Method Based on Image Processing and Deep Learning. Journal of Physics: Conference Series. 1575. 012230. 10.1088/1742-6596/1575/1/012230.
- Pathak, D., Sarangapani, R. N., Katare, D., Gaikwad, A. S., El-Sharkawy, M. (2018). IOT based solution for level detection using CNN and OpenCV. Athens: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). Retrieved from https://www.proquest.com/conference-papers-proceedings/iot based-solution-level-detection-using-cnn/docview/2139488678/se-2?accountid=14696
- Steccanella, Lorenzo Bloisi, Domenico Castellini, Alberto Farinelli, Alessando. (2019). Waterline and obstacle detection in images from low-cost autonomous boats for environmental monitoring. Robotics and Autonomous Systems. 124. 10.1016/j.robot.2019.103346.
- Water level detection python opencv(image-processing): https://www.youtube.com/watch?v=ect8xAhwrD4
- EAST Text Detection: https://pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
- RAFT Git: https://github.com/princeton-vl/RAFT
- RAFT Paper: https://arxiv.org/pdf/2003.12039.pdf
- VGG16 model on Pytorch: https://pytorch.org/vision/stable/models.html

Recurrent All-Pairs Field Transforms (RAFT)

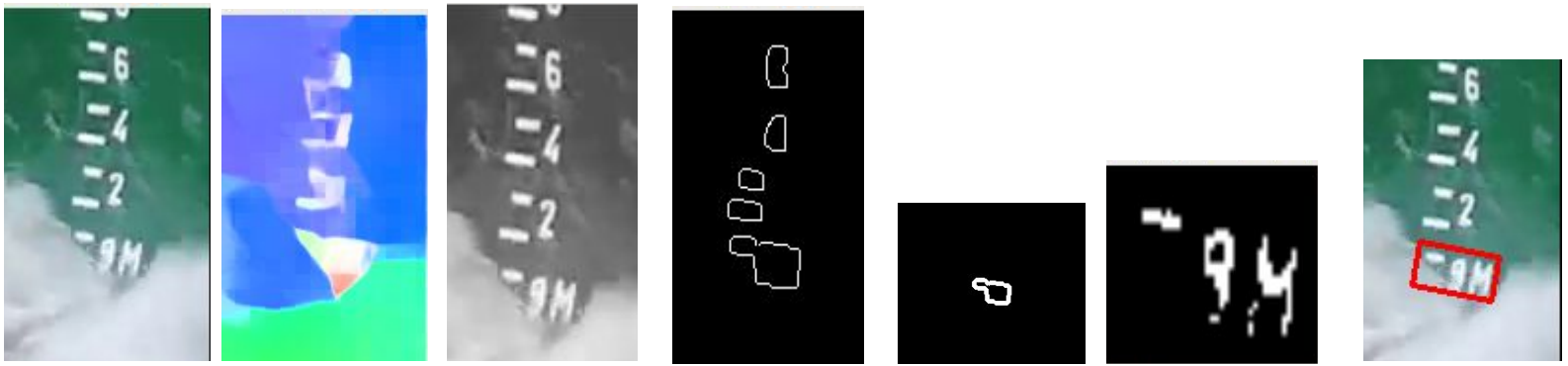


Source: https://arxiv.org/pdf/2003.12039.pdf

Recurrent All-Pairs Field Transforms (RAFT)



Computer Vision Pipeline



Test Image → RAFT → Grayscale/ Threshold → Morphology/ Canny edge detection → Find contours → Homography → CNN → Charactor recognition → Draft depth

Implementation

```
while True:
    success, current_frame = vid_cap.read()
    crop = np.zeros_like(current_frame)
    crop[:,180:350] = 255
    current_frame = cv.bitwise_and(crop,current_frame)

    if not success:
        break
    temp_img = current_frame.copy()
    cv.imshow("Source",current_frame)

    current_frame = load_image(current_frame)

    padder = InputPadder(current_frame.shape)
    image1, image2 = padder.pad(current_frame, prev_frame)

    flow_low, flow_up = model(image1, image2, iters=20, test_mode=True)
    result = viz(image1, flow_up)

    prev_frame = current_frame
    result = result*255
    heat_map_img = result.astype(np.uint8)

    orginal_gray = cv.cvtColor(temp_img, cv.COLOR_BGR2GRAY)
    gray_img = cv.cvtColor(heat_map_img, cv.COLOR_BGR2GRAY)
    ret,thresh_img = cv.threshold(gray_img , 230 ,255,cv.THRESH_BINARY)

    masked_img = cv.bitwise_and(orginal_gray,thresh_img)
    crop = np.zeros_like(masked_img)
    crop[:,180:350] = 255
    masked_img = cv.bitwise_and(crop,masked_img)

    _ , masked_thresh = cv.threshold(masked_img , 230 ,255,cv.THRESH_BINARY)
    kernel = np.ones((7,7),np.uint8)

    masked_thresh_gradient = cv.morphologyEx(masked_thresh, cv.MORPH_GRADIENT, kernel)
    masked_thresh_gradient_canny = cv.Canny(masked_thresh_gradient,100,200,2)
```

```
contours, hierarchy  = cv.findContours(masked_thresh_gradient_canny, cv.RETR_TREE, cv.CHAIN_APPROX_NONE)
    blank_testing = np.zeros_like(temp_img)
    avg_arc_length = []
    for i in contours:
        if 300 > cv.arcLength(i,False)>250:
            cv.drawContours(blank_testing, [i], -1, (0,255,0), 3)
            avg_arc_length.append(cv.arcLength(i,False))
            cv.imshow("mid_drawing",blank_testing)
            print(cv.arcLength(i,False))

            rect = cv.minAreaRect(i)
            box = cv.boxPoints(rect)
            print(box)
            box = np.int0(box)
            cv.drawContours(temp_img,[box],0,(0,0,255),2)
            warp_image(box,masked_thresh)
            cv.waitKey(0)

        if 250> cv.arcLength(i,False) > 110:
            cv.drawContours(blank_testing, [i], -1, (255,255,255), 3)
            avg_arc_length.append(cv.arcLength(i,False))
            cv.imshow("mid_drawing",blank_testing)
            print(cv.arcLength(i,False))
            rect = cv.minAreaRect(i)
            box = cv.boxPoints(rect)
            print(box)
            box = np.int0(box)
            cv.drawContours(temp_img,[box],0,(0,0,255),2)
            warp_image(box,masked_thresh)
            cv.waitKey(0)
    cv.namedWindow("Masked_thresh",cv.WINDOW_NORMAL)
    cv.imshow("Masked_thresh",masked_thresh)
    cv.imshow("blank_testing",blank_testing)
    cv.imshow("temp_img",temp_img)
    if cv.waitKey(0) == ord('q'):
        break
vid_cap.release()
cv.destroyAllWindows()
```

Implementation

```
[12] from IPython.display import Image

     Image("2.png", width=100, height=100)
```

```
image=cv2.imread("2.png")
image=cv2.resize(image,(28,28))
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ar = array(gray)
ar =~ ar
ar = ar / 255
ar = ar.reshape(1,28,28)
for i in range(0,28):
  for j in range(0,28):
    if ar[0][i][j] > 0:
        ar[0][i][j] = 1
```

```
[15] p = model(ar).numpy()
     p

     array([[-150.51517  , -193.78362  ,    3.0387144,  -43.680084 ,
             -358.52463  ,   37.407387 ,   27.430658 ,   -2.5162482,
             -208.78337  , -287.9695   ]], dtype=float32)
```

```
[16] output = tf.nn.softmax(p).numpy()
     output

     array([[0.0000000e+00, 0.0000000e+00, 1.1853741e-15, 6.0832909e-36,
             0.0000000e+00, 9.9995351e-01, 4.6466688e-05, 4.5852822e-18,
             0.0000000e+00, 0.0000000e+00]], dtype=float32)
```

0s completed at 7:57 PM

Implementation

```
output = tf.nn.softmax(p).numpy()
output
```

```
array([[0.0000000e+00, 0.0000000e+00, 1.1853741e-15, 6.0832909e-36,
        0.0000000e+00, 9.9995351e-01, 4.6466688e-05, 4.5852822e-18,
        0.0000000e+00, 0.0000000e+00]], dtype=float32)
```

```
[17] max = 0
     for i in range(0, 10):
            if output[0][i] > max:
                max = output[0][i]
                number_prediction = i
```

```
[18] print("The Number detected was :", number_prediction )
```

```
The Number detected was : 5
```

Results & Conclusions:

- Taking frames from video and sending to Resnet and VGG16 and comparing the results then will use the one with the best results.

Lessons Learned

- Real-time image processing using Optical Flow (especially using RAFT) is very computationally heavy and requires at least GPU for processing.
- When the water from a wave recedes, the optical flow is affected and so we needed to skip frames and use a frame from the first 6 for optical flow comparison for later frames due to the entire image being washed out.
- You cannot simply send in the masked image to a CNN for OCR since it will detect all numbers and so you need to send just the number/marking just above the water mask.
- The hyphen of the drafts confuses some OCRs.
- Using pre-trained models for optical flow and OCR saves time but risks accuracy based on what the models were trained on since they are typically not trained on our application of ship hull. (i.e. animals, chairs, people, handwritten numbers, traffic signs in good weather)

THANK YOU