Man over board

Anna Samoylenko, *a.samoylenko@student.utwente.nl, s3026914, M-BE*,
Diana Canosa Tajes, *d.canosatajes@student.utewente.nl, s3104443, M-BE*
Akram Sabik, *s.akram@student.utwente.nl, s3095681, M-CS*

***Abstract*—This paper addresses the problem of tracking man overboard and measuring distance from the observator to him. As a model for the man overboard white buoy was used. Camera calibration and stabilization, as well as template matching and motion modeling were thoroughly studied and used for tracking, and horizon coordinates were obtained and used for distance evaluation. As a result of our work tracking and distance evaluation throughout the whole test video were performed and validated, results discussed and disadvantages of the methods observed.**

***Keywords*—Camera Calibration, Distance estimation, Template Matching, Object Tracking, Video Stabilization**

I. Introduction

"MAN overboard" is a situation where someone from the ship falls out at sea. In such circumstances it is very important to keep track of the person to be able to rescue him, but it is not always an easy task. Even though the person can be near the rescue boat with his coloring life jacket on, the movement of the camera, together with the heavy waves and the movement of the person himself can really complicate the job. Problem of man overboard event detection has been addressed in many research papers, such as [1-3], however, there is almost no research works dedicated to man overboard tracking, even though necessity for such applications is seen by society [4]. The aim of this project is to analyze some of the problems faced by tracking algorithms, using a test video of a buoy that is the replacement for a human in the sea. For doing that, the first requirement is the detection of the object, followed by the stabilization of the movements of the camera. After this preprocessing step of the image, the next step is keeping track of the object and estimating its distance from the rescue vessel.

In this report, the methods used to address the problem are explained, the obtained results and its interpretation are also presented.

II. Methods and Materials

A. Materials

MATLAB R2022b: Image Processing and Computer Vision Toolbox.
Chessboard calibration images, square size of 40 mm. Fig.2
MAH01462.mp4 video.
Ut-plot-lens-distortion.m.
Distance from camera to the sea.

Fig. 1: First frame from 'MAH01462.mp4'

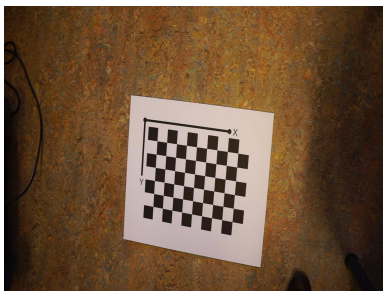

Fig. 2: Camera calibration image (26 in total)

B. Methods

1) Analysis: The idea is to create a system that is able to track a person in real time. For creating the program and testing it, a sampling video from the rescue boat is needed (Fig. 1). From the first frame where the buoy is visible, the user has to manually select the position of the buoy and the horizon because the program is not going to detect the position of the person automatically. Having these 2 inputs, the program is going to stabilize the frames and track the buoy automatically, calculating its position in the image in pixels and its distance from the camera in meters.

2) Camera calibration: Camera calibration is required to know the intrinsic camera parameters. These parameters are needed to remove the nonlinear distortions of each frame, using the function `undistortimage.m` , before continuing with the stabilization part. It is important to rectify these distortions to obtain better results in the stabilization part as we are going to use key points that can be in the corners of the image, where the distortions are more important. It is also required to know the intrinsic camera parameters to create a geometric model that allows the calculation of the distance from the rescue boat to the buoy, which will be explained later. Camera calibration was done using the `MATLAB` app `CameraCalibrator` from the Image Processing and Computer Vision Toolbox. This app

uploads the provided calibration images of the chessboard and allows users to select different features or remove outlier images that increase the mean error. A mean error of 0.12 pixels was finally the best approach, removing one image.

3) Video Stabilization: The goal is to stabilize the video MAH01462.MP4 2 different methods were used to stabilize the video. Matching points approach and target tracking approach.

Target tracking video stabilization works by defining a target to track, establishing a dynamic search region. The target is only tracked in that region. By knowing how the target moves relative to the previous frame, we generate a stabilized video. In our case the target to track was a rectangle towards the mountain. Several targets and ROI sizes were experimented with such as clouds and the horizon. (Fig. 3)

The result is satisfying, but toward the second half of the video, the target to track is lost and the region of interest starts to drift away.

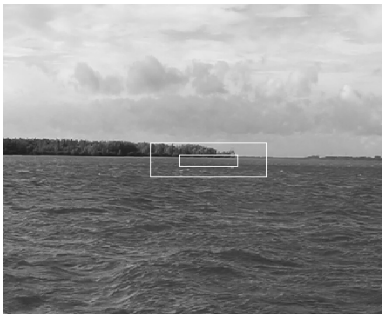

Fig. 3: Target tracking video stabilization ROI

Matching points video stabilization works by finding key points between each frame, extracting and matching descriptors, finding the affine transform and finally warping the image.

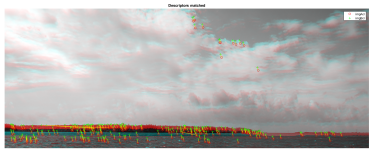

Fig. 4: Matching points between the first 2 frames of the video (FASTFeatures), in red-cyan color composite. Matching points are mainly taken from the horizon, mountains and clouds

An affine transformation has a limited capacity to represent a 3d projection distortion between 2 images. Usually 8 parameters and warps (projective transformation) would be needed to do that. However, in the context of the pr oject, we assume that the background plane has not moved or changed significantly between frames as it is far away from the camera. We also decide to not take into account the ocean's motion as it can't be relied upon to stabilize the camera because it is too close to the camera. This condition is verified if the motion between frames is small enough or if the frame rate is high enough. Following this assumption, it is considered that the affine transformation between frames captures the camera motion. Therefore correcting for this will stabilize the video.

Firstly, salient points are collected from each frame using `detectFASTFeatures`, with a handpicked threshold. The points are only collected in a defined rectangle focusing on the horizon, land, and clouds as the ocean can't be relied upon to stabilize the camera. In Fig 4 we can see the result of the descriptors selection.

The MSAC algorithm is used to search for valid inlier correspondences for each pair of frames with the `MATLAB` function `estgeotform2d`. This allows for a better stabilization performance. The transformation computed between consecutive frames is a similarity and is composed of scale, rotation and translation. Therefore, the transformation only consists of 4 free parameters. Affine transformation was also tested. It consists of 6 free parameters as it introduces shearing. However, this made the stabilized video unusable after a certain number of frames. We realized that affine transformation were adding a shearing effect so we should change to similarity transformation to remove the shearing component.

Algorithm 1 obtaintform.m

Require:
imgAcr (fixed part of image A)
imgBcr(fixed part of image B)
ptThresh (treshold to detect key points)
Ensure:
tform to warp imgBcr towards imgAcr

1. detect key points in imgAcr and imgBcr using detectFASTFeatures.m
2. select the descriptors for each image using extractFeatures.m
3. match descriptors between images using matchFeatures
4. return tform using estgeotform2d.m (similarity transform

4) Tracking: 1.4.1 Template

For the purpose of tracking method of template matching was chosen as it seems less computationally expensive then key points detection. Several methods of constructing template were built in order to pick the best one. The first attempt was to create template manually. For that purpose approximate size of the buoy and its intensity and intensity were found on the picture. After that template with the size of 6x6 was created. However, it appeared to be, that such small size of the template doesn't allow to distinguish buoy from the waves when it is not clearly seen. That is why the second step was expanding borders to 10x10 so that the mistake for matching with anything with not round shape would be big enough Fig.5.a. This template was tested with the whole program and worked well. (function for template creation: function1: templatefunc.m)

However, we also tried to build template automatically. For that purpose after getting first coordinates of the buoy, threshold was applied to the neighborhood area 10x10, where pixels with intensity smaller than the threshold gained medium intensity of waves on the template and ones with intensity

bigger than the threshold gained intensity of 1 on the template (Fig.5.b and Fig.5.c). Fig.5.b shows template, constructed from the first frame, on which the buoy is somehow visible. However, quite oftenly waves have the same intensity as the buoy, that is why we see part of the waves' shape on the template. Also shape of the buoy is lost (it is not round anymore). Fig.5.c shows the template created for a frame, on which the buoy is clearly visible. Nevertheless, even in such perfect conditions part of the waves still interfere into the template. Even though templates from Fig.5.b,c performed not very bad, there still were several events of buoy loss. Creation of dynamic template was not compatible with other approach, used in the projects.

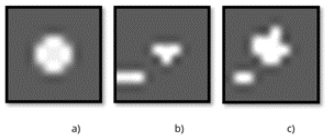

Fig. 5: Template models for the buoy

Taking into account everything mentioned above, the first approach to template creation was chosen for future use in the code.

Algorithm 2 templatefunc.m

Require:
size (side size of the template)
Ensure:
t2 (matrix with template values)

1. Manually assign values for each pixel in t2 based on buoy image values.

1.4.2 Position detection

The main idea of the tracking used in this project is to locate the buoy within the ROI and move the ROI with the movement of the buoy.

For template matching inside the ROI vision.TemplateMatcher Matlab function was used. Sum of square difference was used to calculate loss function, as it proved to perform better than Sum of absolute differences or Maximum absolute difference. ROI was defined as 20x20 matrix, so as to be large enough to compensate movements of the buoy, but small enough not to cause many false detections or big computations(Fig.6).

After finding estimated position of the buoy "distance" between found area and template is calculated. For this purpose equation [1] is used:

$$(1)\ d_c(p,q) = \sum_{n=1}^{N}\sum_{m=1}^{M} z^2(n-p,m-q)$$
$$+\sum_{n=1}^{N}\sum_{m=1}^{M}\alpha_c^2(n,m)$$
$$-2\sum_{n=1}^{N}\sum_{m=1}^{M} z(n-p,m-q)\cdot\alpha_c(n,m)$$

where z - found area, c- template.

Fig. 6: ROI on the image

That is done in order to determine if buoy is visible or not. If dc exceeds certain threshold (in our case it is 3, then buoy is not visible and found position can't be treated as the right one. However, if the position is actual, than center of the ROI moves to this position.

1.4.3 Movement of the buoy in the "blind zone"

The only question that is left: What will happen, if buoy is not visible? Apparently, as the sea, camera and buoy itself move, it will appear not in the same position, as it was before. In order to predict the next appearance of the buoy and move ROI to this position movement model was created.

Due to the previous stabilization of the video, as well as bearing in mind that optical flow in the upper part of the picture is close to linear, it was decided to use 1st order polynomial fitting. Movement of the ROI in the "blind zone", where buoy is not visible is based on the movement of the buoy, when it is still visible. Movement of the boy is determined as change in its position in x and y directions.The number of frames in the window on which motion of buoy is approximated was taken 10, so it would model local motion of the target, but not the average one. Even though initially 1st order polynomial curve fitting was used to approximate motion of the buoy, due to the fact that on stabilized video buoy doesn't change its direction rapidly, it appeared to be that calculating the mean of the position changes works just as good and less computationally and timely expensive.

The first attempt to predict ROI motion within the "blind zone" was to calculate an optical flow in the area of the buoy. Nevertheless, as the buoy moves slower than the sea it wouldn't work. Applying the same idea to the land didn't change the situation either. Apart from that finding optical flow is very computationally expensive and wouldn't be good enough for online applications.

All in all, the tracking part can be explained in next code:

5) Distance calculation: As we are working with only one camera , we need a reference from the real word to estimate the distance. In this case, the horizon was the reference. The program can be modified to use another known distance from the sea level to the camera, for example if we are in a lake and we know the distance from the camera to the border. Taking into account this consideration, a geometrical model was created, (Fig. 7) From this model we obtain several equations (in appendix A) that relate distance in pixels with

Algorithm 3 Tracking algorithm

Require:
stabilized first frame
initial position of the ROI
Ensure:
positions of the buoy

Manually pinpoint position of the buoy on the first frame
while frame left in the video **do**
Get new video frame without
Transform it and get rid of distortion
Define position of the buoy inside the ROI by template matching
Calculate correlation between the template and part of the image where buoy is found
if lager than the predefined threshold **then**
Move center of the ROI to the newly found position of the buoy
Calculate x and y components of velocity of the buoy in pixels per frame by finding difference in coordinates between this and previous frames
else if **then**
Find a mean velocity of the buoy for the last 10 frames
Move ROI in accordance with found velocity
end if
end while

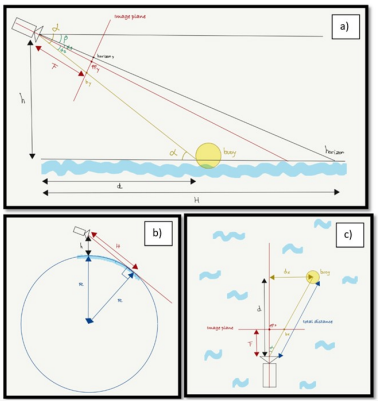

Fig. 7: Geometrical model using horizon as reference distance from real world to calculate relationship between pixel distance and real distance. Image b) shows the relationship between distance from camera to sea, h, the radius of the Earth, R, and the distance to the horizon, H. Image a) shows a vertical vision of the position of the camera, the buoy, the horizon and their projection in the image plane; ppy, by and horizony respectively. Focal distance, F, distance to the buoy, d, and distance h and H are also represented. Image c) shows a horizontal vision of the position of the camera, the buoy and their projections in the image plane, pp and, bx. Distance from the optical axis to the buoy, dx, distance d and total distance from the camera to the buoy are also represented.

real world distance. From Fig.7 b) we obtain the distance from the camera to the horizon (H). From Fig.7 a) and the H,we can calculate angle /betta. This angle was so small that it was neglected but it has to be taken into account when using a different real world reference. Knowing the intrinsic parameters of the camera, focal distance and principal point, from camera calibration, and the buoy position from the tracking part, we can know /alpha and therefore the distance d. Once we know d, using Fig. 7 c) it is easy to obtain the distance in the x direction and therefore the total distance from the camera to the buoy. We create Algorithm 3 to perform these calculations for each frame.

6) Performance evaluation: For evaluate the performance we did several experiments, to validate different aspects of the program and also to evaluate the robustness.

6.1 Validation

Matching points video stabilization performance is evaluated by comparing the mean frame of the unstabilized video and stabilized video.

Definition of the position of the buoy in the picture is a difficult task for the computer. Usage of the neural network might be required, but even in this case it won't give a reliable result. Human perception seems to be a much better evaluator. However, we can't just say that our program tracks the buoy throughout the whole image. Some kind of mathematical evaluation should be used. From the program we have detected coordinates of the buoy. In order to verify their accurateness we should find the position of the buoy on the same frames

Algorithm 4 calculate total distance.m

Require:
pp (camera principal point coordinates in pixels)
buoy (buoy coordinates in pixels)
Horizon (row of the horizon position in pixels)
F (camera focal length in pixels units of x and y)
Ensure:
distance from the camera to the buoy

1. calculate distance along z axis
Assuming beta angle is very small
if pp(y) = buoy(y) **then**
alpha = alpha1
else if pp(y) $<$ buoy(y) **then**
alpha1 + alpha2
else if pp(y) $>$ buoy(y) **then**
alpha1 - alpha2
end if
2. calculate distance along x axis
3. return total distance

ourselves. In order to do that, the distance between those two positions are found for each 20th image. For those images, for which buoy is not visible, distance does not influence the final result. The resulting measure of accuracy is a mean distance for verified frames [2].

$$sres = \frac{1}{num(fr)} \cdot \sum_{n=1}^{num(fr)} \sqrt{(m(1) - a(1))^2 + (m(2) - a(2))^2} \quad (2)$$

where sres - average distance between found and factual coordinates of the buoy, m(1,2) - coordinates of the buoy which were found manually, a(1,2) - coordinates of the buoy which were found automaticly, num(fr) - number of frames that were involved in the evaluation.

For the validation of distance evaluation no mathematical approaches can be applied, as we lack prior knowledge of the distance to the buoy. However, to make sure that the program works correctly at least from a logical point of view, the relationship between evaluated distance to the buoy and number of frames will be found. These results will be compared with the visible direction of movement of the camera.

6.2 Robustness

The idea of robustness evaluation was to try this program on another video/videos to see how it will adapt to a bit different conditions . However, due to a significant amount of restrictions, which will be mentioned later, the task of finding appropriate online video appeared to be too difficult. This is why we decided to record a video to evaluate robustness. However, the template we created for the program is not the best for the video we recorded, so finally the program would not work. In the new video the water has a higher intensity than the object, so the program does not detect the object using the same template. Camera calibration of the new camera was also needed to undistort frames and calculate distance but as the program cannot word for other issues, this step was not performed.

III. Results

As a result of tracking we got a video on which the buoy can easily be found inside the red square and/or an increased image of the area in which the buoy is located. Video and buoy itself moves smoothly, though black borders on frames can be seen.

After performing manual pinpointing of position of the buoy, approximated error appeared to be from 1.3 to 2.8 pixels for different trials. As for the distance evaluation, correlation between distance to the buoy and number of frames was found (Fig.8). From the visual approximation it is clearly seen that boat is shortening distance to the target with time. The same trend we see on the graph below.

The results of the camera calibrations and stabilization are shown on Fig.9 and Fig.10 respectively.

As a result, according to the calculations, the camera moved towards the buoy on 88 meters (almost half of the distance). Average relative speed to the buoy is 0.2 meters per frame.

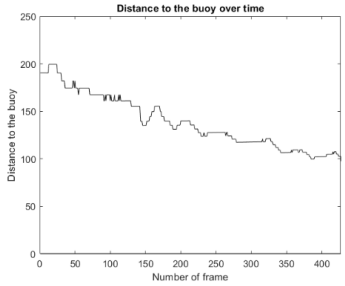

Fig. 8: Distance evaluation of the buoy

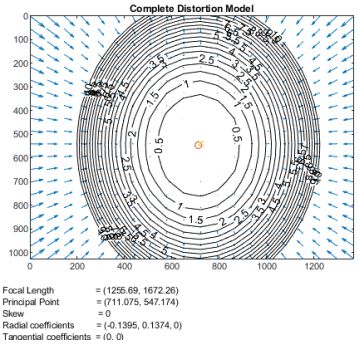

Fig. 9: Distortion model created from camera calibration results, using the function ut-plot-lens-distortion. We can see the intrinsic camera parameters and the plot of radial distortion with a maximum error of 10 pixels at the borders

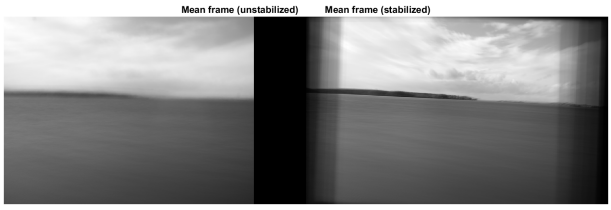

Fig. 10: Camera stabilization performance. The horizon is much more clearer on the right image

IV. Discussion

Even though the main purpose of the project was creating a program that can track the buoy throughout the video and calculate distance to it, the initial “idea” of this task was to create an app, which can be used to track people overboard. As we can see on the video, our program indeed does its main purpose and tracks the buoy, as well as calculates distance to it. As we have shown earlier, accuracy of detection of the buoy on the frame is very high (around 2 pixels), which is a very good result, as picking points manually itself leads to an error caused by a human factor. That is why this program can be regarded as very accurate from a tracking point of view. However, there are quite a lot of restrictions, which don’t allow this program to be used in applications for tracking

man overboard. The first problem is connected to the fact that our template is manually built - it has an ideal round shape and background filled with pixels with intensities close to the sea waves. That means that for an object with non-round shape or lower intensity this template wouldn't fit. However, we think that a human head is close to the buoy's feature (if it is turned faceward to the camera). In order to make this application more autonomous some kind of object recognition may be required to automatically find this target in the sea and create the template accordingly. The second disadvantage comes from the stabilization method itself. In order to stabilize the horizon we applied a similarity transform to the upper part of the image (so just a small amount of water falls in this region). And, again, the height of this region is picked manually. However, for the means of autonomization in an app horizon should be found automatically and, if required, turned into horizontal position. The third problem is the definition of the threshold which shows whether the object is visible or not. In our case it was found by analyzing video frames and values of the correlation on the same frame, so that the required value was gained when buoy changed its state from visible to invisible. The only solution for autonomization is that case, as we see, is only an approximation of this threshold from a significant number of similar videos. One more obstacle to using this program in a tracking application is the fact that developed motion models work only in the far distance, where motion of the buoy is relatively easily described as linear. On smaller distances waves bring a lot of additional motions, which makes our program helpless in tracking the buoy when it is not visible. However, on those distances, such applications wouldn't probably be required. Apart from that, the biggest minus of this program is its slowness, which doesn't allow it to be used in online applications (it took the program 167 seconds to go through all video frames, where video itself lasts only for 16 seconds). It may be somehow connected to the fact that Matlab doesn't support video streaming. However, it may be due to the non-optimal code structure, even though we did our best to optimize it and used less computationally expensive functions. One other useful thing may be adding a possibility to determine on the small scaled image (left down corner of the video) whether the person is conscious or not, so to have an idea how much time savers have (or don't have).

V. Conclusion

To sum up, even though the program developed throughout this research has a lot of implementational restrictions, the main methods proved themselves to be applicable for tracking objects overboard. Stabilization, as well as tracking algorithms and distance measure require further generalization in order to be able to perform their functions in different conditions. Nevertheless, each part camera calibration, stabilization, tracking and distance measure was able to perform its functions perfectly throughout the whole test video.

Appendix A
Equations from geometrical model

$$H = \sqrt{(R+h)^2 - R^2} \tag{1}$$

$$\beta = \arctan\frac{h}{H} \approx 0 \tag{2}$$

$$\alpha_1 = \arctan\frac{|h_y - pp_y|}{F_y} \tag{3}$$

$$\alpha_2 = \arctan\frac{|b_y - pp_y|}{F_y} \tag{4}$$

$$d = \frac{h}{\tan(\alpha)} \tag{5}$$

$$d_x = d\,\frac{|b_x - pp_x|}{F_x} \tag{6}$$

$$total\ distance = \sqrt{d^2 + d_x^2} \tag{7}$$

Appendix B
Matlab scrip (main code and functions created)

```
% main.m
clc;
clear all;
close all;

filename = 'MAH01462.MP4';
video = VideoReader(filename);

%% From camera calibration

% Use camera calibrator, 40mm, 2variables, remove one frame I obtain a mean
% error of 0.12 pixels (before it was 0.13, not very large I think).
load('cameraParams.mat') % Results from camera calibration
% K: calibration matrix
K = cameraParams.Intrinsics.IntrinsicMatrix'; % we need to transpose the
      IntrinsicMatrix
% Plot the distorsions
%ut_plot_lens_distortion(cameraParams, cameraParams.Intrinsics.ImageSize)

%% Stabilization and Tracking preparation

% 1. Constructing template for template matching
TemSize = 10; %defining size of both dimentions of the template
template = templatefunc(TemSize);
templateSingle = single(template); % needed for some functions
%imshow(template);

% 2. Looking for frame with clear bouy to set manually initial tracking point
stop = 0;
j = 0;
while stop == 0
    imgA = readFrame(video);
    imgAcheck = imgA(300:600, 300:1200, :);
    figure(1)
    imshow(imgAcheck, [])
    [xb, yb] = getpts; %coordinates of a buoy (double click), if there is no buoy,
          click enter
    if isempty(xb) == 0
        stop = 1; % when we obtain a visible buoy, we stop the loop
    end
    j = j+1;
end


% params for distance calculation
h = 2.5; % distance (in m) from camera to water
pp = round(cameraParams.Intrinsics.PrincipalPoint); % principal point [ppx, ppy]
F = cameraParams.FocalLength;
[~, horizon] = getpts; % y coordinate of the horizon at cropped image A
horizon = horizon + 300;
buoy = [xb + 300, yb + 300]; % coordinates of the buoy in image A

% Remove distortion in the image A
imgA = rgb2gray(im2single(imgA));
imgAun = undistortImage(imgA, cameraParams, "linear"); % imgA undistorted
movMean =  imgA;
correctedMean = 0;

%imshow(imgAun)


%% Loop

DrawingPerson = vision.VideoPlayer; % Create video viewer
%HelpMe = VideoWriter('p2MOB.mp4'); %create a video reader
%open(HelpMe);
```

```
% Initial params for stabilization.
imgAcr = imgAun(1:550,:); % Stable area of imgA for key points
imgBcr = imgAcr;
imgBp = imgAun;
tform = simtform2d;
tformcum = tform;

% Initial params for tracking
hTM = vision.TemplateMatcher('Metric','Sum of squared differences','ROIInputPort',
      true, 'BestMatchNeighborhoodOutputPort', true);
Idx0 = int32(buoy);
xb = Idx0(1);
yb = Idx0(2);
ROI = [xb-TemSize yb-TemSize 20 20]; %defining ROI position

flow = zeros(video.NumFrames, 2); % Values of movement of ROI/frame (in pixels)
count = 1;

%for evaluation
ceval = 0;
eval = zeros(21,2);
ieval = zeros(21,2);
distance = zeros(video.NumFrames);


for i = j:video.NumFrames-1 % from first frame to see buoy to last frame-1

    % Stabilization part, read new frame and remove camera distortion
    imgB = readFrame(video);
    imgBmod = rgb2gray(im2single(imgB));
    movMean = movMean + imgBmod; % for stabilization evaluation
    imgBun = undistortImage(imgBmod, cameraParams, "linear"); % imgB undistorted
    imgBcr = imgBun(1:550, :);

    % Obtain tform from Bcr to Acr
    ptThresh = 0.05;
    tform = obtaintform(imgAcr, imgBcr, ptThresh);
    tformcum.A = tformcum.A * tform.A; % tform acumulation
    sz = size(imgA);
    imgBp = imwarp(imgBun, tformcum, OutputView=imref2d([sz(1), sz(2)+300]), interp
          = 'linear');
    correctedMean = correctedMean + imgBp;

     %Find new position of the buoy
    Idx = hTM(imgAun,templateSingle,ROI);
    %Treshold to know if buoy is visible
    imgs = im2double(imgAun(ROI(2):(ROI(2)+20), ROI(1):(ROI(1)+20))); %getting ROI
          part of the image
    imenergy = imfilter(imgs.^2,ones(size(template)),'replicate'); %image energy
    imcorr = imfilter(imgs,template,'corr','replicate'); %correlation
    tplenergy = sum(template(:).^2); %template energy
    imssd = imenergy + tplenergy - 2*imcorr; %distance measure
    minim = min(min(imssd)); %minimum distance within the ROI

    % New ROI calculation

    if minim <= 3  % buoy is visible
            % Tracking buoy
            ROI = [Idx(1)-10 Idx(2)-10 20 20]; % New ROI
            buoy = cast(Idx,"double"); % New buoy position (center of ROI)
            flow(count,1) = Idx(1)-Idx0(1);
            flow(count,2) = Idx(2)-Idx0(2);
            count = count + 1;

    else % buoy not visible
    % Creating new ROI with the mean of the las 10 frames positions
         ROI(1) = ROI(1) + round(mean(flow(count-10:count, 1)));
         ROI(2) = ROI(2) + round(mean(flow(count-10:count, 2)));
    end

    % display distance and big ROI in 2 subwindows of the frame
    lilone = imgAun(ROI(2)-10:ROI(2)+30,ROI(1)-10:ROI(1)+30); %image around the ROI
    bigone = imresize(lilone,4); %scaled image around the ROI
    bigone = padarray(bigone,[8 8],0,'both'); %adding black borders
    imgAundo = imgAun;
    %keeping the same distance from ROI to the window
    imgAundo(Idx0(2)+371:Idx0(2)+550, Idx0(1)-600:Idx0(1)-421) = bigone;

    %calculation of the distance
    total_d = calculate_total_distance(pp, buoy, horizon, F, h);
    distance(i+1-j) = total_d;
    input = insertShape(imgAundo, 'rectangle', ROI,'Color', 'red','LineWidth',3);  %
          adding red ROI to the video
    txt = "Distance: " + sprintf('%3.1f', total_d) + " m";       %forming text
    input = insertText(input,[1150 1000],txt,'FontSize',28, 'BoxColor', 'black',...
        'TextColor', 'red', 'BoxOpacity', 0.8); %adding text on the image

    %releasing next function
    step(DrawingPerson, imshow(input))
%   writeVideo(HelpMe,input); %adding frame to the video;


    Idx0 = Idx; % New buoy position
    imgAun = imgBp; % Warped imgBp is the new imgA undistorted
    imgAcr = imgBcr; % Cropped imgBcr is the new imgAcr
    sz1 = size(imgA);
    %correctedMean = zeros([sz1(1), sz1(2)+300]);
    %correctedMean = zeros([1080, 1440]);


    %for evaluation
%     if mod(i+1-j,20)==0  %picking each 20th frame
%         ceval = ceval + 1;
%         figure(3)
%         evalut = imgAun(300:600, 300:1200, :);
%         imshow(evalut) %showing part of an image suitable for buoy observation
%         [eval(ceval),eval(21+ceval)] = getpts; %manually getting position of the
       buoy
%         ieval(ceval,:) = Idx0;  %defining automatically found coordinates
%     end

end

%close(HelpMe);

% Validation, manually pick the position of the buoy for each 20 frames,
% now we compare it with the results of the program.

% disp = 0;
% for s = 1:ceval
%     if s ~= 2 && s~= 7 %those are frames on which buoy is not seen
%         % calculating distance bitween automatically and manually found
       coordinates of the buoy
%         disp = disp + sqrt(cast(((eval(ceval,1)+300-ieval(ceval,1))^2 + (eval(
       ceval,2)+300-ieval(ceval,2))^2), 'double'));
%     else
%         disp = disp;
%     end
% end
% sse = disp/(ceval-2)  %average distance for frames on which buoy is visible


% Stabilization evaluation
correctedMean = correctedMean/(video.NumFrames-j-1);
movMean = movMean/(video.NumFrames-j);
imshowpair(movMean,correctedMean,'montage')


% Finding distance vs frame relation
k = linspace(1,video.NumFrames,video.NumFrames);
figure()
plot(k,distance,'black');
title('Distance to the buoy over time');
xlabel('Number of frame');
ylabel('Distance to the buoy');
xlim([0 427]);
ylim([0 250]);


%% obtaintform.m_____________________________________________________________

function tform = obtaintform(imgAcr, imgBcr, ptThresh)

% imgAcr = image1 to detect key points
% imgBcr = image2 to detect key points
% ptThresh = threshold for FAST features detection

% Detect key points
pointsA = detectFASTFeatures(imgAcr,MinContrast=ptThresh);
pointsB = detectFASTFeatures(imgBcr,MinContrast=ptThresh);

% Extract descriptors
[featuresA,pointsA] = extractFeatures(imgAcr,pointsA);
[featuresB,pointsB] = extractFeatures(imgBcr,pointsB);

% Match descriptors
indexPairs = matchFeatures(featuresA,featuresB, MatchThreshold=5);
pointsA = pointsA(indexPairs(:,1),:);
pointsB = pointsB(indexPairs(:,2),:);

% obtain tform
[tform, ~] = estgeotform2d(pointsB, pointsA,'similarity');


%% obtaintform2.m____________________________________________________________

function tform = obtaintform2(imgAcr, imgBcr)

% Detect key points
pointsA = detectBRISKFeatures(imgAcr);
pointsB = detectBRISKFeatures(imgBcr);

% Extract descriptors (no change with scale)
[featuresA,pointsA] = extractFeatures(imgAcr, pointsA,"Method","BRISK");
[featuresB,pointsB] = extractFeatures(imgBcr, pointsB, "Method","BRISK");

% Match descriptors
indexPairs = matchFeatures(featuresA, featuresB, Method = 'Exhaustive');
pointsA = pointsA(indexPairs(:, 1), :);
pointsB = pointsB(indexPairs(:, 2), :);

% Estimate geometric transform by matching inliers from B to A
[tform, ~] = estgeotform2d(pointsB, pointsA, 'similarity');

templatefunc.m____________________________________________________________

function t2 = templatefunc(size)

% Manually creating a template for the buoy of size = 10x10 pixels

t2 = zeros(size,size);
t2 = (1 + t2) * 0.36;

t2(4,4) = 0.50;
t2(4,7) = 0.50;
t2(4,5:6) = 0.95;
t2(5,4:7) = 0.95;
t2(6,4:7) = 0.95;
t2(7,5:6) = 0.95;
t2(7,4) = 0.50;
```

```
t2(7,7) = 0.50;


%% calculate_total_distance.m ____________________________________________________

function total_d = calculate_total_distance(pp, buoy, horizon, F, h)

% We are supposing horizon always above pp

% IMPUTS
% pp: coordinates of principal point (in pixels) [ppx, ppy]
% buoy: coordinates of buoy detection in the image (in pixels) [bx, by]
% horizon: y coordinate (supposing a straigh line)
% F: focal lengh (in pixels units of Ax and Ay) [fx, fy]
% h: distance from the camera to the sea (2.5 m)

% OUTPUT
% total_d = total distance in real world units (m) from the camera to the buouy.
%-------------------------------------------------------------------------------

% Calculate dz: distance in z axis from the camera to the buoy
alpha1 = atan(abs(horizon - pp(2)) / F(2));
alpha2 = atan(abs(pp(2) - buoy(2)) / F(2));
% R = 6371000; % Earth radius, in m
% H = sqrt((R+h)^2 - R^2);
% betta = atan(h/H); % In this case betta is nearly zero
betta = 0;

if pp(2) == buoy(2)
    dz = h / tan(alpha1 + betta);
elseif pp(2) < buoy(2)
    dz = h / tan(alpha1 + alpha2 + betta);
else
    dz = h / tan(alpha1 - alpha2 + betta);
end

% Calculate dx: distance in x axis from the camera to the buoy
dx = dz* abs(pp(1) - buoy(1))/ F(1); % distance in x axis from the camera to the
      buoy

total_d = sqrt(dz^2 + dx^2);
```

REFERENCES

[1] Katsamenis, I., Protopapadakis, E., Voulodimos, A., Dres, D., Drakoulis, D. (2020, June). Man overboard event detection from RGB and thermal imagery: Possibilities and limitations. In Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments (pp. 1-6).

[2] Örtlund, E., Larsson, M. (2018). Man Overboard detecting systems based on wireless technology.

[3] Bakalos, N., Katsamenis, I., Voulodimos, A. (2021, June). Man Overboard: Fall detection using spatiotemporal convolutional autoencoders in maritime environments. In The 14th PErvasive Technologies Related to Assistive Environments Conference (pp. 420-425).

[4] Selmy, A. S. (2016). The Need of Man Overboard (MOB) Detecting and Tracking System Descriptive Analyses. In Conference: The 9th China International Rescue Salvage Conference, China.