
DETECTION OF BALLAN WRASSE IN UNDERWATER IMAGES

Authors:
Mikkel Aas
Magnus Johan Gluppe
Jakob Frantzvåg Karlsmoen
Mikael Falkenberg Krog

Abstract

This project report contains the methodology, implementation, and results of several object detectors that attempt to find ballan wrasse in images. The goal was to detect a ballan wrasse then draw a box around it in underwater images.

Four different approaches were utilized: template matching, blob analysis, and the feature detection algorithms SURF and SIFT. For comparison, the YOLOv3 algorithm was used, which uses neural networks. The training was done on a data set of 152 images, and the finished implementations were tested on a test set of 50 images.

Different methods excelled at different types of images, but the overall best was blob analysis, achieving an average 56% match on the predefined ground truth of the images.

Contents

List of Figures	iii
List of Tables	iii
1 Introduction	1
2 Background	1
2.1 Data set	1
2.2 Detection evaluation metric	2
2.3 Methods	2
2.3.1 Template matching	2
2.3.2 SIFT	3
2.3.3 SURF	4
2.3.4 Blob analysis	4
2.3.5 YOLOv3	5
3 Implementation	5
3.1 Template matching	5
3.2 SIFT	5
3.3 SURF	6
3.4 Blob analysis	6
3.5 YOLOv3	8
4 Results and Discussion	9
4.1 Results	9
4.1.1 Template matching	9
4.1.2 SIFT	11
4.1.3 SURF	12
4.1.4 Blob analysis	14
4.1.5 YOLOv3	15
4.2 Discussion	16
4.2.1 Example images	17
5 Conclusion	18
Bibliography	19
Appendix	20
A GitLab repository	20

List of Figures

1	Example images from the data set	1
2	IoU equation visualization	2
3	The template used in all attempts	5
4	Image and it's binary image	7
5	Box plot of all the methods	9
6	Images showing examples where template matching does well and where it does not	10
7	The image with the best IoU score	11
8	The fish key points matches with the key points of the water pump	11
9	The four different pre-processing methods	12
10	Fish with both SURF implementations	13
11	Images that do not work well	13
12	An image where blob analysis works well	14
13	An image where blob analysis does not work well and its corresponding binary image	15
14	The best result	15
15	The outlier (no match found)	15
16	Image where blob analysis does better than the other methods	17
17	Image where template matching does better than the other methods	17
18	Image where SIFT and SURF does better than the other methods	18

List of Tables

1	Processing time and final IoU for the different methods	9
---	---	---

1 Introduction

This report documents and presents the process and results of a project in the course IDATG2206 Computer Vision. The project aimed to find a method for locating ballan wrasse in images. The purpose of this project is twofold, firstly for academic and learning purposes, and secondly to create a commercially viable object detector for locating ballan wrasses in images. Ballan wrasses are useful for cleaning sea lice from farmed salmon, which is why they are sought after in the fish farming industry. [1]

All necessary background information will be explained first, including explanations of the data set, detection evaluation metric used, and the theory behind the methods used for object detection. Then the individual methods and how they were implemented will be explained. Finally, the different results from each method will be discussed, and a conclusion will be reached.

2 Background

2.1 Data set

The data set used in this project was provided by NTNU. It consists of 202 images of ballan wrasse in different environments. These environments range from a bright tub with clean water where the fish is clearly visible to in-nature environments with turbid water where the fish can be challenging to locate even for humans. The picture quality and resolution vary greatly. Some pictures have a high resolution where many details are easily distinguishable, while others have a very low resolution.

Along with the images, a list of ground truths for all images was also included. The ground truth contains coordinates for the location of the fish in the images. It can be used to compare how well a method detected the fish in a particular image.



(a) Ballan wrasse in a tub



(b) Ballan wrasse in dirty water

Figure 1: Example images from the data set

The data set was split into two smaller sub-sets. A set of 50 hand picked images were taken out as a testing data set. This is the set that was used to measure the effectiveness of the different methods. The remaining 152 images were used as a training set. This is the data used to implement or train the different methods.

2.2 Detection evaluation metric

An evaluation metric is needed to compare the different image detection methods. The evaluation metric used in this project is Intersection over Union (IoU). IoU calculates a number between 0 and 1, specifying how much found bounding box overlaps with the ground truth bounding box. An IoU score of 0 means there is no overlap between the two, and an IoU score of 1 means that the two bounding boxes are completely overlapping. What is considered a good IoU score varies from project to project, but generally, an IoU score greater than 0.5 is considered good.[2]

The IoU score is calculated by dividing the overlap area of both boxes by the total area of both boxes.

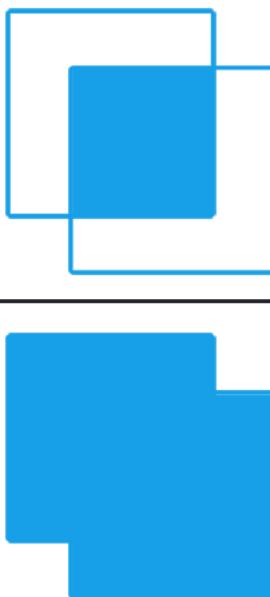
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 2: IoU equation visualization

Source: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>

Other detection evaluation metrics that could have been used are Average Precision (AP) or Mean Average Precision (mAP). Using more evaluation metrics might result in more accurate statistics regarding the detections. Due to time restrictions, only the IoU scores were computed.

2.3 Methods

2.3.1 Template matching

Template matching is a technique in computer vision used to locate parts of an image that match a predefined template. The two most common approaches are template- and feature-based template matching. This technique is widely used in object detection fields, such as surveillance, vehicle tracking, robotics, medical imaging, and manufacturing. [3]

Template-based template matching, also known as the level histogram method, is generally seen as the most computationally expensive. The process involves moving the template image over every part of the larger image and computing a “score” for each position. This numeric score indicates how well the template matches the larger image at the given position. Implementations based on this approach are very robust and have been widely studied. [3, 4]

Feature-based template matching, on the other hand, is a somewhat new approach. One of the drawbacks of the other approach is that it is hard to set the templates and define the rules for matching with this template. With a deep learning-based approach, it is possible for the model to learn the rules and details of a template. It can then use what it learned to locate the template in other images, and in many cases, with higher speed and accuracy than most template-based template matching approaches. [5]

2.3.2 SIFT

SIFT or scale-invariant feature transform is a feature detection algorithm. The algorithm describes and detects local features in images. The algorithm was first published and patented by David Lowe in 1999. It has several use-cases, including robotic mapping, image stitching, 3D modeling, gesture recognition, and video tracking.

The SIFT algorithm extracts key points from a reference image (or set of images) and saves them in a database. Objects are recognized in a new image by individually comparing each feature to the database of key points. It finds the candidate matching features by calculating the euclidean distance of their feature vectors. In short, the euclidean distance is calculated by taking the cartesian coordinates of the points and applying the Pythagorean theorem. The result of this is a large set of matches. These matches are subsequently filtered down using key points correlated to the object's location, scale, and orientation. Hough Transform is used to determine consistent clusters. Each cluster containing three or more matching features is subject to further verification where outliers are discarded. Finally, the probability of a feature match is computed, given that an object is found.[6]

There are four main steps to this algorithm. Firstly, a Gaussian pyramid is constructed from the input image by repeated smoothening and subsampling. Secondly, a difference-of-gaussian pyramid is computed by taking the differences of the adjacent levels in the gaussian pyramid. Interest points, or key points, are obtained by looking at the extrema points in the difference-of-gaussian pyramid concerning the spatial coordinates in the image domain.

When the potential key points are detected, they have to be refined to generate more accurate results. The first step is to use the Taylor series expansion of scale-space to get a more accurate location of the extrema. If the intensity at this extrema is less than a set threshold value, it is rejected. This threshold is called the contrast threshold in OpenCV. Further, the difference of gaussian has a high response for edges; this is filtered out by using an edge threshold. The filter applies a 2x2 Hessian matrix that calculates the principal curvature. Principal curvatures are known as the eigenvalues of the shape of a point on a surface. They measure how the surface bends in different directions. The result of thresholding is strong interest points where low-contrast and edge key points are discarded.

Orientation is assigned to each key point to accomplish invariance to image rotation. The gradient magnitude and direction are calculated in a region around the key point, resulting in an orientation histogram with 36 bins covering a 360-degree angle. From the histogram, any peak over 80% is considered when calculating the orientation. New key points with the same location and scale but with different directions are formed.

Keypoints between two images are matched by identifying their nearest neighbor. Due to, for example, image noise, the second closest match may be very near to the first. Therefore a ratio between the closest distance and the second closest distance is made. If this ratio is greater than 0.8%, the second closest is rejected. It eliminates roughly around 90% of false matches and discards only 5% of correct matches.[7]

2.3.3 SURF

SURF or speeded up robust features is feature detection algorithm. It was first published in 2006 to the European Conference of Computer Vision by Herbert Bay, Tinne Tuytelaars and Luc Van Gool. It is loosely based on SIFT, but is faster and more robust.

The SURF algorithm has three main steps: interest point detection, local neighborhood description, and matching. For interest point detection, SURF uses integral images, which is a way to calculate the sum of pixel values in a given image. The interest points themselves are based on Hessian-matrix approximations where blob-like structures are detected. For robustness, the key points must be evaluated in different scales because the object that is being detected might be of a different scale than the object in the sample image. This is achieved by using a scale-space. The calculation using integral images is independent of size, which lends itself well to the big filter sizes SURF utilizes.

The scale space is built up using Gaussian smoothing, where each level is represented in an image pyramid. Other object detection algorithms like SIFT use the difference of Gaussian, where the image is scaled down, smoothed, and pyramid levels are subtracted to find edges and blobs. However, SURF can use the integral images to upscale the filter rather than iteratively reduce image size. The scale space is then divided into octaves, where each octave is the image with different sizes of filters. For each new octave, the filter size is doubled. Finally, SURF uses scale-space interpolation to localize interest points. [8]

The next step is to find the orientation of the interest points SURF detected. SURF is robust when it comes to image rotation, which is achieved by finding the local orientation of the interest points. This is done by calculating the Haar wavelets in a circular neighborhood. This means that a circle is drawn around the key point and creates a vector based on the distribution of Haar wavelets. The sampling of the wavelets is based on the scale in the scale space. This way, each interest point has a unique orientation. Since the interest points have a unique orientation, it does not matter what rotation the image is if there are enough corresponding interest points. For instances where orientation is not relevant, it is possible to use U-SURF or upright SURF which skips this step entirely for computational efficiency.

The final step is matching corresponding interest points. These points are typically found in matching blob-like structures. SURF is fast at matching interest points because it uses the sign of Laplacian, which distinguishes bright blobs from dark backgrounds. By utilizing this, it only compares features with the same type of contrast. This has no extra computational cost, as this is already calculated in the detection phase. This creates faster matching without reducing the performance of the descriptor.[8]

2.3.4 Blob analysis

Blob analysis is a method of analyzing an image that has undergone binarization. Binarization means that an image is turned into a matrix consisting of only ones and zeroes. To binarize an image, a threshold needs to be set. For instance, one could convert an image into grayscale, and to find the dark parts of the image, a low threshold would be chosen. Every pixel with a value under that threshold gets a value of 1, and everything else a value of 0.[9]

Once a binary image has been found, a blob detector is used to find blobs in the binary image. The blobs are connected areas of white in the binary image. One of the most commonly used blob detectors is based on the Laplacian of the Gaussian, where a Gaussian kernel convolves the image at a specific scale t to find a scale-space representation. Then the result of applying the Laplacian operator is computed; this usually results in strong positive responses for dark blobs of radius \sqrt{dt} , where d is the number of dimensions in the image and strong negative responses for bright blobs of a similar size. Using the operator at a single scale like this may be a problem. Therefore a *multi-scale blob detector with automatic scale selection* should be found instead. To find this a *scale-normalized Laplacian operator* is used instead, and the scale-space maxima/minima are taken into account. [10]

2.3.5 YOLOv3

YOLOv3 (You Only Look Once) is a real-time object detection algorithm. It is one of the most effective object detection algorithms that encompasses most of the innovative ideas produced by the computer vision research community. The algorithm requires only one forward propagation pass through the neural network to make predictions. In short, it learns generalizable representations of objects by being fed an extensive training set.

The algorithm is based on regression. YOLOv3 works by applying a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. There are a few different implementations of the YOLO algorithm. For this project, the darknet neural network version was implemented.[11]

3 Implementation

3.1 Template matching

All template matching implementations were done in MATLAB. MATLAB is a proprietary programming language and numeric computing environment developed by MathWorks. [12]

The first attempt at performing template matching used an approach called *the sum of absolute differences*, or *SAD*. The sum of absolute differences is calculated by taking the absolute difference of each pixel in the template and the corresponding pixel in the bigger image and then adding them together. This would be done for every possible position of the template in the bigger image. This attempt was discarded, though, as it was apparent that it would be too inaccurate and slow to be useful.

Further attempts use an approach based on *normalized cross-correlation*. Cross-correlation, also known as *sliding dot product* or *sliding inner-product*, is a measure of similarity between two time series. Normalized cross-correlation allows for comparisons between time series of different value ranges. The scoring value is also easier to understand than non-normalized cross-correlation, as it is simply a value from 0 to 1. [13]

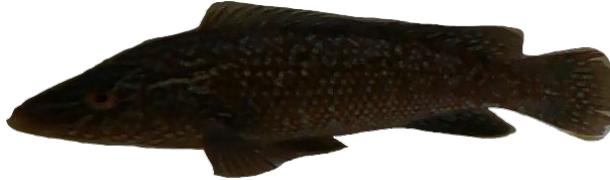


Figure 3: The template used in all attempts

3.2 SIFT

The SIFT algorithm was implemented using the OpenCV library in python. OpenCV (Open Source Computer Vision Library) is an open-source machine learning and computer vision library. The library was used to ensure the best result and to save time from implementing the algorithm from scratch.

Locating the bounding box coordinates was done by using the coordinates of the matched key points. These key point coordinates were found by iterating over every match that the BFMatcher produced. The list of produced key point coordinates was then iterated to find the most extreme values on each axis. These extreme values were then used to find the bounding box corners. Finding the coordinates this way is an inadequate implementation. It demands that a matching key point is found on the top, left, bottom, and right of the fish, which is rarely the case. If less than four key points are found, a box can not be drawn, and if the matching key points are matched in a small area, a tiny bounding box is drawn.

The SIFT algorithm needs an object to find. Therefore a template had to be chosen. The template was chosen based on three factors, the lack of noise, the best quality, and the perspective. Since most fishes were in a sideways position, a template where the fish is seen from the side was chosen. The chosen template can be seen in Figure 4..

3.3 SURF

The SURF functionality in this project is implemented using MATLAB. MATLAB has built-in SURF features, making it possible to extract key points without writing the algorithm from scratch.

The first step was to detect SURF features in two images, find corresponding interest points, and use the 300 strongest. The earliest comparison includes outliers. The outliers are interest points that are not on the fish but other objects in the picture. Then the search is narrowed only to include inliers with an estimated geometric transformation. When the inliers have been discovered, a box is drawn around the corresponding interest points.

An early obstacle was that the *detectSURFFeatures* function failed to find enough interest points to compare the images. This was solved by being more specific in the parameters of the *detectSURFFeatures* in MATLAB. By setting the *MetricThreshold* to 100 and *NumScaleLevels* to 5 enough interest points were found. The next challenge was a lack of corresponding inliers to make an accurate bounding box. Frequently only 2-3 matching inliers were found, so the box ended up as a thin line if a box was drawn at all. To increase the amount of corresponding key points, the *MetricThreshold* was lowered to 50, and the *NumOctaves* was set to 4. With 4 octaves it is easier to detect features of larger sizes. This improved the results some but was still inadequate for a good image detector.

To work around this, it can use the *matchedpoints* which includes outliers instead of using the inliers. The problem is that it cannot use the estimated geometric transformation to draw the bounding box the same way as the previous implementation. Instead, it iterates through all the key points and finds the top left and bottom right corners. Then a box is drawn using these coordinates. This increased the success rate of detecting fish but reduced the accuracy of the box. Different templates were used to improve the method further, and a sideways fish gave the best results.

3.4 Blob analysis

The Blob Detection is implemented in MATLAB, using the built-in blob analysis function from the computer vision toolbox.

First, the image has to undergo *binarization* and to binarize an image; a threshold is needed. The MATLAB app called *color thresholder* is used on images in the training set to find different thresholds. The thresholds can be in different color spaces, and the color spaces used in this implementation are HSV and RGB. Why these two were chosen is discussed in the "Results and Discussion" section. An example of what a binary image looks like is included in Figure 4.



(a) Image



(b) Binary image

Figure 4: Image and it's binary image

After finding a binary image, it is opened with a rectangular *morphological structuring element*. The structuring element used in this implementation is a rectangle. Why this was chosen is discussed in the "Results and Discussion" section.

After the binary image is opened, blob analysis is performed on the binary image found from HSV color thresholding; if no blobs are found, a new binary image is found using a different threshold. This is still in the HSV color space. If no blobs are found here, a third binary image is found by thresholding in the RGB color space, and blob analysis is performed on this image. If there still are no blobs found, it is assumed that there are no fish in the image.

If more than one bounding box is found in an image, the IoU score is calculated for each box, and then the average IoU score of all the bounding boxes is calculated.

The main problem with this implementation is that it will not try again with a different binary image when it finds a blob that is not a fish. Another problem is that it sometimes finds bounding boxes not containing the fish; this may be because the thresholds are too sensitive. Removing the RGB thresholding and only using HSV might fix this problem. The thresholds could also be fine-tuned a lot more.

3.5 YOLOv3

YOLOv3 was implemented using pythons OpenCV computer vision and machine learning library, combined with the darknet neural network.[14]

Firstly an image data set had to be prepared. It was suggested to use at least 100 images, so the data set was divided into two groups. The first group consisted of 150 training images and the second of 50 testing images. Having the images was, of course, not enough. The custom object, in this case, the ballan wrasses, had to be located in each image. For this operation, a software called LabelImg was used. LabelImg is a software that generates a ground truth text file. The ground truths had to be in a YOLO format. Therefore, the ground truths given to us by NTNU could not be used.

Secondly, the data set was trained online by an accessible server offered by Google Colab. On Google Colab, one can run python scripts and use machine learning algorithms that take advantage of Google's powerful hardware. The only disadvantage with the service is that one can only use it for twelve hours. However, there are no limits to restarting the service.

After setting up a connection between google drive and Google Colab, the images were trained using the darknet neural network. In short, it takes the ground truths and their corresponding images to create kernels that are specifically trained to detect what is within the ground truth.

The result of the neural network was tested by using the OpenCV library. First and foremost, the neural network was loaded using the produced weights from the darknet. Then the output layers were identified by iterating over them in the net. Lastly, a loop iterating over all the test images was constructed. Within this loop, the objects were detected by using the OpenCV blobFromImage algorithm.

4 Results and Discussion

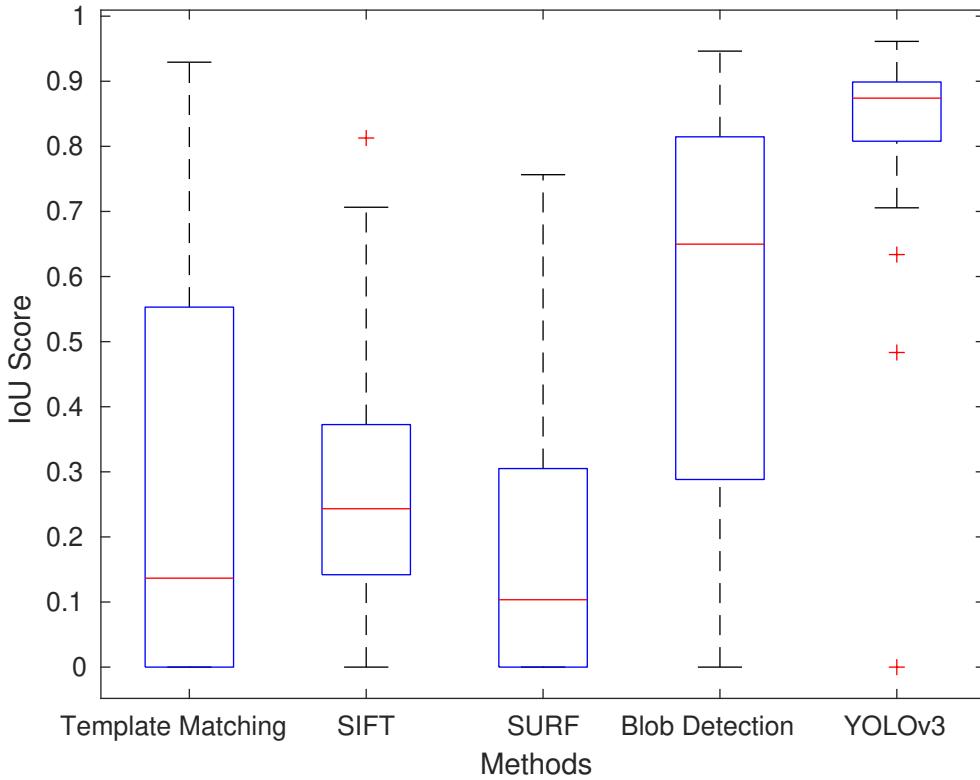


Figure 5: Box plot of all the methods

	Template Matching	Sift	Surf	Blob Analysis	YOLOv3
Processing time	9.9s	15s	36.6s	1.8s	27s
IoU score	0.28	0.29	0.25	0.56	0.83

Table 1: Processing time and final IoU for the different methods

4.1 Results

In all images containing bounding boxes, the green boxes will show the ground truth, and the red boxes will show the detection.

4.1.1 Template matching

Without any alterations to the template or the source image, the first implementation got an IoU score of 0.20 and used 11.4 seconds. This is not a great score, so it has room for improvement. Looking at the template, it is apparent that it is very dark compared to many of the test images. Performing histogram equalization on the template and image could equate to better performance. The template is also bigger than some of the images in the test data set, which means it is not possible to perform template matching on them. Resizing the template or image could fix this issue.

Performing histogram equalization on the template increased the IoU score to 0.23 and decreased the processing time to 8.4 seconds. While it is not a considerable increase, it is somewhat better. Performing histogram equalization on the template and image simultaneously or only the image both yielded worse results.

Resizing the template is a little tricky, as the shape of the images in the test data set varies a lot. This makes it hard to ensure that both the width and height of the template are resized correctly to fit inside the image. A dirty solution was then to have a loop that continuously shrinks the template by some factor until it fits in the test image. After adding a loop that shrinks the template by 25% for each iteration, the IoU score rose to 0.28. While this solution did add 1.5 seconds to the processing time, an increase of 0.05 on the IoU score makes this an acceptable trade-off.

Rotation is another aspect of the template that can be important for getting a good match. Therefore, an attempt was made to implement a loop that would search every image with four rotations of the template and choose the best match. This increased the processing time by almost 4x (to 35.9 seconds) and somehow decreased the IoU score to 0.25. It is unclear why this was the case, as it, in theory, should only have been able to produce an equal or better IoU score than if it did not rotate the template. Since there were no bugs found within the implementation and no reasonable explanation for it, somehow seemingly finding different matches that were, in reality, worse, it ended up being scrapped because of the regression in both IoU score and processing time.



(a) Good match



(b) Bad match

Figure 6: Images showing examples where template matching does well and where it does not

4.1.2 SIFT

On the first implementation of the SIFT algorithm, the IoU average score was 0.29, and the time was approximately 14.9 seconds. The low score was presumably due to the diversity of perspectives, distortions, and fish poses in the test set. In images where the perspective is equivalent to the one in the template and very little to no distortion is present, the IoU score can get as high as 0.81. Another negative factor could be inaccurate matches. There are many instances where key points on the fish get matched with other materials in the tank.

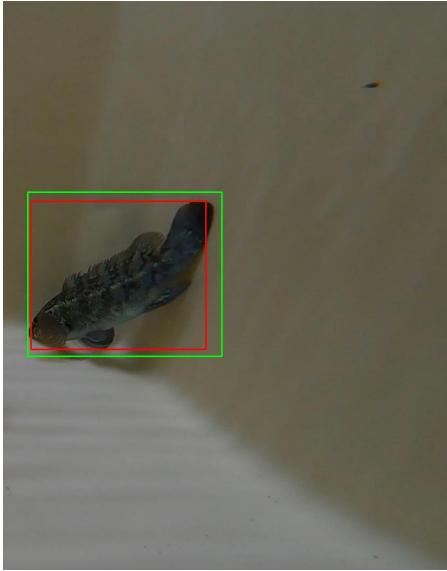


Figure 7: The image with the best IoU score

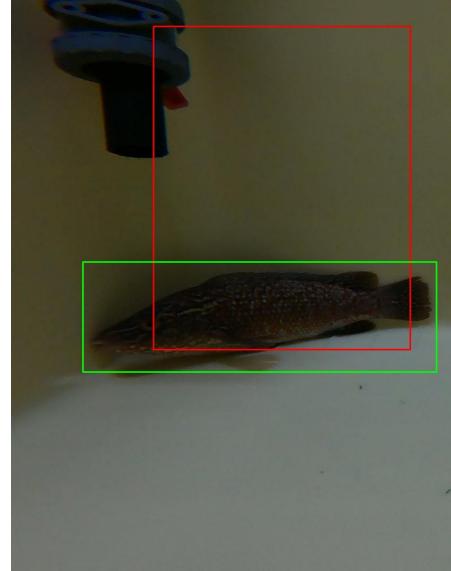


Figure 8: The fish key points matches with the key points of the water pump

There was an attempt to fix the distortion problem by sharpening the image. A kernel filter was applied on top of each image in the test filter to sharpen the image. The result was an average IoU score of 0.30, but the time it took was 17.5 seconds. Sharpening the images seems to have highlighted certain features that were otherwise too smooth. Contrarily, denoising the images did not alter the result, although it took 89.6 seconds. This result is not unexpected, as most of the images in the test set are not noisy. When looking at a noisy picture, the IoU score remained unchanged. Upon combining both the sharpening and the denoising, the average IoU score lowered by 0.02. The sharpening and denoising could have caused a counteractive effect and made the images too distorted. However, the time increased to about 52.6 seconds.

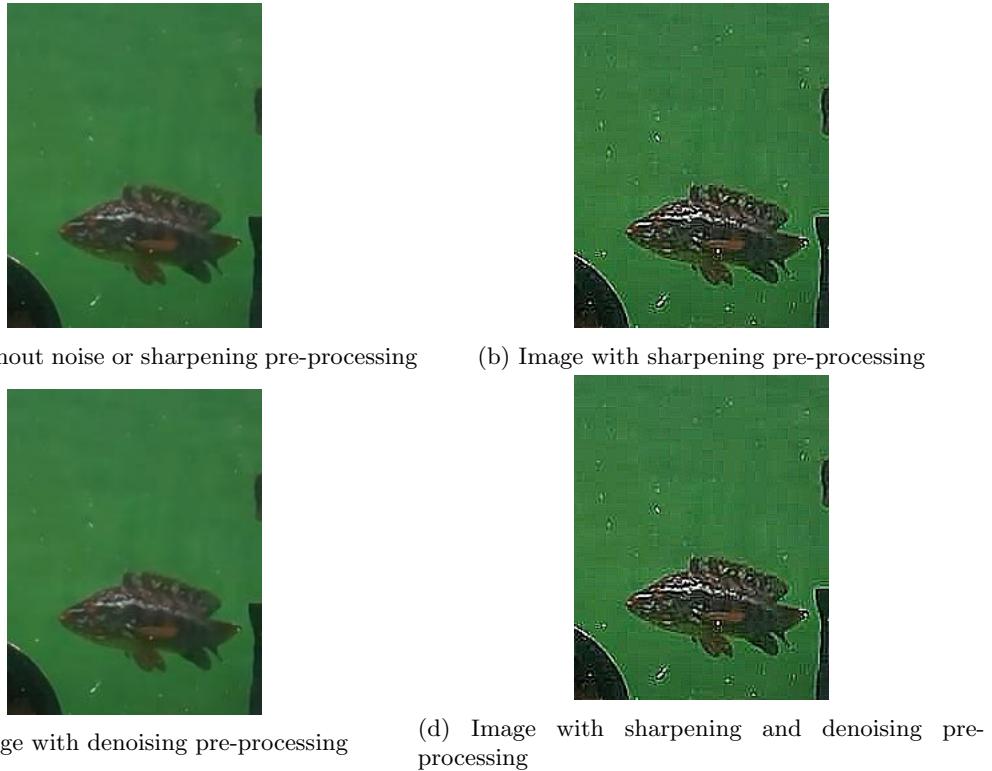


Figure 9: The four different pre-processing methods

Perspective is an essential factor in object recognition using SIFT. If two images of a fish are taken from different angles, the algorithm has a hard time matching the fishes because the angles of the fishes are very different. On the other hand, if the fishes are seen from the same angle, the key points are very similar. In other words, the more similar the perspective is, the more key point matches are found. Using multiple templates where the fishes are seen from different perspectives should therefore improve the results. However, the side perspective template proved to provide a better IoU score in all cases, even when matching against a fish seen from above.

4.1.3 SURF

Using the second implementation, SURF analyzed the data-set in 36.6 seconds and managed an IoU score of 0.25. It discovered an object in 45 out of 50 images, though several being false positives. The SURF algorithm is very fast and can be used in real-time computation, but this implementation has to loop through every key point, which takes some time. Therefore the SURF method in this project is not capable of real-time object detection. The cause of this could be the current implementation is not how SURF was intended to be used. By drawing a box around the top left and bottom right key points, one gets an incomplete picture of the detected object. Instead of estimating the size and position of the fish, it simply draws a box around the cluster of key points. The first iteration gives a more accurate bounding box by analyzing the corresponding key points and estimating the fish's size and orientation. As seen in Figure 10, the blue box is the first implementation, and the red box is the second implementation. The first iteration gave very accurate results; however, due to the lack of matching key points this implementation found even fewer matches than the second implementation, where only 8 out of 50 were usable.

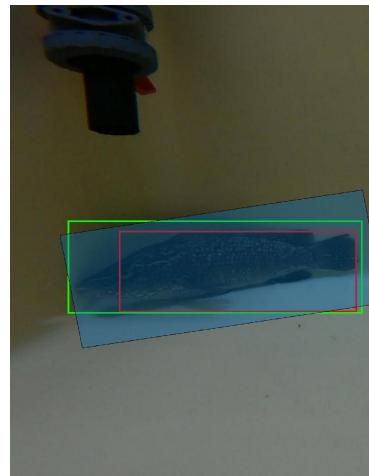
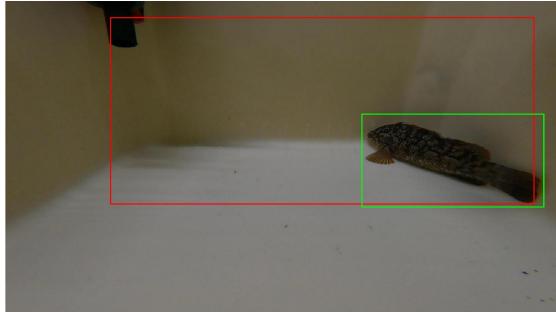


Figure 10: Fish with both SURF implementations

SURF excels at matching images similar to each other, being robust at different scales and rotations. It performs well at sideways fish if the sample image also sideways but fails at different viewpoints and poses for the fish. The best result was achieved using a sideways fish template. Drawing a box around the cluster of key points fails to detect the entire fish but often finds key points on the fish. These boxes are small and decrease the average IoU score but can be used to detect where the fish is in the image; see Figure 11. In some cases, it might be sufficient to detect where the fish is in the image; then, this SURF implementation might be more useful. It would be possible to give images where the bounding box is on top of the fish more weight than simply going by IoU score. However, in several instances, SURF detected other objects in the image and created a large box that overlapped the fish but was not very accurate, demonstrated in Figure 11.



(a) A large box overlapping the fish



(b) A small box on the fish

Figure 11: Images that do not work well

4.1.4 Blob analysis

In the first version of this implementation, only the HSV color model was used when finding the binary image. With this color model, the program achieved an IoU score of about 0.20. Later, when some tweaks were made to the threshold, an IoU score of 0.29 was achieved. HSV is used because, unlike RGB, it separates the image intensity and the color information. Other color spaces that also do this, like LAB, could also be used.

In later implementations, thresholding in LAB and RGB were also added. This implementation achieved an IoU score of 0.42. If no bounding box was found in the binary image from the HSV thresholding, LAB and RGB thresholding were used instead. It took 2.6 seconds to go through 50 images, meaning it could almost be used in real-time.

In a later version, some more modifications to what color models used were made. Since HSV gave the best results, it will still be the first one attempted, but instead of using LAB as the second model, HSV is used again. This time with a different threshold. The final threshold used was still in the RGB color model. This resulted in an IoU score of 0.43, which is a slight improvement from the previous version.

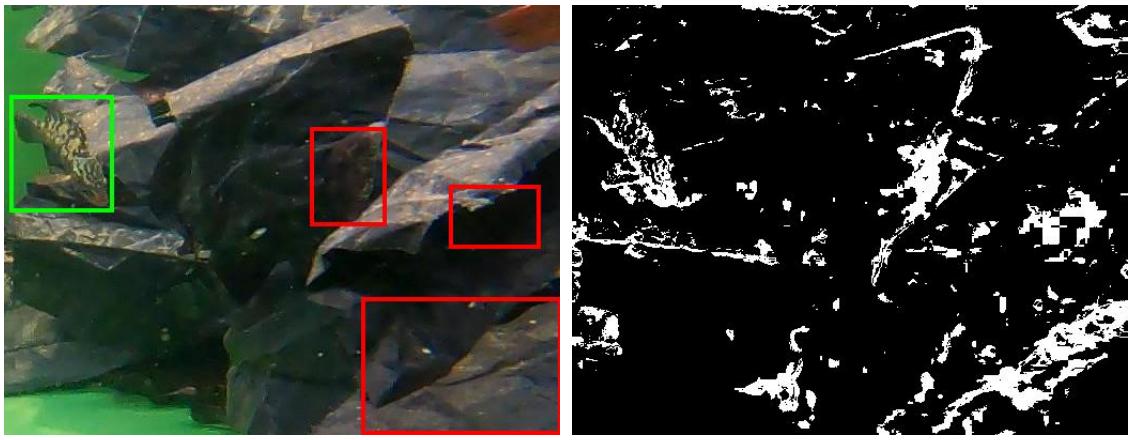
More improvements were made by changing the *morphological structuring element* used when opening the image to a rectangle instead of a disk. This is because one wants a structuring element that resembles the object to detect in the image. Most fish will be seen from the side, which will look more like a rectangle than a disk. Another change was that the minimum blob size was increased when using the first binary image but decreased when using the second and third binary image, which improved the IoU score. This improves the IoU scores because instead of finding small bounding boxes that are not the fish when doing blob analysis on the first binary image, it will not find any bounding boxes. It will then use the second binary image found with a different threshold. After these changes, the IoU score was now at 0.56; the processing time for 50 images was 1.8 seconds. The median score ended up being 0.65

Figure 12 shows an image where blob analysis works well. It most likely works well in this instance because the fish clearly stands out from the background in the image.

Figure 13 shows an image where blob analysis does not work well. This is because the fish is not separated from the background in the image, and the illumination and colors of the background are too similar to the fish.



Figure 12: An image where blob analysis works well



(a) Image

(b) Binary image

Figure 13: An image where blob analysis does not work well and its corresponding binary image

4.1.5 YOLOv3

Overall, YOLOv3 produced a median IoU score of 0.88, with a high of 0.96 and a low of 0.00. The time it took for the algorithm to run was approximately 27.0 seconds. This deep learning method was only meant for comparison with the standard computer vision methods. Therefore, no extensive pre-processing was done to improve the result.

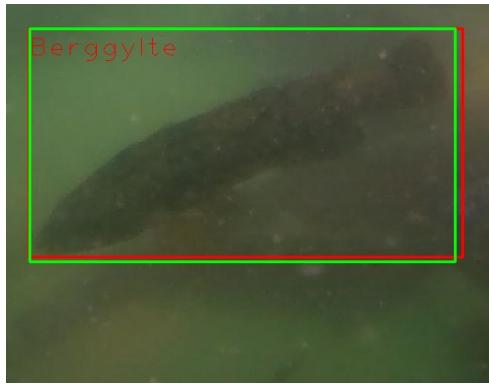


Figure 14: The best result



Figure 15: The outlier (no match found)

4.2 Discussion

After quite a few methods have been tried out, it is possible to analyze the results of these and see if any of them are viable. The project's goal was to "find a way to locate ballan wrasses in images for use in fish farming.". The results from some of the methods look promising for fulfilling this goal.

From looking at the box plot in Figure 5, it is possible to get some idea of how the methods compared to each other. At first glance, it is evident that YOLOv3 is performing significantly better than the other methods. As YOLOv3 is much more advanced than the implementations of the other methods in this project, this was expected. Blob analysis is somewhere in the middle, while template matching, SIFT, and SURF are towards the lower end of the plot.

Template matching is very spread out between 0 and 0.55. It also does get a few results close to 1.00. This was expected, as in several of the images, the fish is very similar to the template used. These results indicate that template matching can be quite accurate if one looks for something that does not vary much from image to image. Adding more templates to match for will also improve the result but will lead to higher processing times.

Blob Analysis finds a good match in many images, but almost half of the images achieve an IoU score below 0.5. It does, however, have a processing time of 1.8 seconds on 50 images, meaning it could be used in real-time detection with 25 frames per second. A problem that will most likely occur is that the blob analysis will detect other objects in the image that is not a ballan wrasse; for instance, other fish that look similar or other objects with similar color, saturation, and lightness. This is because the main thresholding method using HSV is not very refined in this implementation. The IoU result for blob analysis could be further improved in several different ways. For instance, one could only include blobs that have a shape resembling fish. Blob analysis could also be enhanced by better pre-processing. Methods that may improve the IoU score include histogram equalization and noise removal.

The IoU scores calculated when using SIFT are generally low. Half of the scores are between 0.14 and 0.37. The median is 0.24, the lower adjacent is 0, and the upper adjacent is 0.81. Extremely noisy and unsharp images presumably give the lower quartile between 0 and 0.14. On the other hand, very sharp and similar images to the template give the upper quartile between 0.37 to 0.81. Because it takes the algorithm about 15 seconds to run, it is not suitable for live tracking. Neither is it ideal for tracking ballan wrasse in still images as the IoU score is 0.26 below what is regarded as a good prediction. SURF and SIFT share many similarities, and SURF has an equally low IoU score of 0.25. Both algorithms excel when the objects are very similar to the sample image.

SIFT and SURF are somewhat in the same boat as template matching. They are prone to variations in the fish from image to image. If the fish is viewed from a different perspective than the template used for these methods, they will have difficulty locating it. However, they have less spread in their results based on the box plot in figure 5. This could make them more reliable than template matching, as it can be easier to predict if what it locates is useful.

YOLOv3 achieves a better result than any of the non-neural object detection methods used in this project. There are multiple reasons machine learning performs better than standard computer vision techniques. For instance, ballan wrasse will have varying colors depending on their habitat. The varying colors make it harder to detect using non-neural object detection methods. The perspective of the image will also impact how SIFT and SURF will perform negatively. With enough training images and GPU power, machine learning will quickly detect these changes in color and perspective. Overall, YOLOv3 produced a median IoU score of 0.88, with a high of 0.96 and a low of 0.00. The lowest score is due to a very blurry image where the fish blends with the background and is seen from the front. In contrast, the highest score is produced by an image where the fish is seen from the side in an almost equally blurry image. The time it took for the algorithm to run was approximately 27.0 seconds, making it unsuitable for live tracking. Unless a framerate of 1.7 would suffice. However, it is suitable for tracking ballan wrasses in still images as it has an average IoU that is 0.30 above what is considered a good detection prediction.

4.2.1 Example images

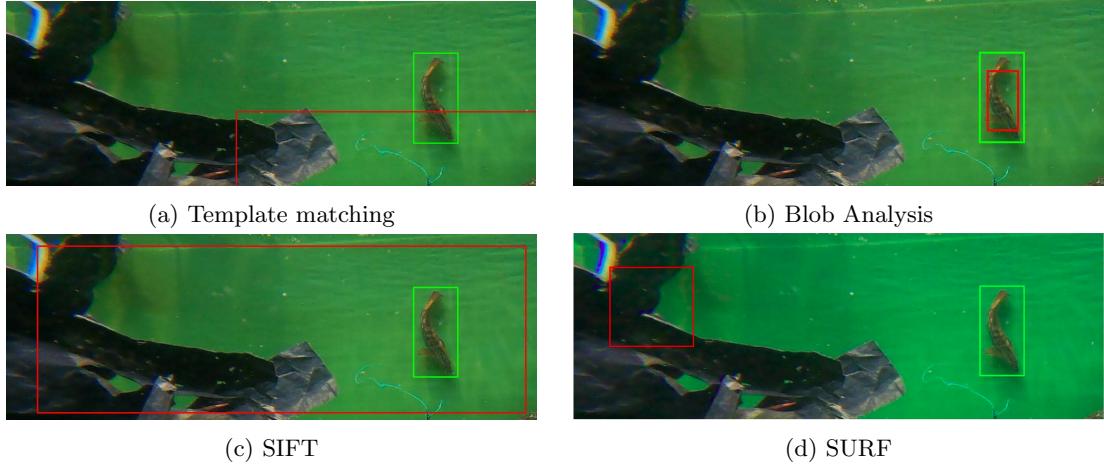


Figure 16: Image where blob analysis does better than the other methods

Blob analysis does the best in Figure 16 because the fish stands out from the background color-wise and light intensity-wise. However, SIFT and SURF have trouble finding matching key points because the fish is in a different perspective and pose than in the template images. Likewise, the template matching algorithm also uses a template where the fish is in a different view and pose, which results in a bad match.

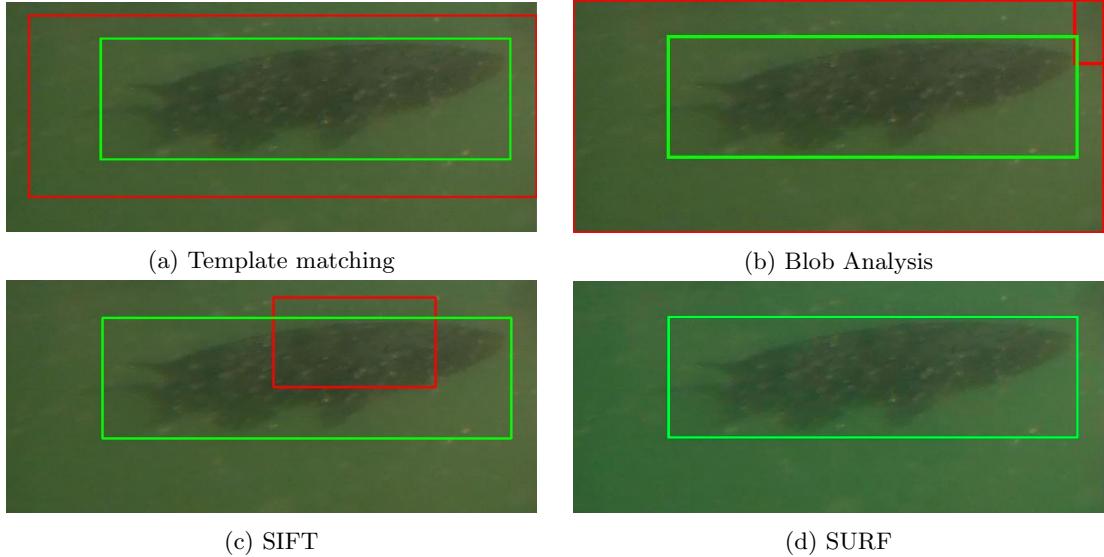


Figure 17: Image where template matching does better than the other methods

Template matching does well in Figure 17 because the fish in the image is very similar to the template—both in shape and color. The surroundings are a different color, so it gives the fish a clear border like the template. The other methods are having a hard time because the image is very blurry, and it is, therefore, hard to find edges to use for good key points.

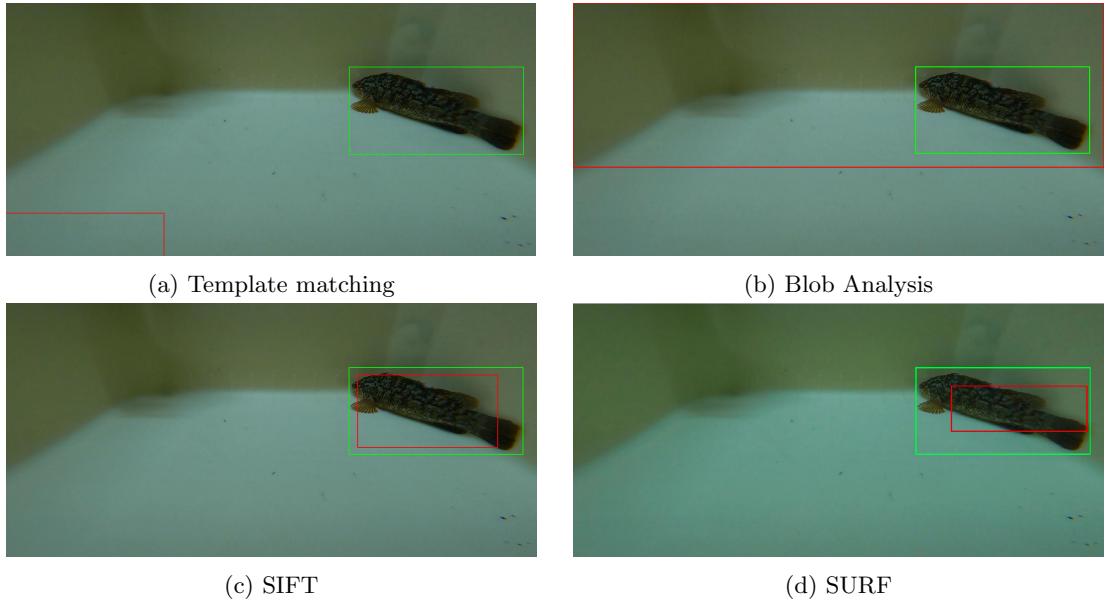


Figure 18: Image where SIFT and SURF does better than the other methods

SIFT and SURF perform better than the other methods in Figure 18 because the fish blends in with the background color and illumination. This affects SURF and SIFT less than blob analysis. Also, there are no other objects in the image, so there are no other potential key points than fish. The fish is flipped, which confuses the template matching, but it is still similar enough to detect enough corresponding key points.

5 Conclusion

The conclusion reached is that the most successful detection method used in this project was blob analysis. While blob analysis gave the best result, it is not accurate enough to be certain the detected object is a ballan wrasse. In contrast, when template matching finds a match, it will most likely be a ballan wrasse. SIFT and SURF are rotation invariant and scale, thus outperforming template matching in a few cases. However, both SIFT and SURF struggle with different viewpoints and multiple objects in the image. Blob analysis struggles when the fish blends in with the background. SURF and SIFT perform better in this environment, but if they cannot detect key points because of a murky image, they also fail. The best method used in this project is blob analysis, excluding YOLOv3, which utilizes neural networks. For some images, different techniques will be better on a case-by-case basis. Machine learning techniques will consistently outperform the non-neural object detection methods when detecting fish.

Bibliography

- [1] The Research Council of Norway. *New approach to combating sea lice: Wrasse to the rescue*. 2010. URL: <https://www.sciencedaily.com/releases/2010/04/100423215021.htm> (visited on 15th Apr. 2021).
- [2] Adrian Rosebrock. *Intersection over Union (IoU) for object detection*. 2016. URL: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> (visited on 8th May 2021).
- [3] Anan Banharnsakun and Supannee Tanathong. ‘Object Detection Based on Template Matching through Use of Best-So-Far ABC’. In: *Computational Intelligence and Neuroscience* (2014). DOI: <https://doi.org/10.1155/2014/919406>.
- [4] Frédéric Jurie and Michel Dhome. ‘Real Time Robust Template Matching’. In: *BMVC 2002* (2002), pp. 123–132.
- [5] Biao Hou et al. ‘Object Detection in High-Resolution Panchromatic Images Using Deep Models and Spatial Template Matching’. In: *IEEE Transactions on Geoscience and Remote Sensing a Publication of the IEEE Geoscience and Remote Sensing Society* 58.2 (2020).
- [6] David G. Lowe. *Object Recognition from Local Scale-Invariant Features*. 1999. URL: <https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf> (visited on 18th Apr. 2021).
- [7] opencv. *Introduction to SIFT (Scale-Invariant Feature Transform)*. 2021. URL: <https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf> (visited on 20th Apr. 2021).
- [8] Tinne Tuytelaars Herbert Bay Andreas Ess and Luc Van Gool. ‘Speeded-Up Robust Features (SURF)’. In: *Computer vision and image understanding* (2007). DOI: <https://doi.org/10.1016/j.cviu.2007.09.014>.
- [9] George C Shapiro Linda G. Stockman. *Computer Vision*. Prentice Hall, 2002. ISBN: 0-13-030796-3.
- [10] Tony Lindeberg. ‘Scale Selection Properties of Generalized Scale-Space Interest Point Detectors’. In: *Journal of Mathematical Imaging and Vision* 46.10 (2013), pp. 177–210. URL: <https://link.springer.com/article/10.1007%2Fs10851-012-0378-3> (visited on 28th Apr. 2021).
- [11] Joseph Redmon and Ali Farhadi. ‘YOLOv3: An Incremental Improvement’. In: *arXiv* (2018).
- [12] MathWorks. *MATLAB*. 2021. URL: <https://se.mathworks.com/products/matlab.html> (visited on 4th May 2021).
- [13] Anomaly. *Understanding Cross-Correlation, Auto-Correlation, Normalization and Time Shift*. 2016. URL: <https://anomaly.io/understand-auto-cross-correlation-normalized-shift/index.html> (visited on 28th Apr. 2021).
- [14] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. <http://pjreddie.com/darknet/>. 2013–2016.

Appendix

A GitLab repository

All code for the implementations can be found in this repository: <https://git.gvk.idi.ntnu.no/jakobfk/2021-cv-group1>