TÉCNICO LISBOA

Tracking animals in underwater videos

João Santos Teixeira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Alexandre José Malheiro Bernardino
Prof. Helena Sofia Andrade Nunes Pereira Pinto

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Prof. João Paulo Salgado Arriscado Costeira

December 2020

Acknowledgments

I would like to thank my supervisors Prof. Alexandre Bernardino and Prof. Sofia Pinto for guiding me throughout this last year, sharing their knowledge and experience, and for all the time spent during this process.

I would also like to thank Ocenário de Lisboa for the collaboration and the interest shown in the project as it kept me motivated to work in such an interesting real life application.

Finally, I want to thank my family for always supporting me and helping me achieve my goals.

Abstract

Object tracking using video is an important tool that has applications in many different areas. In this work, we focus on tracking fishes, using videos recorded in the tanks of Oceanário de Lisboa, an aquarium that recreates real-life underwater conditions of different habitats. Having a reliable tracking system can help automate the monitoring of the tanks and allows the development of more complex systems like anomalous behaviour detection. We present a system that detects and tracks fish of different species in two tanks with different characteristics. There are many challenges that complicate this task like the frequent occlusion of the targets, the presence of schools or high visual similarities between different individuals. Several state-of-the-art detectors were tested to assess their performance in the two different tanks and a tracker based on color and position was developed to associate detections to existing tracks. We evaluate different variants of the system and optimize the parameters for each environment. New datasets were manually built using one video for each environment.

Keywords

Object tracking, Object detection, Fish, Video

Resumo

O seguimento de objetos através de vídeo é uma ferramenta importante com aplicações em diversas áreas. Neste trabalho, vamos focar-nos no seguimento de peixes, usando vídeos gravados nos tanques do Oceanário de Lisboa, um aquário que recria condições subaquáticas reais de diferentes habitats. Ter um sistema de seguimento fidedigno pode ajudar a automatizar a monitorização dos tanques e permite o desenvolvimento de sistemas mais complexo tais como detetores de comportamentos anómalos. Apresentamos um sistema que deteta e segue peixes de diferentes espécies em dois tanques com características diferentes. Existem vários desafios que complicam esta tarefa como a frequente oclusão entre peixes, a presença de cardumes ou a grande semelhança visual entre diferentes indivíduos. Vários detetores do estado-da-arte foram testados para avaliar os seus desempenhos nestes dois tanques e um seguidor foi desenvolvido baseado em posição e cor para associar as detecções às trajectórias existentes. Avaliamos diferentes variantes do sistema e optimizamos os parâmetros para cada ambiente. Novos datasets foram construídos manualmente utilizando um vídeo para cada ambiente.

Palavras Chave

Seguimento de objectos, Deteção de objectos, Peixes, Video

Contents

1 Introduction **1**

1.1 Motivation . . . 3

1.2 Problem Scope and Challenges . . . 4

1.3 Objectives . . . 6

1.4 Contributions . . . 7

1.5 Outline . . . 7

2 Background **9**

2.1 Videos . . . 11

2.2 Object detection . . . 12

2.3 Tracking . . . 13

2.4 Deep learning . . . 14

3 Related work **17**

3.1 Pre-processing . . . 19

3.2 Object detection . . . 20

3.2.1 Background Subtraction with Classical Methods . . . 20

3.2.1.A Gaussian Mixture Models . . . 20

3.2.1.B Sigma Delta . . . 21

3.2.1.C Adaptive Kernel Density Estimation For Background Subtraction (KNN) . . . 21

3.2.1.D ViBE . . . 22

3.2.1.E Pixel-Based Adaptive Segmenter (PBAS) . . . 23

3.2.1.F Lobster . . . 24

3.2.1.G Pixel-based Adaptive Word Consensus Segmenter (PAWCS) . . . 24

3.2.1.H SubSENSE . . . 25

3.2.1.I GSOC . . . 26

3.2.2 Background Subtraction with Deep Learning . . . 26

3.2.2.A Deep background subtraction with scene-specific CNN . . . 26

3.2.2.B Deep CNN for video sequence background subtraction . . . 26

3.2.2.C FgSegNet . . . 27
3.2.3 Deep Learning Based Detectors . . . 27
3.2.3.A YoloV3 . . . 28
3.3 Tracking . . . 29
3.3.1 KCF . . . 29
3.3.2 Hungarian algorithm with Kalman Filter . . . 29
3.3.3 Interactive Multiple Model . . . 30
3.3.4 Tracking crowds . . . 31
4 Implementation **33**
4.1 Overall Architecture . . . 35
4.2 Color equalization . . . 35
4.3 Object Detection . . . 37
4.3.1 GSOC . . . 37
4.3.2 YoloV3 . . . 39
4.4 Tracking . . . 39
4.4.1 Color history . . . 41
4.4.2 Temporary tracks . . . 42
4.4.3 Movement prediction . . . 42
5 Evaluation **43**
5.1 Datasets . . . 45
5.2 Experiment A: object detection . . . 47
5.2.1 Experiment A1: background subtraction . . . 47
5.2.2 Experiment A2: bounding box prediction . . . 49
5.3 Experiment B: tracking . . . 50
5.3.1 Evaluation Metrics . . . 51
5.3.2 Algorithm Variants and Parameter Tuning . . . 51
6 Results **55**
6.1 Results of experiment A . . . 57
6.1.1 Results A1: background subtraction . . . 57
6.1.2 GSOC Tuning . . . 60
6.1.3 Results A2: bounding box prediction . . . 63
6.2 Results of experiment B . . . 64
6.2.1 Tracking using GSOC detector . . . 64
6.2.2 Tracking using YOLO detector . . . 68
6.2.3 Tracking: GSOC vs YOLO . . . 71

7 Conclusion **75**

7.1 Future work 77

List of Figures

1.1 The environments of the two tanks where the videos were recorded. 5

1.2 Examples of fish schools and occlusions. 6

1.3 The same coral at different times with different shapes. With the current, the corals move around and should not be mistaken as fish moving. 6

1.4 Simplified overview of the proposed system architecture. 7

2.1 Left: Example of a frame at 720×480 resolution. Right: Portion of the image zoomed in, illustrating the pixels organization. 11

2.2 (a) Segmentation mask; (b) Background subtraction applied to the frame in 2.1 13

2.3 Output for the frame in 2.1 after running it through the object detector YoloV3 [27], with the bounding boxes shown in pink. 14

4.1 System architecture. The user first selects the targets of interest initializing the tracks. Then, each frame F at time t may be applied the color equalization resulting in F'^t. The frame is then used in the object detection module that can use two different techniques producing a list of detected objects d^t. In the end, the tracker uses the active tracks $\hat{T}^t$ and tries to match them with each detection in d^t. 36

4.2 Result of applying the color equalization technique to a frame from the video of the main tank. Left: original; right: after equalization. 37

5.1 Top row: Original frame and corresponding segmentation; Bottom row: Frame and segmentation after augmentation using width shifting, horizontal flipping and zoom. 46

6.1 F1-score testing results for the algorithms in the coral reef tank on the left and the main tank on the right. 57

6.2 F1-score comparison for the single-pass and two-pass variants for the coral reef tank at the top and for the main tank at the bottom. 58

6.3 Results of background subtraction algorithms for the coral tank using the 24^{th} frame. On the top row there is the image on the left and the manually labeled mask on the right. On the second row, we have the result of FgSegNet on the left and the result of two-pass PAWCS on the right. On the third row we have the result of GSOC on the left and the two-pass GSOC. 59

6.4 F1-score comparison for the use of color equalization for the main tank. The variant without equalization used with FgSegNet is the single-pass variant and the rest of them use the two-pass variant. 60

6.5 Results of background subtraction algorithms for the main tank using the 24^{th} frame. On the top row there is the image on the left and the frame after color equalization on the right. On the second row, we have the manually labeled mask on the left and the result of FgSegNet using the equalized frame on the right. On the third row we have the result of two-pass GSOC on the left and the two-pass KNN on the right, both using the equalized frame. 61

6.6 Evolution of F1-Score with the number of background color samples for each environment using GSOC. 62

6.7 Evolution of processing time with the number of background samples for each environment (five minutes video each) using GSOC. 63

6.8 Example of the use of temporary tracks in the coral reef tank. In the top row, without using temporary tracks, the original target (identified as number 1) lost its target as another track that was new got the association. In the bottom row, using the temporary tracks (shown as red), the original track was able to keep track of the original target. 66

6.9 Precision and Success plots using GSOC for the training dataset in the coral reef tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter. 66

6.10 Distribution of the ratio of predicted boxes in each track with IOU higher than 0.33 for the different system implementations using GSOC for the training dataset. Each white dot in the same system implementation represents a different tested track in the coral reef tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter. 67

6.11 Example of using Kalman Filter in the main tank for movement prediction for the track 1. The red star is the position considered for the position similarity. In the top row, without using movement prediction, when detection fails the position remains the same in those frames, allowing incorrect associations afterwards. In the bottom row, using movement prediction, the position keeps being predicted until it eventually detects the fish and recovers the original target. 68
6.12 Precision and Success plots using GSOC for the training dataset in the main tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter. . . 69
6.13 Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system implementations using GSOC for the training dataset. Each white dot in the same system variant represents a different tested track in the main tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter. 69
6.14 Precision and Success plots using YOLO for the training dataset in the coral reef tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter. 70
6.15 Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system using YOLO for the training dataset. Each white dot in the same system variant represents a different tested track in the coral reef tank.B - baseline, H - average histogram, T - temporary tracks, K - kalman filter. 71
6.16 Precision and Success plots using YOLO for the training dataset in the main tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter. . . 72
6.17 Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system implementations using YOLO for the training dataset. Each white dot in the same system variant represents a different tested track in the main tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter. 72
6.18 Precision and Success plots for the system implementations with each object detection technique for the testing dataset in the coral reef tank; left: as a function of location error; right: as a function of overlap. 73
6.19 Precision and Success plots for the system implementations with each object detection technique for the testing dataset in the main tank; left: as a function of location error; right: as a function of overlap. 74

List of Tables

5.1 Description of the videos used for testing. 45
5.2 Specifications of the computer used in the development and testing of the tracking system. 45
5.3 Background subtraction dataset 47
5.4 Tracking dataset 47
5.5 The different variants of the background subtraction tests using the testing dataset and in which tanks they are used. 48
5.6 Background subtraction algorithms used in the comparative study 49
5.7 FgSegNetv2 training time 49
5.8 The different variants of the algorithm to be tested using the validation dataset for each tank. In each variant, the corresponding parameters are tuned. 51
5.9 Kalman tuning 53

6.1 Default values for the GSOC parameters. 62
6.2 Values obtained in the parameter fine tuning for GSOC 63
6.3 Performance of YOLO object detector for both environments. 64

1

Introduction

Contents

1.1 Motivation 3
1.2 Problem Scope and Challenges 4
1.3 Objectives 6
1.4 Contributions 7
1.5 Outline 7

1.1 Motivation

In order to study the behaviour of known fish species, it is important to observe them through long periods of time to identify patterns and reoccurring actions. Some of the behaviours that the fish exhibit may happen more regularly than others and vary between different species. How fish feed themselves, how they reproduce or which mechanisms they use to avoid their predators are all important behaviours that biologists are interested in studying to better comprehend the aquatic wildlife ecosystem. There are some difficulties in observing these animals given that they live underwater, and sometimes at very low depths, which limits the time that specialists can spend in direct contact with them.

Another very important topic related to marine wildlife is species extinction risk and endangerment. It is important to preserve all species and their habitats to avoid reaching a point where species disappear forever, particularly if the cause of extinction is human. If we combine extinction with big decreases in population of marine species we get a phenomenon called marine defaunation. According to a study [22], marine defaunation levels are still considered to be low when compared to terrestrial defaunation, which has been happening for a longer period of time. When the study was made (2015), there were 15 global extinction of marine species recorded by the International Union for Conservation of Nature (IUCN), whereas there were 514 for terrestrial species. This is a big difference given that there were only 6 times more terrestrial animals cataloged than marine ones [23], but there were also many marine species that were not included due to lack of data. By taking the current values of marine defaunation and comparing them to when the same values occurred in terrestrial defaunation evolution, it is possible to see that a large increase in the rate of extinction of marine species could happen in the future if no actions to prevent it are taken.

One method to study the animal behaviour and swimming patterns of certain endangered species is through tracking. Fish tracking can be done in different ways, each with their own advantages and disadvantages. One tracking approach is by using Global Positioning System (GPS) for animals that come to the surface frequently. A GPS receiver is attached to a certain fish as well as a transmitter. Then, the recorded position values are used to determine the trajectory of the fish. This way of tracking has some advantages like being able to follow long distances and over large periods of time. It also allows to detect each fish individually even if they are very close together. On the other hand, it is necessary to catch and attach the device to each individual fish that we want to track, which can be both hard to do and uncomfortable for the fish. As it is usually attached externally, it may get separated from the fish, losing its track. Also, smaller fish might not be able to have that type of device attached to them, and it would not work well for species that never come to the surface. Another possible way of studying animal behaviour is through video based tracking animals captive in tanks that try to replicate as close as possible the maritime environment. The recorded videos are automatically processed and then computer vision techniques are used to locate and track the fish. This approach has the benefit of being

able to track every fish that swims in the view field of the cameras, and is less invasive to the fish as they can move freely without any attachments. On the downside, the detection of each fish separately can be problematic when they are behind each other and the tracking space is limited to the area captured by the camera. Also, a regular camera would not be able to capture anything in low lighting conditions.

Oceanário de Lisboa is an organization whose a goal is to educate the general population about ocean conservation. They are also responsible for breeding programs and conservation medicine studies, and participate in the assessment of risk extinction of the species. Oceanário de Lisboa has a public aquarium with more than 30 tanks and has more than 500 species of animals and plants from different habitats. Some tanks mix together fish that would not be seen together in the wild due to living in different locations, so their interactions and behaviours around each other can also be studied there. It is important that the biologists continuously observe and monitor the tanks to make sure all animals are well and safe, and not showing anomalous behaviour. But having that many tanks makes it difficult to keep track of everything. Having a system that can automatically keep track of the fish could be an important tool in the observation task. This would make the work of specialists more efficient, since they would not need to spend as much time observing the tanks.

1.2 Problem Scope and Challenges

In figure 1.1, we can see the two environments inside the tanks in which the videos were recorded. They have different characteristics. The first environment is the main tank, which consists in a large open space with different types of fish, from medium sized to big ones, including sharks, tunas and rays, among others. Given the large size of the tank, both in depth, width and length, it is difficult to capture everything, and also the distance which the camera is able to capture is relatively short meaning that fish swimming away from the camera will get lost. The fish present in this tank share a lot of similarities in their colors, being mainly gray, and their movement is also slow and without abrupt changes in direction. There is also the presence of very big schools swimming around that can make the tracking process difficult.

The coral reef tank, on the other hand, is a lot smaller and the distance between the glass and the back of the tank is very short meaning the fish do not disappear in the distance. The fish in this tank are small in size and their movement patterns are less predictable as they change directions easily. They can also appear to be very quick given that they are closer to the camera due to the smaller size of this tank. There is a lot more color present in this environment, both in the coral reefs and in the fish.

As already mentioned, these specific environments present different challenges to tracking, like some of them are shown in Figures 1.2 and 1.3. First, there is the problem of occlusion, that happens when fish cross paths and momentarily hide behind other fish/objects, shown on the right side of Figure 1.2.

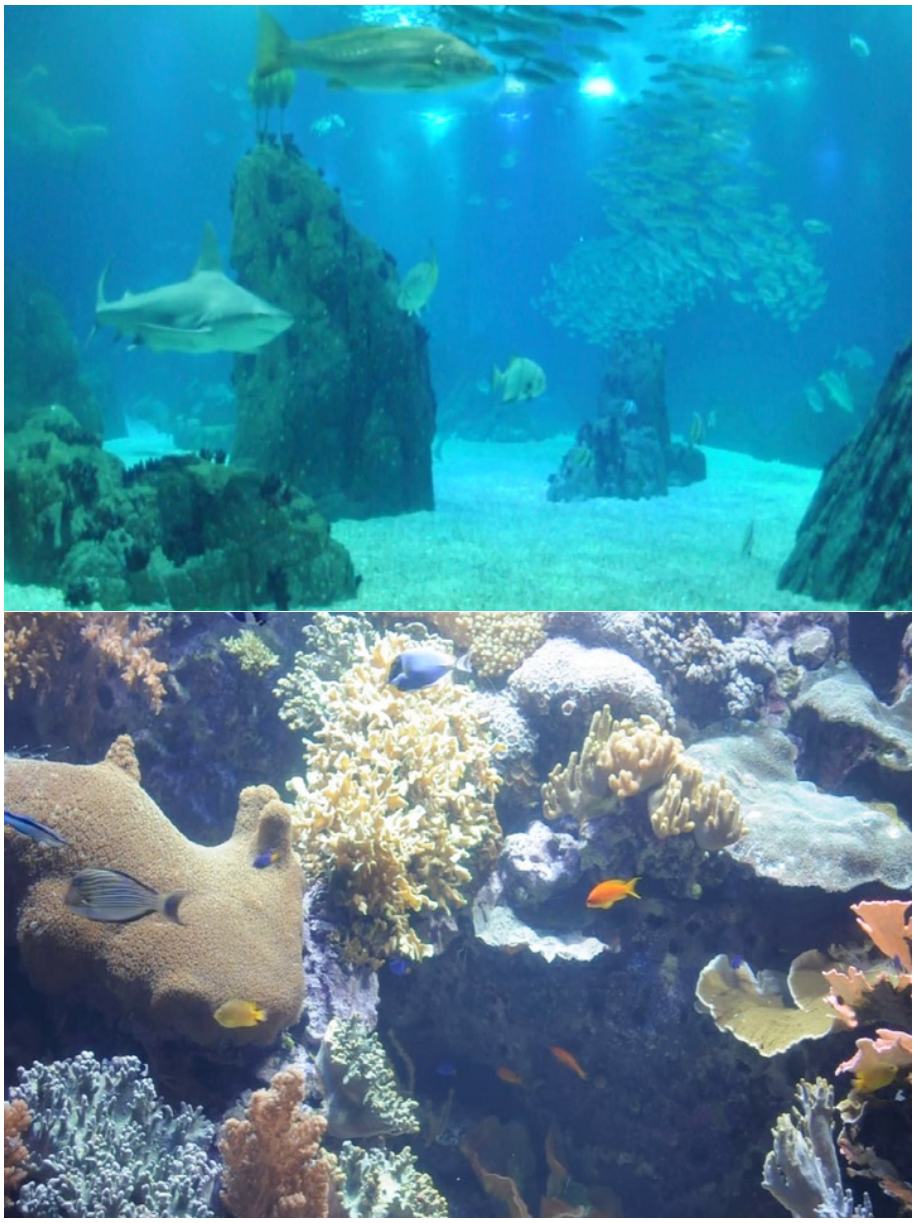


Figure 1.1: The environments of the two tanks where the videos were recorded.

Figure 1.2: Examples of fish schools and occlusions.

Figure 1.3: The same coral at different times with different shapes. With the current, the corals move around and should not be mistaken as fish moving.

Then, there is the presence of schools, show on the left side of Figure 1.2, which is heavily linked with the occlusion problem but can bring other problems in detecting other fish, as we will see later. Having a big group of fish swimming very close to each other is one of the main obstacles for the main tank. Next, we have the movement of the coral reefs with the currents, shown in Figure 1.3, that could be mistaken as fish moving around.

1.3 Objectives

In this thesis, we want to develop a system that tracks fish inside tanks. The target fish is manually identified in a frame and the system must track the fish in the following frames until it swims out of the field of view captured by the camera. The tracking system has to be robust in order to track fish of different sizes, colors, shapes and also has to be able to handle different tank conditions that include different depths, lengths and various types of coral. Additionally, some of the challenges like occlusion, light variations, fish swimming at different speed and acceleration, changes in apparent size of the target have to be handled in order to adequately track the chosen target.

In this work, we assume that each tank has a single camera, but future solutions could consider the existence of multiple cameras to enlarge the working space and allow more robust and precise tracking.

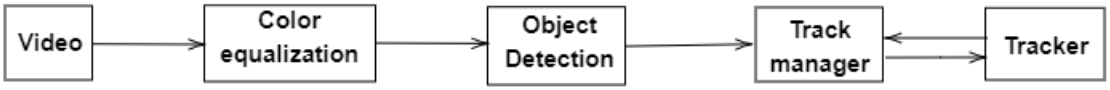


Figure 1.4: Simplified overview of the proposed system architecture.

1.4 Contributions

We develop a simple fish tracking system using video and computer vision techniques, that can work with multiple species of fish and in different types of habitats. First, it can equalize the colors of the underwater frames, if necessary. Next, it is able to detect fish using two different methods that can be used alternatively, one using background subtraction and the other using bounding box prediction. Then, it tracks the fish through data association, by matching each detected object with an existing track after comparing their similarities in position and color distribution. The tracking mechanism uses three tracking features that give more robustness to the tracker: color history, temporary tracks and movement prediction. A simplified system architecture of the proposed system is shown in Figure 1.4.

We also present a comparative study on multiple state-of-the-art object detection approaches to understand which one works better in the two considered underwater scenarios. It includes traditional background subtraction approaches as well as one using deep learning.

Finally, two small datasets were manually created, one regarding fish detection and the other regarding fish trajectories.

1.5 Outline

In chapter 2 we explain some of the concepts and theory needed to understand the rest of the work. Then, in chapter 3, we review the state-of-the-art approaches relevant to our problem. In chapter 4, we present the implementation of the system developed, that details all modules of the pipeline. In chapter 5, we explain the evaluation procedures, the metrics and the datasets used in each evaluation. Next, in chapter 6, we present the results and then finally in chapter 7, we make a summary of our work and suggest additional steps that could be implemented in future work.

2

Background

Contents

2.1 Videos 11
2.2 Object detection 12
2.3 Tracking 13
2.4 Deep learning 14

In this section, some of the concepts that are needed to understand the rest of the work will be presented.

2.1 Videos

One of the fundamental concepts that this paper will rely on is what videos are and how to extract information from them. A video can be described as a collection of temporally ordered images, called frames, that were captured by some type of recording device. A video can be recorded in different resolutions which translates into having different widths and heights. The resolution of the video corresponds to the number of pixels distributed horizontally and vertically. Higher resolution usually translates to better quality because there are more pixels to portray what was captured. Another important aspect is the number of frames captured in each second which is commonly known as Frames Per Second (FPS). Higher FPS makes the video look smoother and facilitates the job of tracking algorithms because the targets have smaller motions between two frames. In this work, we deal with videos of 720×480 resolution at 24 FPS. A pixel corresponds to a location of the frame and has a color associated with it. By combining multiple of them in a grid, an image is created, as illustrated in 2.1.

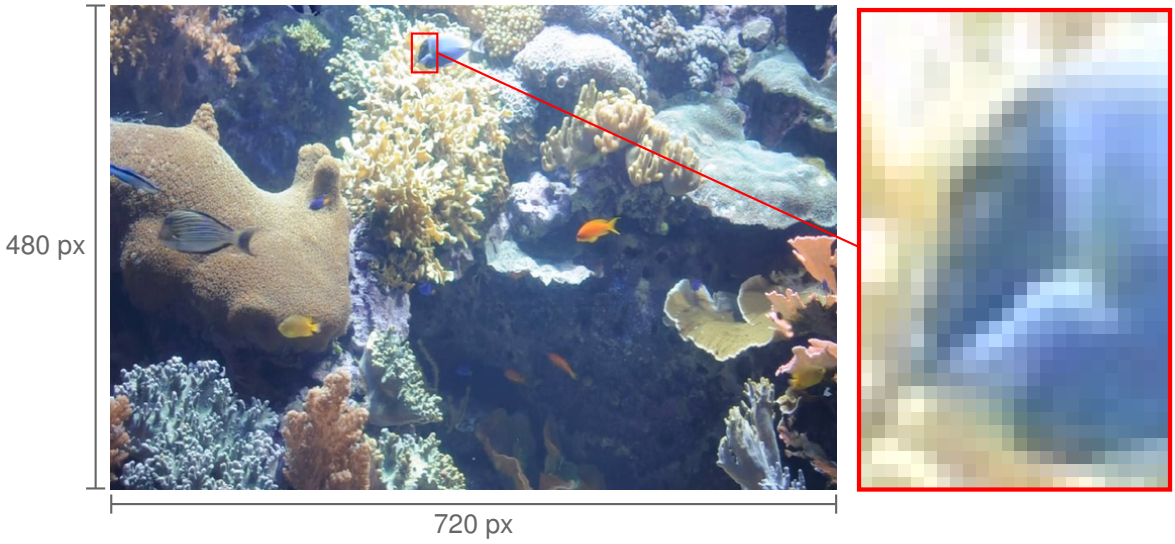


Figure 2.1: Left: Example of a frame at 720×480 resolution. Right: Portion of the image zoomed in, illustrating the pixels organization.

There are different color spaces that can be used to describe a color, resulting in monochromatic or colored images. Grayscale is a very simple model that only uses one value per pixel to describe the intensity of the color, that can range between black and white and everything in between is a different shade of gray.

There are multiple colored color spaces, being Red, Green, Blue (RGB) the most known. RGB is an

additive color model. Each channel has the intensity value of the correspondent primary color. In this color space, we can't separate the brightness from the color itself, so changes in light conditions will result in different values across the channels.

Another common color space is Hue, Value, Saturation (HSV). Hue is the type of color, Saturation is the intensity or purity of the color and Value determines how light or dark the color is. With this representation, we have a separate channel for the luminance. There is also the YCbCr color space, that, like HSV, has a channel for luminance, Y, and two others: Cb and Cr where both encode the chrominance.

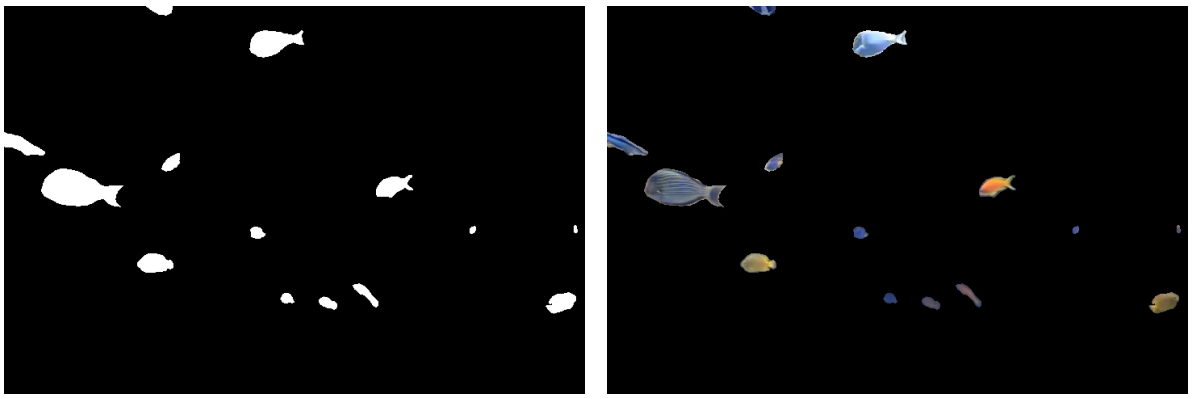
CIELAB, or L*a*b*, is another color space, which has a channel L* for the lightness and channels a* and b* for the color. The first color channel a* corresponds to the variation between green and red, and channel b* corresponds to the variation between blue and yellow. Like in HSV and YCbCr, there is a separation between the color components and the light intensity.

2.2 Object detection

Object detection is one of the core parts of this work, given that we have to detect the animals in order to track them. There are two main lines of research relevant in this particular topic: background subtraction and object detectors using neural networks.

Background subtraction algorithms can come from different approaches, including probabilistic, sample based, code-book based and through deep learning. Typically, a background model of the scene is computed and then is used to compare with the new incoming frames. This leads to the classification of each individual pixel as either background or foreground, depending on the difference between the values of the pixels in the current image and the background model. The pixels classified as foreground constitute the foreground segmentation mask. This process is demonstrated in Figure 2.2. The background model must be initialized in a prior stage to tracking. The initialization can vary between approaches and the presence of foreground objects makes this task harder. After the background model is initialized, it can be updated online to reflect long term changes in the scenario. Those changes can come from simple illumination changes, that result in slight color variations, or moving elements from the background, like coral moving with the current. There is the need to balance the background update rate so it is neither too fast, becoming very sensible to small changes, nor to slow, which can lead to an outdated background model.

Other types of object detectors using neural networks are getting more relevance recently. These methods work by training a neural network to identify objects' locations and some times classify them into predefined categories. After training, the application of the network to new frames of video outputs the bounding box locations of the objects, as shown in Figure 2.3. There are many challenges in



(a) **(b)**

Figure 2.2: (a) Segmentation mask; (b) Background subtraction applied to the frame in 2.1

working on these approaches like designing the architecture of the network or getting a good dataset with numerous different examples to train with. One upside of using these approaches is that objects that are touching each other on the frame can still be individuated, as two different objects, whereas in background subtraction they are frequently identified as a single one.

2.3 Tracking

Tracking can be described as a way of following the trajectory of objects until they get out of the field of view. A multi-target tracking system should also be able to correctly differentiate between different objects and maintain the same label for each object in its lifespan. There are many factors that can make this task hard such as:

Occlusion It happens when tracked object is covered by other objects or the scene, meaning that parts of the object can't be observed and, in cases of full occlusion, it's impossible to see the whole object .

Light variations Given that the color being captured is influenced by the lighting of the scene, variations of the lighting can affect the performance of the tracker recognizing the objects.

Scale variations The objects may appear to vary in sizer due to getting closer or further away from the camera

Rotations The objects may also look different across the frames because of rotations, so it is important to take into account that some objects look distinct from different perspectives.

Unpredictable movement Fast changes of directions and speed also make it difficult to predict their movement and future positions.

Usually, the object is identified in a frame and a representation model of that object is created. In

Figure 2.3: Output for the frame in 2.1 after running it through the object detector YoloV3 [27], with the bounding boxes shown in pink.

the subsequent frames, the algorithm tries to identify the area of the frame that best approximates the model created (top-down) or tries to match the detected objects with the existing tracks (bottom-up). Some algorithms even perform a model update throughout the video to include changes to the target that happen during the tracking process. It is important that the tracker can handle occlusions well since fish tend to cross paths very often and even swim behind rocks or through the coral reefs.

2.4 Deep learning

Recently, deep learning has gained a lot of traction in many different tasks of computer vision. One reason behind it, is the evolution of computational power that is allowing these approaches to run in many platforms with reasonable efficiency. As deep learning techniques will be used in this work, a short introduction on the subject will be given.

Neural networks (NN) are inspired by how the brain works. A neuron is the elementary computational unit of the brain that receives signals from other neurons, processes them and then propagates an output signal to some other different neurons. A NN is a graph where the neurons are nodes. A common architecture of NNs is based on layers. In such architectures, the neurons are organized into layers, where each neuron has no connection with the others in the same layer, only with ones from other

layers. The inputs of the neurons of one layer are the outputs of the neurons from the previous layer. When all neurons of a layer are connected to all the nodes from the next, it is called a fully-connected layer. Layers are grouped into three categories: input layer, hidden layer and output layer. Each neuron has weights associated to them in equal number to the inputs received. The network operates by receiving an input in the input layer and multiplying those values with the weights of the neurons of the following hidden layer. Then, the resulting values go through an activation function, which is where the non-linearity of these models comes from. There are different types of activation functions, like sigmoid or Rectified Linear Unit (ReLU). The output of each neuron is used as the input for the next layer and these steps repeat until it reaches the output layer. The number of layers and neurons per layer can vary a lot, and while having more neurons means more complex models can be learned, it also means that it is easier that over-fitting occurs. To train the weights of the network, a training dataset is needed. The learning process requires running the instances of the dataset through the network multiple times and calculating an error functions (also known as loss function) by comparing the final output of the network, also called the prediction, with the ground-truth value, a value that is considered to be the right answer. A gradient of the loss function with respect to each of the weights is computed, and the weights are updated by using back-propagation, meaning the update starts at the last layer going backwards until reaching the first one.

There are special types of neural networks designed to process image data inputs, called Convolutional Neural Networks (CNNs or ConvNets). Instead of the inputs being single vectors like in NNs, they are now 3-dimensional (width, height and color channels). So each layer works with 3D volumes. Instead of each neuron being connected to all the pixels of the image, they only have a connection with a smaller region of the image, where a convolution is applied between the image and the filter. We can look at the learned filters as different patterns or features learned. This is where these approaches may have an advantage over traditional techniques, as the features are learned automatically instead of being created manually. In most cases, the features learned at a certain position are probably useful in the other areas, so there should not be the need to relearn the same things in different neurons. Instead of having different filters for each neuron, we can share the filters across all neurons.

3

Related work

Contents

3.1 Pre-processing 19

3.2 Object detection 20

3.3 Tracking 29

In this chapter, a revision of the state-of-the-art will be made, divided into three categories: image pre-processing, object detection and tracking.

3.1 Pre-processing

Pre-processing techniques might be important to use in some scenarios where the colors may change due to specific conditions of the working environment, so a review will be made related to color spaces and color manipulations to improve the quality of the underwater images.

One important aspect to take into account is that the colors of underwater images do not reflect the exact colors of the objects. This happens because of the way water absorbs light. Colors with bigger wavelengths, like red, are absorbed at shorter depths than colors with shorter wavelengths, like blue. That is why underwater images are very blueish. However, there are ways to try to recreate the original colors after recording underwater like in [17] and [14]. Even though it is not possible to obtain the exact same color, some improvements can be made in order to get values closer to the true color of the objects.

In histogram equalization, all pixels are subject to the same transformation but when the image has some areas that are a lot darker/lighter than the rest of the image, an equal equalization across the entire image doesn't provide good results. Adaptive Histogram Equalization (AHE) [24] only uses the values from a square neighbourhood for each pixel to build the histograms to equalize.

In [17] is presented a video based system to detect fish. The authors of that work transformed the color space to CIELAB and then applied a histogram equalization technique called Contrast Limited Adaptive Histogram Equalization (CLAHE) on the luminosity channel of the new color space. This approach was used in a similar context to ours, in underwater recordings, although in their videos the water was more blurred. CLAHE uses the same principles of AHE but clips the histogram using a predefined value and the part of the histogram that exceeds the clipping limit is redistributed equally among all the bins.

Another approach explained in [14], named Unsupervised Color Correction Method (UCM), also tries to enhance the quality of underwater images. The first step is to divide the R channel and G channel by their averages and multiplying by the average of the B channel, given that the underwater images always have high blue values. Then contrast correction is applied in the the R and B channels. As the red has the lower values, a contrast correction is applied to the upper side of the histogram, meaning that the histogram is stretched to the maximum limit 255 but is not stretched in the lower side. On the B channel the opposite occurs, the correction is applied only to the lower side of the histogram. Then, the image is converted into HSI and correction on both sides of S and I histograms is performed.

3.2 Object detection

As previously said, there are two object detection approaches that we considered relevant: either using background subtraction (using classical methods or deep learning) or object detectors based on a neural network. So, in this section we will review some works in those three categories.

3.2.1 Background Subtraction with Classical Methods

We will start by reviewing background subtraction algorithms using classical methods.

3.2.1.A Gaussian Mixture Models

One of the most studied approaches is the Gaussian Mixture Models (GMM). The first background subtraction method based on GMM's was introduced in [10] but it has seen many different variations and improvements over the years. One of the most relevant works is [33]. The main idea is to use Gaussians to describe how a pixel's color changes over time. This change can be caused by moving objects, dynamic background like trees, and changes in lighting conditions. The color of each pixel is modeled by a mixture of K Gaussians, given the observed color intensities over time. The probability of observing a new value x^t is given by

$$P(x^t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(x^t, \mu_{i,t}, \Sigma_{i,t}) \tag{3.1}$$

where $\omega_{i,t}$ is the weight estimation of the i^{th} Gaussian, $\mu_{i,t}$ is the mean value of the same Gaussian, $\Sigma_{i,t}$ is the covariance and η is the Gaussian probability density function. For every new pixel value x^t, it is checked if any of the current Gaussians is a close representation of the the new value. If there is no match, the Gaussian with the lowest importance (smallest $\omega_{i,t}$) is replaced by a new one corresponding to the new observation, starting with a high variance and a small weight. The weights have to be updated continuously, so that the most matched Gaussians have higher weights. To do that, a learning rate α is used to make an average of the current weights and $M_{k,t}$, a vector with zeros in all positions except for the position of the matched model, which is 1. Only the μ and σ^2 of the matched Gaussian are updated, according to a learning rate ρ.

In [37], another improvement to GMM was made by adopting an online procedure to adapt the number of components of the mixture used automatically.

3.2.1.B Sigma Delta

The algorithm [21] works by using two Σ-Δ filters to compute two statistics for each pixel at each frame: the motion likelihood, Δ_t, and the variance, V_t. The motion likelihood, for each pixel x at frame t, is given by:

$$\Delta_t(x) = |M_t(x) - I_t(x)| \tag{3.2}$$

where $M_t(x)$ is the estimated background value and $I_t(x)$ is the input. The first value of estimated background $M_0(x)$ is initialized equal to $I_0(x)$. Then, it is updated as following:

$$M_t(x) = \begin{cases} M_{t-1}(x) + 1, & \text{if } M_{t-1}(x) < I_t(x) \\ M_{t-1}(x) - 1, & \text{if } M_{t-1}(x) > I_t(x) \end{cases} \tag{3.3}$$

The variance $V_t(x)$, that represents how "moving" or "stationary" the pixel is, is initialized equal to $\Delta_0(x)$ and then, for each pixel x where $\Delta_t(x)$ is non-zero, is updated according to:

$$V_t(x) = \begin{cases} V_{t-1}(x) + 1, & \text{if } V_{t-1}(x) < N * \Delta_t(x) \\ V_{t-1}(x) - 1, & \text{if } V_{t-1}(x) > N * \Delta_t(x) \end{cases} \tag{3.4}$$

where N is a user-defined variable. Then, if $\Delta_t(x)$ is less than $V_t(x)$, the pixel is classified as background, and as foreground otherwise. The next step is to perform spatial regularization on the output of the binary classification, which is described in [21].

3.2.1.C Adaptive Kernel Density Estimation For Background Subtraction (KNN)

This technique was first introduced in [9] and then improved in [38]. It is possible to compute the density estimation of the background model, using an uniform kernel, at time t for a certain pixel x, using a set X_T with the last T intensity values, containing values corresponding both to the background and foreground (BG + FG), by counting the number of values k that are inside a volume V with diameter D according to

$$\hat{\rho}(x^t|X_T, \text{BG} + \text{FG}) = \frac{1}{TV} \sum_{m=t-T}^{t-1} K\left(\frac{||x^{(m)} - x^t||}{D}\right) = \frac{k}{TV} \tag{3.5}$$

where kernel function $K(u)$ = 1 if $u < 1/2$ and 0 otherwise. One of the improvements was to use a kernel size D that is not fixed, and becomes larger in areas with fewer number of samples and smaller in areas with more samples so there is always a fixed amount of k samples covered (10% of T). There is a threshold to which the densities are compared with, c_{thr}, that is going to be equivalent to $1/V$. In this version, there is a binary indicator $b^{(m)}$ for each sample in X_T showing if the sample belongs to the background model or not. To classify if a new pixel intensity as background or foreground, a density estimation is performed only using the samples belonging to the background model. If the value of the

estimation is higher than the threshold, the pixel is classified as background. After that, a new estimation is computed but this time using all the samples in X_T and if the the estimated density is higher than c_{thr} then the new sample value is added to the background model with $b_m = 1$, replacing the oldest one.

3.2.1.D ViBE

ViBE [2] is also a very well known background subtraction algorithm. The authors separate the approach in three parts: pixel model and classification, model initialization and, at last, model update. Regarding the model, N background color samples are kept for each pixel and, using a conservative update policy, only background samples are stored. The euclidean distances between a new pixel color and each of the current samples are computed to check if there are enough samples in a given radius R (which they considered to be 20). If the number of nearby samples is higher than a certain threshold, $\#min$, then the new pixel value is classified as background. The model initialization, is done using only the first frame. Instead of using the concept of temporal information, they considered that the color distribution of a pixel over time is similar to the observed values of the neighbouring pixels. The first samples for each pixel correspond to some random neighbours' value.

In the updating step, there are three components. The first one is a memoryless update policy. Instead of replacing the oldest samples first, a sample is selected at random following an uniform probability density function. The probability of a sample that appeared at time t still being in the samples group at time $t + dt$ is given by

$$P\left(t, t+dt\right) = \left(\frac{N-1}{N}\right)^{(t+dt)-t} \tag{3.6}$$

where N is number of samples kept for each pixel. Such probability can be rewritten as

$$P\left(t, t+dt\right) = e^{-\ln\left(\frac{N-1}{N}\right)dt} \tag{3.7}$$

and this way the lifespan of a sample is considered to have an exponential decay. With this formula, the probability of a sample still being in the model is independent of when it was added, thus being a memoryless update. The next component is time subsampling. Instead of updating the background pixel model every frame, for each pixel classified as background, a random probability is calculated to determine if the model is to be updated or not. The authors used a chance of one in sixteen of a sample being used to update the model. The last component is spatial consistency through background samples propagation. If, for a pixel x, a new value $v(x)$ was chosen to update the samples of x, then the samples of a random neighbour of x are also updated. This approach is not deterministic, making it, at the time, unique. The authors tested the system varying the parameters mentioned above and concluded that N = 20 and $\#min$ = 2 provided good results. The authors tested the approach using two different video sequences and compared the results with other algorithms. ViBE (both RGB and grayscale versions)

outperformed all the other algorithms in terms of accuracy and most of them in processing speed, getting around 200 fps with RGB and 250 fps with grayscale.

3.2.1.E Pixel-Based Adaptive Segmenter (PBAS)

Another background subtraction algorithm is PBAS [13], an algorithm that borrows some of the ideas of ViBE [2]. This approach creates a background model $B(x_i)$ and maintains, for each pixel (unlike ViBE), a decision threshold $R(x_i)$ and a learning threshold $T(x_i)$. Similarly to ViBE, for each pixel, N samples are stored and to classify a new value as background/foreground we calculate the distance to the other samples. To update the model, after a pixel being classified as background, there's a probability $p = 1/T(x_i)$ that the update is performed, choosing one of the samples to replace it randomly using an uniform distribution. There is also the same probability p of a neighbouring pixel being updated, but in this approach the value to be added to the samples is the neighbours new value instead of the current pixel's value. The authors considered that the threshold $R(x_i)$ (the maximum distance to classify two samples as close), should adapt automatically depending on how dynamic that part of the scene is. To do this, an array of minimal distances is kept for each pixel and when an update is performed, the minimal distance between the new value and the other samples is computed and added to the array. Then, the average of the minimum distances is computed, and depending on its value, $R(x_i)$ may increase or decrease by a defined amount (5% according to the authors). The update of the learning rate $T(x_i)$ is also based on how dynamic the background is, using again the average of minimum distances. The learning rate has an upper and lower bound, tuned to 2 and 200 respectively. If the pixel was classified as background, the learning rate is decreased by $\frac{T_{dec}}{\bar{d}_{min}(x_i)}$ and if it was classified as foreground it is increased by $\frac{T_{inc}}{\bar{d}_{min}(x_i)}$ (T_{dec} was tuned to 0.05 and T_{inc} to 1). The algorithm works on each color channel separately and in the end merges the results. Besides the value v of the color, it also uses gradient magnitudes m, which means that each pixel is defined by $\{I^v(x_i), I^m(x_i)\}$ and the background samples are defined by $\{B^v_k(x_i), B^m_k(x_i)\}$. The authors evaluated and tuned the parameters using the dataset from the Change Detection Challenge [11]. The dataset is composed of videos with different challenges, including dynamic background and shadows. Given that this approach is pixel-based, the authors decided to use a median filter to smooth and reduce some noise and used a 9×9 filter. This approach got the first place in the challenge (2012 edition), having the best average ranking across the different categories and the best average ranking across the different metrics, averaging 48 fps. The tests showed that using the gradient magnitudes, the F1-score increased by 2% and the percentage of bad classifications decreased by 0.13%.

3.2.1.F Lobster

Lobster (LOcal Binary Similarity segmenTER) [30] uses the same approach of ViBE but with some adaptations. Instead of using only color intensities for samples, are also used Local Binary Similarity Patterns (LBSP). LBSP works by comparing the intensity of a pixel with its neighbours to obtain a binary code. Given a region R with center pixel c and P pixels in the neighbourhood, LBSP is defined as

$$LBSP_R(c) = \sum_{p=0}^{P-1} d(i_p, i_c)2^p \tag{3.8}$$

where i_p and i_c are the intensities of the pth pixel in P and the center pixel,with

$$d(i_p, i_c) = \begin{cases} 1 & \text{if } |i_p - i_c| \leq T_d \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

where T_d is a similarity threshold. To check if pixel is foreground or background, for each channel, first the L1 distance (instead of the L2 in ViBE) is used to compare the color intensity similarities between the model and the new sample. If the intensity distance is higher than a threshold, the pixel is classified as foreground. If not, the the Hamming distance is used between the LBSP using the current sample (as center pixel) and an LBSP computed using the model sample (as center pixel). Again, the distance is checked against a threshold to classify as foreground or background. At the end, the cumulative distances across all channels of both the color intensity and LBSP descriptors are checked against a new threshold for the total distance allowed.

3.2.1.G Pixel-based Adaptive Word Consensus Segmenter (PAWCS)

Another approach is PAWCS [31] following the work of [16] and [35], while also using some ideas of ViBE [2] and PBAS [13]. The background model uses a global dictionary and a local dictionary for each pixel with codewords to describe the samples using its colors, Local Binary Similarity Patterns (LBSP) features [4] and a persistence indicator which represents occurrence count and time stamps for when it was first/last seen. Unlike other works, the codewords are not clustered in the local dictionaries. Like some of the methods described previously, when a new frame is processed, each new pixel codeword is compared to the existing code words in the local dictionary trying to see if there is any similarity. Instead of checking if the number of close samples (distance smaller than a radius $R(x)$) is higher than a certain threshold, a persistence sum of the close codewords is computed. If the sum is lower than a threshold $W(x)$, the pixel is classified as foreground. The codewords are sorted by persistence to reduce some

similarity computations. The persistence of a codeword ω at time t is computed as

$$q(\omega, t) = \frac{n_{occ}}{(t_{last} - t_{first}) + 2 \cdot (t - t_{last}) + t_0} \tag{3.10}$$

where n_{occ} is number of occurrences, t_{first}/t_{last} is the time it was first/last seen and t_0 is a high constant value to prevent newly created codewords having a lot of importance. In the model update, there's a chance that the color representation of a codeword is updated when the local features are very similar and there is very little color distortion, in order to be more robust to light variations. Once again, there are also updates in the neighbouring pixels, where a randomly chosen neighbour of pixel x is updated, replacing one of its samples by the current observation at x. To characterize the dynamics of a background, a value D_{min} is computed using the minimal distances between the features and if the persistence sum is lower than the persistence threshold, the distance is increased. Then a learning rate α is used to average the previous D_{min} with the new calculated distance. There is also another dynamic indicator, $v(x)$, that captures the noise from blinking pixels (pixels classified differently in consecutive frames with the object borders removed) in dynamic backgrounds, working as an accumulator for each pixel, increasing its counter by 1 for each blinking pixel and decreasing it by 0.1 otherwise. Then, both D_{min} and $v(x)$ are used to update the learning threshold $T(x)$, increasing it when the new pixel value is classified as foreground (meaning less frequent updates, as the update probability is given by $p = 1/T(x)$) and decreasing it otherwise. The distance threshold $R(x)$ is also susceptible to updates, depending on if its previous value is lower than an exponential D_{min}. The persistence threshold is computed as

$$W(x) = \frac{q(\omega_0, t)}{R(x) \cdot 2} \tag{3.11}$$

where $q(\omega_0, t)$ is the first word of the local dictionary, taking into account that it is sorted by persistence value.

3.2.1.H SubSENSE

SubSENSE [32], by the same authors of [31], has the same principles of [31] regarding thresholds and their dynamic updates but instead of codewords for pixel representation, each pixel is defined by pairs of color value and LBSP features. To classify a sample as background, first a color comparison is made and if the color is similar, the LBSP features are generated and compared, so it's only considered background when there is a match in both descriptors. This approach got the best F1-score of the 2014 CDNet edition.

3.2.1.I GSOC

GSOC is an algorithm that has no paper associated with it but is included in the OpenCV contrib package and was found to have good performances in our environments. The algorithm was developed during the Google Summer of Code, hence the name GSOC. This approach has some of the characteristics of previously reviewed work. For each pixel, some color samples are stored. Each sample stores a color, number of hits (matches) and the last time there was a hit. When a new pixel value is retrieved from the new frame, it is compared against the color of the stored samples. If the color is too different from all the samples, the pixel is considered foreground but there is still a chance that the oldest background sample is replaced by a new sample created for this new pixel value. If not, then the closest sample is updated to include the new latest matching time and there is a possibility that the match is propagated to the neighbours.

3.2.2 Background Subtraction with Deep Learning

There are also some works on background subtraction using neural networks. More recently, the study of Convolutional Neural Networks (CNN) applied to background subtraction has gained relevance. Instead of the low-level or hand-crafted features used so far, like the values of the color or the binary similarity patterns, this technique learns spatial features automatically through training.

3.2.2.A Deep background subtraction with scene-specific CNN

One of the first approaches was [6]. The algorithm is separated into several parts. On the first part, a background image is estimated by converting the first 150 frames to grayscale and applying temporal median over each pixel. Then, a scene-specific dataset is generated using samples of pairs of input and background patches and calculating their target values through other background subtraction algorithms or manual labelling. After that, the network is trained. It is composed of two feature stages (convolutional layer plus a max-pooling layer) and a two-layer Multi Layer Perceptron (MLP) (every neuron in the first layer is connected to every neuron on the second layer). The output layer is a sigmoid unit. The system was evaluated on the CDNet 2014 dataset and outperformed the other approaches in terms of overall F1-score using the ground truth labels as the target values of the training samples. One drawback of this approach is that the network has to be trained for each new scenario.

3.2.2.B Deep CNN for video sequence background subtraction

The authors of [1] also propose an algorithm using CNNs that can be applied to different scenarios without retraining. They extract background samples using SubSENSE algorithm [32] (reviewed previously), to make an array of colors corresponding to the background for each pixel. This array is fixed in length

and old samples are replaced by the more recent ones. To build the background image, an average of the values is computed, but the number of samples bm considered in the average isn't fixed to prevent fast corruption of the model. In order to know how many samples are to be considered, they use the Flux Tensor [34] algorithm, that acts as a motion detector, computing the temporal variations of the optical flow for each pixel. It outputs a binary image containing the pixels that are considered to be "moving". After that, the percentage of moving pixels F_s is computed, and it's used to increase or decrease bm according to

$$bm = \begin{cases} 5 & \text{if } F_s \geq 0.25 \\ 5 + \frac{F_s - 0.02}{0.25 - 0.02} & \text{if } 0.02 < F_s < 0.25 \\ 90 & \text{if } F_s \leq 0.02 \end{cases} \tag{3.12}$$

This way, if the background is static, it's possible to use more samples that are a good representation of the background. The network is composed of three convolutional layers and a two-layer MLP. To train the network, they selected some samples that don't have significant background changes and extracted the background using the proposed approach. A median filter is applied to the output of the network to remove some noise and then each pixel value of the output is compared to a threshold to decide if the pixel is foreground or background.

3.2.2.C FgSegNet

This convolutional neural network presented in [20] uses an encoder-decoder approach to segment the foreground. This is an improved version of a previous network developed by the same authors. In the encoder, the first four blocks of a pre-trained VGG-16 net are used. The encoder outputs feature maps F that are then fed into a feature pooling module (FPM) before going into the decoder. In the FPM, the features F go through multiple dilated convolutions with increasing dilation rates, where the resulting features of each convolution are concatenated with the original features F before applying the next convolution. Dilation rates allow capturing a larger area of the input, where the bigger rates mean larger spacing between samples used in the convolution. A new feature map F' is obtained by concatenating all the resulting multi-scale features. At the end, F' is used as input to the decoder that outputs a probability for each pixel, which is then thresholded to obtain the foreground mask. One interesting aspect of this approach is that the authors were able to train the network with a small number of frames. The network achieved very high F1-scores in different datasets while only using 25 or 200 frames for training.

3.2.3 Deep Learning Based Detectors

Besides background subtraction techniques, there are also the CNNs that can locate objects and classify them. We review it as it can be used as alternative to background subtraction.

3.2.3.A YoloV3

YoloV3 [27] is a CNN that is trained to predict object classes and bounding boxes in an image. It is an improved version from the original YoloV1 network [25] and also YoloV2 [26]. The way YoloV1 approaches object detection and classification by dividing the image in a $S \times S$ grid. Each cell is responsible for detecting the objects whose centers fall inside them. Then, B bounding boxes with four components (x, y, w, h) and a confidence score each are predicted per cell. The first two components (center coordinates of the bounding box) are relative to the grid cell and the other two (width and height) are relative to the entire picture. The score is given by $P(\text{Object}) * \text{IOU}_{pred}^{truth}$, where $P(\text{Object})$ indicates if there is an object present and $\text{IOU}_{pred}^{truth}$ (which is an overlapping measure between the ground truth bounding box and the predicted one) tells us how well it fits that object. If no object belongs to that cell, the confidence score for those bounding boxes should be zero. Else, the score should be the Intersection Over Union (IOU) between the two bounding boxes. Additionally, each cell predicts C class probabilities, $P(\text{Class}_i|\text{Object})$, that represent the probability of the cell belonging to a certain class if there is an object in the cell. By multiplying the confidence score with the class probability, we obtain a prediction of a bounding box for an object belonging to a certain class. The network is composed of twenty-four convolutional layers and two fully connected layers. The authors tested the network on the PASCAL VOC dataset, with a 7×7 grid, 2 bounding boxes per grid cell, 20 classes so the output of the network was a $7 \times 7 \times (2 * 5 + 20)$ tensor. Although it was not as accurate as some others state-of-the-art detectors, it was very fast, being able to run in real time and was more than twice as accurate as the other real time detectors.

A new YoloV2 [26] was developed to improve the original approach, regarding localization errors and recall. The changes include the use of batch normalization, a higher resolution classification training for the last epochs, multi-scale training and a different model network called Darknet-19 with 19 convolutional layers. Another important difference from the previous approach is the use of anchor boxes (or priors) to predict the bounding boxes, inspired by the anchor boxes in [28]. Instead of predicting the coordinates for the bounding boxes, an offset to the anchor boxes is predicted. Given that the anchor boxes are first predefined, instead of generating random boxes, the authors used k-means clustering on the ground truth bounding boxes of their dataset in order to find the most appropriate priors to start with. This way, the learning process is easier at first than starting with random anchors. The position of the boxes are again relative to the grid cell's location. The class prediction is made per anchor box instead of per grid cell like it was in the previous approach. A good trade-off between the number of anchor boxes and the complexity, obtained through the k-means clustering, was five anchors and the authors changed the grid size to be 13×13. So, the output is a $13 \times 13 \times 5 * (5 + 20)$ tensor.

The Yolov3 allows for multi-label predictions for each box as some labels are not mutually exclusive and also makes predictions at three different scales, which means there are priors for each scale. This

time, three priors were used per cell in a total of nine across all scales. Additionally, a new model (Darknet-53) is used.

3.3 Tracking

Tracking algorithms can be split into two main groups: single target trackers and multiple object trackers.

3.3.1 KCF

One approach to track a single target is by using correlation filters like the Kernelized Correlation Filter (KCF) proposed in [12]. Using the properties of the Fourier Domain, circulant matrices and Gaussian kernels, it is possible to compute a very fast correlation filter to identify the target in the following frame. This is done by training a target regression using the sample from the current frame and in the following frame trying to find the location that better matches the filter created. Once the area with the maximum response is found, a new training is done in order to keep the target model updated. By using cyclic shifts and circulant matrices, it is possible to do dense sampling instead of the traditional random sampling but in a very computationally inexpensive way. This filter is not robust to size variations. The KCF tracker can be used with different image descriptors like the raw pixel values or Histogram of Oriented Gradients (HOG). The HOG descriptor captures the information regarding changes in color intensities and their orientations by computing a histogram of the direction of the gradients using the magnitude value for each cell according to a subdivision of the image. Although we need to compute the HOG before using them in KCF, according to the authors not only did it achieve a much higher precision than using raw pixel values but also performed at higher frames per second. The higher FPS are explained by having a single HOG descriptor for each 2×2 cell, reducing the computations regarding the Discrete Fourier Transform.

To deal with the scale variations, a scale-adaptive KCF was introduced in [19]. A scaling pool $S = \{t_1, ..., t_k\}$ with different scale variations is used and multiple patches of different sizes of the current frame are extracted according to each element in S. As the KCF needs all windows to have the same size due to dot-products being used, a bilinear-interpolation is performed to make all images the same size as the original one. Then, the same steps of the regular KCF are used to find the best matching window.

3.3.2 Hungarian algorithm with Kalman Filter

One typical approach in multiple target tracking is through data association using methods like the Hungarian Algorithm [18]. Considering that we have a set T of tracks from the previous frame, and a set d of

detected objects using some kind of object detector, one can try to associate each track T_i to a detected object d_j. To do this, we have to use a similarity measure to compare each object from the current frame to the ones from the existing track set of the previous frame. After that, the Hungarian algorithm is used to maximize the similarity value for each association between the tracks and the objects. One possible feature to use in the similarity metric is based on the position of the target, by using its last known position or alternatively an estimation of the position of where the target is believed to be according to past movement. This is the approach used in [3].

After the objects are detected, a Kalman filter [15] is applied to predict the position of the previous targets in the current frame. Kalman filter is an estimator that is commonly used to predict future positions of the target in a linear movement. The estimated state $\mathbf{x}_k$ and the observation or measurement $\mathbf{y}_k$ are given by:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{3.13}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{3.14}$$

where $\mathbf{F}$ is the state-transition model, $\mathbf{B}$ and $\mathbf{u}$ are the control-input model and control vector respectively, $\mathbf{H}$ is the observation model, $\mathbf{w}$ and $\mathbf{v}$ are both Gaussian noise samples. Then, the estimate covariance matrix $\mathbf{P}$ is computed. In the first step, both estimated $\mathbf{x}$ and $\mathbf{P}$ are computed and, in the next step, they are both updated combining their a priori estimations with the observations. For each object, the position and velocity (vertically and horizontally) are used to make a prediction and then, after associating the tracks with the new objects the measured position is used to update the model. For each existing target, a bounding box is computed using the predicted position, and Intersection-over-Union is computed as a metric of similarity between the bounding boxes of the detected objects and the bounding boxes of the current targets to afterwards apply the Hungarian algorithm. Regarding the maintenance of the tracks, a new one is created when no new object matches the newly predicted bounding boxes with the velocity set to zero and an existing one is deleted after consecutive fails in matching.

3.3.3 Interactive Multiple Model

The Kalman filter performs well in a linear motion scenario, but in maneuvering targets others approaches may give better estimations. In [36], an Interactive Multiple Model (IMM) is used which combines more than one model to cover different types of trajectories and conditions. A particle filter is also used to estimate the target's state. To cover every type of possible motion patterns, a lot of models would have to be used but to make it computationally viable usually only a few models are adopted. They include a constant velocity model (CV) for the cases of non-maneuvering motion, constant acceleration model (CA) and constant turn model (CT) for maneuvering motion. In that work, a self-adaptive CT model is used and to compute the current turn angular rate ω the previous speed estimations are

taken into account. In the first step of the IMM, the input interaction step, the probabilities of each model $\mu_i(k)$ based on the output of last iteration and the initial state vector for each model are computed. In the filtering step, the state estimation $\hat{x}_i(k)$ is computed using the initial state vector from the previous step. After that, the probability of each model is updated and in the last step an output is calculated as

$$x(k) = \sum_{i=1}^{M} \mu_i(k)\hat{x}_i(k) \tag{3.15}$$

where M is the number of models being used. As the authors of [36] used a Particle Filter, an additional step is required at the beginning, to initiate the particles and the second step from the traditional IMM is separated into two. First, a prediction is made using the particles and then updated using the measurement value. Second, the particles with smaller weights are ignored and resampling is done. Then, the state estimation is the arithmetic mean of the values of the particles.

3.3.4 Tracking crowds

Trying to solve the tracking problem in extremely crowded scenarios, the authors of [8] approached it using Binary Quadratic Programming. For the targets being tracked, several candidate locations are sampled and the best locations are chosen through the objective function:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} f(\mathbf{x}) = & \overbrace{c_a^T\mathbf{x}}^{\text{appearance}} + \overbrace{\zeta c_m^T\mathbf{x}}^{\text{target motion}} + \overbrace{\eta c_{nm}^T\mathbf{x}}^{\text{neighbour motion}} \\ & + \underbrace{\mathbf{x}^T C_{sp}\mathbf{x}}_{\text{spatial proximity}} + \underbrace{\mathbf{x}^T C_g\mathbf{x}}_{\text{grouping}} \end{aligned} \tag{3.16}$$

where x is a matrix encoding the possible combinations between the targets and the candidate locations. Constraints are also considered to make sure that every value is either zero or one and that a candidate location can only be associated with one target. The first part of the function corresponds to the appearance of the target. The same discriminative approach of KCF or another alternative algorithm based on Normalized Cross Correlation are used to model the appearance and compare the targets with the candidate locations, where c_a is the affinity vector encoding the appearance cost. Next comes the target motion where the authors use a Kalman filter model to predict the targets locations in the following frames computing the c_m vector. The third component tries to model the motion of the neighbouring targets, clustering the targets into different groups This is one of the distinctive parts of the approach to solve the tracking in high-density crowd scenes. The fourth component is a spatial proximity soft constraint which tries to discourage the tracker from selecting locations that are too close. The last part is a grouping constraint, that is again one attempt at solving tracking in very crowded scenes, where groups of people with similar movement are formed since people in groups tend to maintain their formations.

The authors evaluated the approach on nine sequences of high-density crowd using multiple variations of the objective function starting with a baseline as the NCC tracker and adding the other components incrementally and in the end replacing the NCC tracker with the KCF. In seven out of nine, the best performance was obtained with the five components using the KCF algorithm and in the other two the best one only used the motion term and the spatial proximity term on top of the baseline. In these two sequences the motion of the targets is more heterogeneous.

4

Implementation

Contents

4.1 Overall Architecture 35
4.2 Color equalization 35
4.3 Object Detection 37
4.4 Tracking 39

In this chapter, we describe the implementation of the system by first giving an overview of the architecture and then a more detailed description of each of the components.

4.1 Overall Architecture

The architecture of the system is shown in 4.1. A tracking system can be developed using automatic initialization of the objects or a manual one. In our case, we want the system to track specific fish, so the first procedure is the manual initialization of the tracks for the fish that the user wants to track, allowing selection of multiple targets. The initialization is made by selecting the bounding boxes around the targets, which are saved together with the corresponding frame numbers. After this, the tracking procedure can start.

Our approach consists of three main components working in a pipeline: color equalization, object detection and tracking. A frame F^t is retrieved from a video at time t, and first goes through the color equalization module to possibly be altered in an attempt to improve the colors displayed. This module outputs the processed frame F'^t that is used in the next module of object detection.

In the object detection step, we try to identify where the objects are present in the frame. The objects are detected either through background subtraction or deep learning. With background subtraction, we are able to retrieve both the mask (pixels inside the shape corresponding to the object) and the bounding box for the objects, while with deep learning, we can only get the bounding boxes. Either way, this module outputs a list of detected objects to be used later.

The final module is related to the tracking step. Here, we use the list of detected objects as well as a list of currently tracked objects and try to pair a detected object with a track. Since both the detections and the tracks may not always get a match with one another, we need a fourth component, the track manager, that is responsible for the creation and deletion of tracks. The track manager also updates the tracks in case of a successful match between an object and a track.

In the following sections, each component is described in more detail.

4.2 Color equalization

This module is the first of the pipeline but it is optional and depends on the environment displayed in the video as some videos do not present some of the problems. The colors captured in the coral reef tank video are clear enough as everything in that tank is closer to the camera and is well illuminated, so no color equalization is needed for that video. The idea behind this module was to bring the color displayed by the video of the main tank closer to what a human would see if it would be above the water,so it would be easier to later identify the fish. As this environment is not close to the surface, the lighting is poor

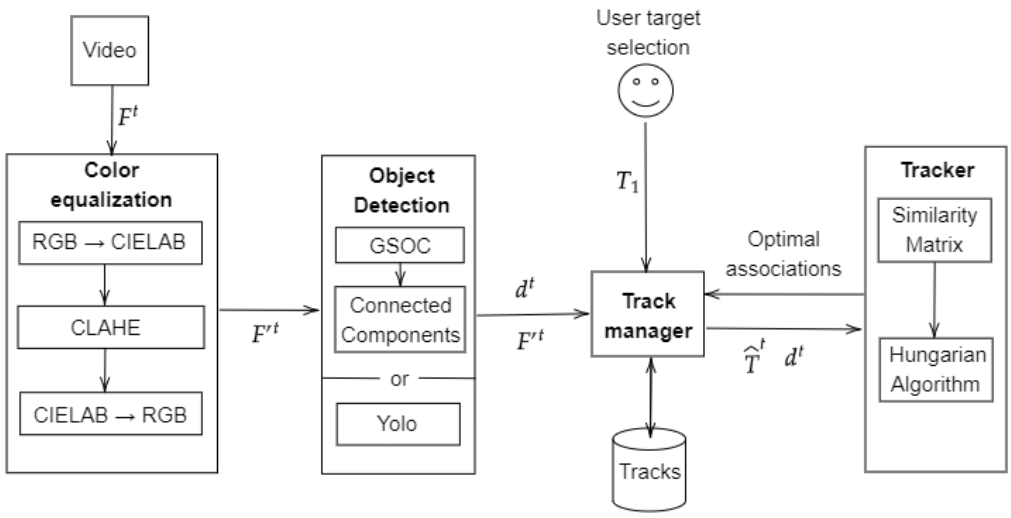


Figure 4.1: System architecture. The user first selects the targets of interest initializing the tracks. Then, each frame F at time t may be applied the color equalization resulting in F'^t. The frame is then used in the object detection module that can use two different techniques producing a list of detected objects d^t. In the end, the tracker uses the active tracks $\hat{T}^t$ and tries to match them with each detection in d^t.

and makes it harder to detect some of the fish that are far away from the camera as well as the ones swimming in areas where their colors are similar to those of the background.

The type of transformation chosen to try to improve the frames was the one reviewed in the related work using Contrast Limited Adaptive Histogram Equalization [17]. In that work, the authors also tried to improve the contrast of underwater images and it seemed suitable to apply in the video sequences of our main tank.

To do this, we convert the frame from its original color space RGB to CIELAB. Next, we apply the CLAHE operation included in the OpenCV library in the L channel, that represents the lightness of the color. There are two parameters that we can change in this transformation. The first one is the grid size. This indicates in how many tiles we are going to divide the image into, so that we can apply the equalization in each tile individually. The next parameter is the clip limit, that is the threshold used to clip the bins that are above it, distributing that part of the histogram across all bins equally. Following the work of [17], we used a 16×16 grid with a clip limit of 2. After the transformation, we convert the frame back to RGB to be used in the following modules.

An example of a transformation obtained using this approach in the frames of our main tank is shown in 4.2.

Figure 4.2: Result of applying the color equalization technique to a frame from the video of the main tank. Left: original; right: after equalization.

4.3 Object Detection

This module is responsible for identifying and locating fish in a given frame. We included two different types of approaches in the detection process that can be used alternatively. The first one is through the more traditional method of background subtraction where the objects are obtained using a segmentation of the predicted foreground pixels. Each pixel is classified individually and then pixels that are close together form a blob that is considered to be a fish detection. The algorithms are not totally accurate so there will be blobs detected that only contain pixels belonging to background elements and there will also be fish whose pixels will be classified as background so they will not have a corresponding blob. There is also the case of when a fish segmentation is fragmented in multiple blobs instead of a single one. There are many background subtraction algorithms that could be used, so a comparative study was performed on frames of our videos using different algorithms. This study, that is presented in the chapters ahead, showed that the best performing algorithm for our environments was the GSOC, therefore it was the one chosen to be used in our system.

The alternative to background subtraction is an object detector using a Convolutional Neural Network that is trained to locate the fish. We use the YoloV3 net that is one of the best object detectors currently and has showed good results in other domains. Once again, false positive detections as well as false negative ones can be generated using this object detector.

4.3.1 GSOC

The implementation of this algorithm is available in OpenCV contrib repository[1], and it has no paper associated to it. Being an implementation of a background subtraction algorithm, it uses a background model that is built and updated over time. This model is then compared with new frames from the video to decide for each pixel if it is in the background or in the foreground. In this particular approach, the

[1] https://github.com/opencv/opencv_contrib

model is based on color samples, meaning that for each pixel, a buffer of color samples is stored. The sample of this model has three important elements: the color intensity for three channels of RGB, a reference to the last frame this color was seen on in that pixel location and how many times it was seen in the past, also referred to as a hit.

The first step of GSOC is to initialize the background model, as we need some samples to compare the first frames with. The initialization, in this case, takes the first frame and creates, for each pixel position, multiple samples using the color of the corresponding pixel in that first frame and both the reference to last frame and the number of hits are set to zero. The number of background samples stored can be defined by the user.

Once the model is initialized, we can start the pixel classification process as background/foreground. First, we compare each pixel color with the colors from each background sample to determine what is the closest one. This comparison is made using the squared Euclidean Distance, that computes the distance D between two colors $c1$ and $c2$ as

$$D(c1, c2) = (c1_R - c2_R)^2 + (c1_G - c2_G)^2 + (c1_B - c2_B)^2 \tag{4.1}$$

Then, we compute the color threshold c_{thr} to decide if it is background or foreground that is given by

$$c_{thr} = \alpha * M^t + \beta \tag{4.2}$$

where α and β are given by the author as 0.01 and 0.0022, respectively, and M^t, which is used as an automatic manager of the threshold value and is initialized at 0.005, is given by

$$M^t = \delta M^{t-1} + (1 - \delta)D^t \tag{4.3}$$

using δ equal to 0.1. So, if the minimum distance between the new pixel color and all the background samples stored for that position is higher than c_{thr}, the pixel is classified as foreground. But even if it is classified as foreground, there is still a chance that this color is used to create a new background, with a probability equal to $P_{replace}$ defined by the user. When a new sample is created, it replaces the oldest one (the one that has not been seen the longest). If the minimum distance is lower than c_{thr}, then it means that it is similar to a color of an existing background sample, so that position is classified as background. In that case, the color of the sample that was the most similar to the new pixel value is updated (with an average between the color stored and the new color) as well as the timestamp of the last hit and the number of hits. On top of that, there is also the possibility that this sample that got a match with the new color is going to replace one of the samples from the neighbours sample set. For this to happen, there is a probability given by $P_{propagation}$ and the number of hits of the sample has to

be higher than the hits threshold h_{thr}. There is also a blinking detection mechanism, that can replace samples, that is influenced by the changes of classification (between background and foreground) for two consecutive frames.

The last step of the algorithm is to remove the noise in the segmentation. A connected components algorithm is applied to label the pixels into groups, where pixels from the segmentation mask that are connected get the same label. Then we change the classification of the pixels belonging to groups of very small areas. This is applied to both a foreground mask and background mask, in order to remove the very small patches of foreground and background. Then, a Gaussian Blur is applied to the final segmentation mask, and the pixels with values higher than 127 are classified as foreground, and as background otherwise. After getting a final segmentation mask with each pixel classified, we apply again the connected components algorithm to extract a list of foreground objects that is used as fish detections d for the tracking part.

4.3.2 YoloV3

The use of an object detector like YoloV3 results in a different working flow and a different type of output. Instead of having a background model that is initialized and then continuously updated as the frames are processed, we have a network that is trained beforehand using ground truth examples of real targets (fish in our case). The objective is for the network to learn patterns automatically so that it can later identify the regions of the images that correspond to fish. The network used in this work had already been trained to detect a single class of objects: fish. There is no distinction between different species or groups/families. More information on the network and training can be found in chapter 5.

Once the training is done, we can start running the frames through the network and getting its predictions. Instead of doing the prediction step while running the system, as it was a bit slow and would take too much time to evaluate the experiments, the bounding box predictions for every frame of the video are made previously and saved. So when we get to the detection module of our main system, we search for the bounding boxes that were predicted for the corresponding frame. When we get those boxes, we can create the list of detected objects by cropping them from the original frames using the coordinates of the predictions. This is where the output can have some differences between approaches: using GSOC the module outputs a list of blobs that can have multiple shapes and using Yolo the outputs are always rectangular.

4.4 Tracking

The tracking is the last step of our system. It is here that we try to understand which of the new detections corresponds to each of the existing tracks. Our approach is a multi-target approach instead of a single-

target one, as we create tracks for every new fish detected in the video instead of only creating a track for the selected fish. This is important as we approach the tracking as a problem of data association. Having tracks for every fish can help us prevent incorrect associations, as we have to maximize the association of all tracks.

The first important element in this module is the track. A track T_k corresponds to a trajectory of a detected fish. In this case, a trajectory is the temporally ordered list of frame coordinates of where the fish has been detected and the corresponding frame numbers of when those detections occurred. We want to allow tracking fails for a certain amount of frames before terminating the track instead of terminating as soon as the tracking fails, as we need to account for the detection fails. So, each track will not have a corresponding detection in every frame and will have some jumps along the trajectory. In addition to the position and frame number, there are other elements stored that are important to tracking. The first one is the track identifier. Each track has a different ID so we can tell them apart and understand if the associations are being successful and consistent over time. We also store an image of the target and use it to build an RGB histogram that is used in the association step. Then we have a variable indicating if the track was initialized by the user or not so we can end the tracking when there are no more active tracks $\hat{T}$ selected by the user. An active track has an indicator to show that it is still being used for the tracking. There is also a counter for the number of consecutive frames in which the track did not match with any detected object, that will be used to turn active tracks into inactive ones.

The next component in this module is the track manager. It is responsible for adding new tracks and update them over time as well as return the valid tracks to be used in the tracking for the current frame. When there is a new detection that was not associated in an existing track, a new track has to be created by the track manager. Every time there is a new frame to be tracked, the track manager looks up in the list of all tracks, the ones that are still active. After the association between detected objects and existing tracks, the update step occurs. Every track with a successful match is updated to include the new bounding box, build a new histogram for the new image and set the counter of consecutive fails to 0. For every detection that that did not match with an existing track, a new track is initialized. For every existing track without a match, the consecutive fails counter is increased. If the counter becomes higher than the maximum allowed, the track becomes inactive. The threshold for the maximum number of consecutive fails was set to 48 frames.

Once we have the list of detected objects and the list of active tracks $\hat{T}$, we can progress on the tracking procedure. In order to perform the data association for the two sets, we need first to compute the similarities between each element of one set with the elements of the other, obtaining a similarity matrix. There are two components used to compute the similarity S between a active track $\hat{T}_k$ and a detection d_j, color and position, and is given by

$$S(\hat{T}_k, d_j) = S_c(\hat{T}_k, d_j) + S_p(\hat{T}_k, d_j) \tag{4.4}$$

where S_c is the color similarity and S_p is the position similarity. To compute the color similarity, we need to use the RGB histogram H stored in the track and build one for the detection. Then we compute the Hellinger distance D_H between the two, that returns a value between 0 and 1, where lower values indicate more similarity than higher ones. This distance is given by

$$D_H(H_{\hat{T}_k}, H_{d_j}) = \sqrt{1 - \frac{1}{\sqrt{H_{\hat{T}_k} H_{d_j}}} \sum_I H_{\hat{T}_k}(I) * H_{d_j}(I)} \tag{4.5}$$

and we use this value to compute S_c

$$S_c(\hat{T}_k, d_j) = 1 - D_H(H_{\hat{T}_k}, H_{d_j}) \tag{4.6}$$

The position similarity is computed as

$$S_p(\hat{T}_k, d_j) = 1 - \frac{\sqrt{(\hat{T}_{k_x} - d_{j_x})^2 + (\hat{T}_{k_y} - d_{j_y})^2}}{D_{max}} \tag{4.7}$$

where D_{max} is equal to 866, which is the approximate maximum possible Euclidean distance between two pixels in a 720×480 frame. Having a similarity matrix, we convert it to a cost matrix and apply the Hungarian algorithm, that optimizes the associations between all tracks and detections. Then, there is a final step where we verify if the associations should be allowed or not. So, for each pair of detection and track, we verify if the color or the distance is not too different, and if it is, we discard that pair. The color similarity should be higher than the color threshold c_{thr} and the distance should be smaller than distance thresholds d_{thr}. This step is needed because, even though the Hungarian algorithm returns the best matches, it does not mean that they are all matched with high similarity. One simple example of when this occurs is when there is a single fish on scene being tracked and there is one frame where that fish is not detected but another one entered the scene and was detected. If there is no verification, the track will be matched with the new fish even if it is on a completely different part of the frame or is visually different in terms of color.

This is what the baseline of our tracking approach consisted of. While testing the system and evaluating possible improvements based on the failures happening on the validation dataset, three new tracking features where added in order to try to solve those issues.

4.4.1 Color history

The first feature implemented was motivated by the fact that when fish are close together or even in front of each other, the detection mask, or bounding box, will contain features from both fish. If the other fish have different colors, the histograms will get corrupted and it may prevent the correct association

later on, after they are detected separately. To solve this, instead of only using the last image to create the histogram, we compute an average of histograms between the last histogram of the track and the histogram from the new detection allowing us to have some kind of color history. This average using γ as the weight of the new histogram is computed according to

$$H^t_{\hat{T}_k} = (1-\gamma) * H^{t-1}_{\hat{T}_k} + \gamma * H_{d_j} \tag{4.8}$$

4.4.2 Temporary tracks

Then, we noticed that there were times when the segmentation was fragmented, producing more than one bounding box for the same fish or additional bounding boxes were incorrectly being predicted in areas around the fish. In these cases, new tracks will be created for the extraneous detections. Thus, in subsequent frames it is likely that the original track would end up being associated with the wrong box, and another track would end up following the fish. The original track would keep being associated with an incorrect detection until it was terminated. To prevent this, we added a temporary track mechanism, as new tracks are not as reliable as older ones and may just be a miss-detection. When a track is created automatically (not by user selection), it is initialized as a temporary track. Next, we make a first application of the Hungarian algorithm only using the permanent tracks, and then remake the association step using the unmatched detections and the temporary tracks. This way the permanent tracks will have priority over the temporary ones. Once a temporary track gets 5 successful matches, it turns into a permanent track.

4.4.3 Movement prediction

The final feature was the addition of a movement prediction algorithm. There are times when the tracked fish are not detected because they go behind other fish or simply because the detection failed. So it is important that the track does not use the last known location for the position similarity, as the fish will eventually get too far away if there are many consecutive matching fails. This is where the Kalman Filter can be used in order to predict the movement of the fish based on the past positions and predictions.

For each frame, the position of each tracked fish is predicted. This value is then used in the position similarity computations instead of the position of the last bounding box of the track. Once the matching is done and is associated with a detection, the prediction is updated to include the position of the detection, commonly known as the update step of the Kalman Filter. If there is no associated detection for a track in a certain frame, there is no update step to the movement prediction, but the Kalman Filter will keep predicting new positions.

5

Evaluation

Contents

5.1 Datasets 45

5.2 Experiment A: object detection 47

5.3 Experiment B: tracking 50

In order to test and evaluate our approach, one video for each of the environments was used. Both videos were recorded at Oceanário de Lisboa using a stationary camera positioned outside the tanks, so there is a glass wall between the camera and the water. Each environment presented different challenges.

Each video, for which the corresponding environments have been described in chapter 1, is summarized in table 5.1.

Table 5.1: Description of the videos used for testing.

Environment	Description	Resolution	Total frames	FPS
Main tank	Large tank with medium to large fish with similar colors	720 × 680	7200	24
Coral reef tank	Small tank with small fish with different colors	720 × 680	7200	24

All the implementation, testing and evaluation was performed on a computer with the specifications shown in table 5.2.

Table 5.2: Specifications of the computer used in the development and testing of the tracking system.

CPU	Intel Core i7-8750H @ 2.20 GHz
Memory RAM	16 GB
GPU	NVIDIA GeForce GTX 1050 Ti

5.1 Datasets

Using those two videos mentioned, two different datasets were built.

The first one, described in Table 5.3, consists of pairs of frames and their corresponding foreground/background segmentation. The segmentation was made by hand for each frame individually using a segmentation tool in Matlab. For each video, eight frames were selected to be used in this dataset. Because this is a time consuming task, there had to be a compromise between the number of annotations and the time spent doing it. The selection included the last frame of the seconds 1, 10, 30, 60, 120, 180, 240 and 300. Since some of the background subtraction approaches included the training of a convolutional neural network, the dataset had to be extended so the same frames were not being used in both training and evaluation. Thus, we applied dataset augmentation techniques to the previously mentioned frames and corresponding manually labeled segmentations. The augmentation was obtained using the Pre-Processing module of Keras [7] and included a combination of the following transformations: shearing, width shifting, horizontal flipping, zooming and using reflection as a filling

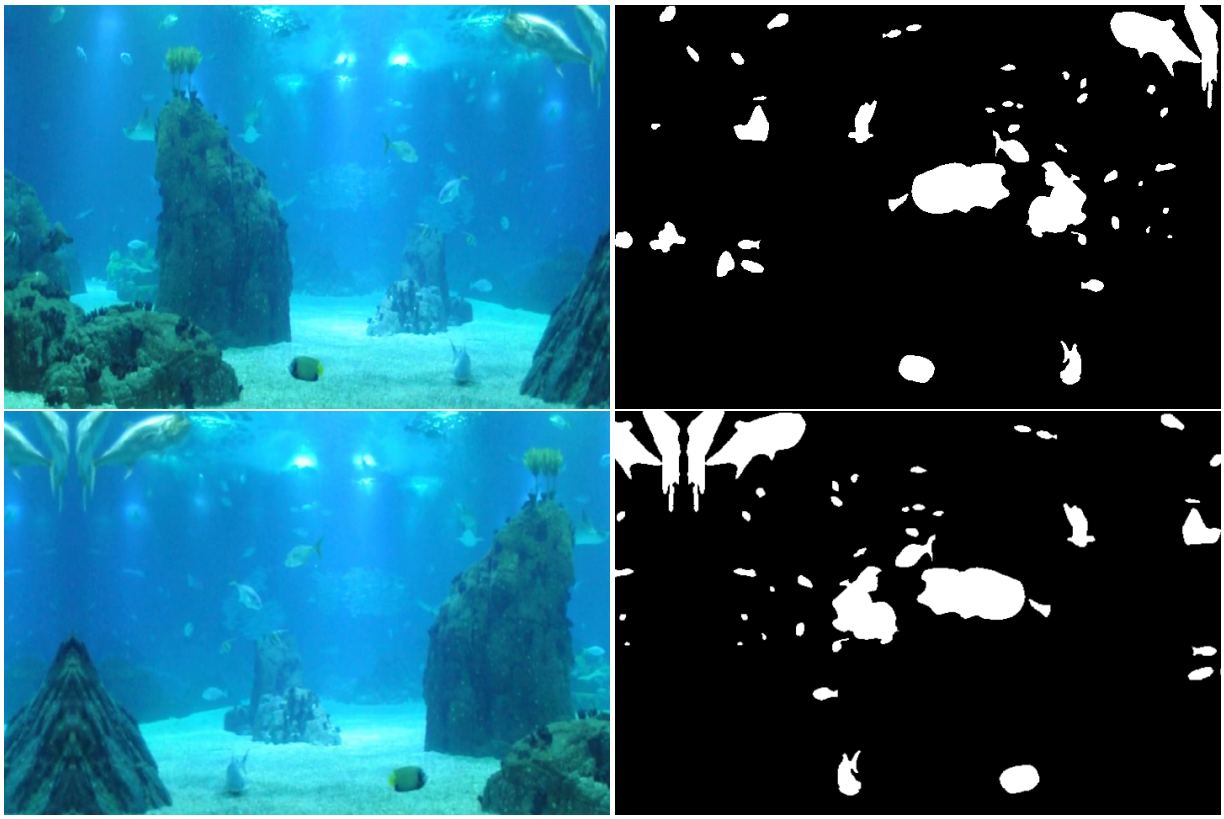


Figure 5.1: Top row: Original frame and corresponding segmentation; Bottom row: Frame and segmentation after augmentation using width shifting, horizontal flipping and zoom.

mode. An example of the data augmentation is illustrated in Figure 5.1. Only the augmented frames are used in the training process, leaving the original ones to the testing process so we can compare with the traditional approaches. There were 25 augmented frames produced for each video.

The second dataset, described in Table 5.4, was built for the tracking evaluation. After choosing a fish to track its trajectory, a bounding box was manually selected for each frame in which the corresponding fish appeared. For the frames that the fish is totally occluded, no bounding box was selected even if the position of the fish is known, as we only want the system to identify the fish when it is visible. This dataset was separated into two groups for each of the videos. The trajectories were randomly selected throughout the entire video with a selection of different types of trajectories and targets. The first group is the validation set containing trajectories of different fish with varying movements and swimming patterns to help tune the tracking module parameters and choose the features to be included. The other group is the testing set containing trajectories not present in the validation set, which will be used to perform the final evaluation of the tracking. The length of the 15 trajectories of the validation set for the main tank varied between 152 and 827 frames, and the 17 trajectories of the validation set for the coral reef tank varied between 58 and 1687 frames. Both testing sets were composed of 4 trajectories each, with the length of the trajectories varying between 192 and 385 frames.

Table 5.3: Background subtraction dataset

Environment	Description	Quantity	Usage
Main tank	Pairs of frames and segmentations	8	Testing
Coral reef tank	Pairs of frames and segmentations	8	Testing
Main tank	Pairs of frames and segmentations obtained through dataset augmentation	25	CNN training
Coral reef tank	Pairs of frames and segmentations obtained through dataset augmentation	25	CNN training

Table 5.4: Tracking dataset

Environment	Quantity	Shortest trajectory (frames)	Longest trajectory (frames)	Usage
Main tank	15	152	827	Validation
Coral reef tank	17	58	1687	Validation
Main tank	4	197	385	Testing
Coral reef tank	4	192	264	Testing

5.2 Experiment A: object detection

To evaluate the object detection, we separate the experiment in two parts: background subtraction and bounding box prediction.

5.2.1 Experiment A1: background subtraction

In the first experiment, we make a comparative study on background subtraction where we test several different approaches (listed in Table 5.6) to understand which one is more adequate to use in both of our environments. It is important that we have the best possible object detection as the following steps of the system's pipeline rely heavily on it and also affect their performance. Each environment is tested separately. For this study, we will run the videos and apply each of the techniques, one at a time. For each frame that we have a corresponding manually labeled segmentation, we will get the proposed segmentation by the current algorithm and compare both, pixel by pixel. Each pixel is a binary classification of either background or foreground. Then, we calculate the number of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) across those predefined frames for each algorithm. With these values we can compute Precision (Pr) and Recall (Re) as

$$Pr = \frac{TP}{TP + FP} \tag{5.1}$$

$$Re = \frac{TP}{TP + FN} \tag{5.2}$$

Using Precision and Recall, we can compute F_1-score as

$$F_1 = 2\frac{Pr * Re}{Pr + Re} \tag{5.3}$$

The higher F_1-score the better. On top of that, we also register the total time spent running the algorithms on the entire video sequences and take it into account when analysing the results. There are many algorithms being tested, each one with multiple parameters that can be changed depending on the scenario but testing every variant for each algorithm would take too much time. Instead, we test each algorithm using the default values provided for each parameter and in the end we will choose the one with the best performance and fine tune its parameters. The different variants are described in Table 5.5. First we test the single-pass variant, a simple version of the algorithms is applied to the video frames once and evaluate the results.

Table 5.5: The different variants of the background subtraction tests using the testing dataset and in which tanks they are used.

Variant	Description	Tanks
Single-pass	The base variant where each algorithm is applied to the video once.	Both
Two-pass	First run to build background model, and a second to evaluate.	Both
Color equalization	Verify if color equalization helps the algorithms.	Main tank

Then, a two-pass variant of these tests is included, that allows us to run through the videos twice. In the first run we will only initialize the background and update it over time and in the second run we will evaluate the results using the already created model. This is an attempt to verify if some background subtraction techniques can have better results at the start if a model is already initialized and updated, even though the last updates do not correspond to what was happening prior to the start of the video.

The test will also include a variant comparing the results between using the color equalization techniques and not using them in the video of the main tank, as it is in this one that the colors are a bit distorted due to being underwater in greater distances. So each frame will be applied color equalization and then the same procedure described earlier will take place. The equalization will be done using the CLAHE approach of [17]. First the frame will be transformed to the CIELAB color space and then CLAHE is applied with a clip limit of 2 and a grid sized 16×16 only on the L channel. Then, it is transformed back to RGB and is ready to be used as input of the object detection module.

The algorithms being tested are listed in Table 5.6. There were two libraries used that aggregated

Table 5.6: Background subtraction algorithms used in the comparative study

Name	Library	Training needed	Number
Lobster [30]	BGS	No	1
PAWCS [31]	BGS	No	2
SigmaDelta [21]	BGS	No	3
SubSense [32]	BGS	No	4
ViBE [2]	BGS	No	5
GSOC	OpenCV	No	6
KNN [38]	OpenCV	No	7
MOG2 [37]	OpenCv	No	8
FgSegNetv2 [20]	-	Yes	9

Table 5.7: FgSegNetv2 training time

Environment	Time	Epochs
Main tank	00:09:41	40
Main tank with CLAHE	00:09:00	37
Coral reef tank	00:10:19	43

some of the algorithms together. The first one is the OpenCV [5] which contains some background subtraction algorithms among other useful tools on computer vision. The second one is the BGSLibrary (BGS) [29]. Every algorithm was tested using its Python version.

To train the NN FgSegNetv2, we follow the same procedure as the authors [20]. The RMSProp optimizer is used with a batch size of 1, the learning rate is initialized at 1e-4 and is reduced by a factor of 10 after 5 epochs without validation loss improvements, a maximum of 100 epochs is set. It is applied early stoppage if the validation loss stops does not improve for 10 epochs. The binary cross entropy loss is also used, and as there is a big difference between the number of foreground pixels and background ones, making the classes imbalanced, we give more weight to the foreground and less to the background. As the output of the network is a value between 0 and 1, we tested multiple thresholds and chose the one that presented the best results for the training dataset. The training performance is outlined in Table 5.7, taking around 10 minutes and around 40 epochs for each of the three settings.

5.2.2 Experiment A2: bounding box prediction

To evaluate object detectors such as YOLO, a different approach has to be used and a direct comparison with the background subtraction algorithms is not possible. This object detector outputs bounding boxes

where fish are predicted to be, instead of classifying each pixel as foreground or background. So to evaluate this approach we will compare the outputs with manually labeled bounding boxes. To have some similarities with the previous part of the experiment, the same eight frames were considered. In these eight frames of each environment, the fish present in them were manually selected by drawing a bounding box around it but the selection varied between environments. In the coral reef tank, all fish were selected, even the smaller ones, as long as they were mostly uncovered/inside the field of view, being clearly recognizable. The fish in this scenario are easy to detect as there are not many and there is a reef behind them preventing them swimming away from the camera and disappearing. In the main tank, the selection was made carefully. As there are many fish and they can swim far away, only fish that were clearly identifiable and not part of a school were selected, and certain areas where the identification was difficult, like around the schools, too far away from the camera or indistinguishable by the human eye, were ignored to not punish detections in those areas. Even though some hard areas for detection were ignored, we tried to include in this ground truth set some challenging boxes too so we can have more meaningful results. So when comparing the results of the two tanks, one should take this into account. In terms of the evaluation, the predicted boxes are compared with the ground truth boxes in terms of overlapping using the Intersection Over Union (IOU) and classified according to the following definitions:

- True Positive (TP): IOU between a predicted box and a GT box is higher than 0.5
- False Positive (FP): IOU between a predicted box and every GT box is lower than 0.5
- False Negative (FN): IOU between a GT box and every predicted box is lower than 0.5

Then we can compute Precision and Recall using the equations [5.1] and [5.2].

The network used was already trained and is available online [1] as well as the training dataset used.

5.3 Experiment B: tracking

This experiment will be used to evaluate the tracking performance of our system. Once again, tests will be run for each environment separately. First, the validation dataset will be used and the tests will allow us to tune the parameters of the tracking module as well as understand the influence of each feature. Each test will consist of one run per trajectory and the tracker will be initialized using the first bounding box of the ground-truth trajectories that were manually labeled. The tracker will try to to match each detected object with a track. After many unsuccessful matches with the track that is being tested, the tracking will stop. The bounding boxes that were associated with that track will be used to compare with the ground-truth ones. We will run these tests on the validation dataset using two different

[1] https://github.com/rocapal/fish_detection

object detection techniques to tune the parameters. After that, we will use the testing dataset to directly compare both approaches in each environment and get our final results from there.

5.3.1 Evaluation Metrics

We will measure the performance using two types of evaluations. We start with a group of metrics that is commonly used in the literature. The first metric of this group is the Precision plot. To build this plot, we first calculate the Euclidean distance between the center of the predicted bounding box and the center of the GT bounding box. Then, we plot the percentage of frames that have a smaller distance to the ground truth than a threshold varying between 0 and 50. The next one is the Success plot. For a predicted bounding box, we compute its overlap ratio with a GT bounding box by dividing the intersection of the two areas by their union. Then, we plot the percentage of frames where the overlap ratio is higher than a certain threshold, varying between 0 and 1, being 1 complete overlap and 0 the boxes not touching each other at all.

With those two plots, we can analyse and compare between different versions of the system by looking at a global indicator for all bounding boxes of every track together but if we want to analyse the performance per track we will need another approach. To evaluate each track individually, we check if for each frame, there is a corresponding predicted bounding box for that trajectory. If there is, we compute the overlap ratio and we classify it as a match or no match depending on if the overlap is higher than a threshold or not. When there is no predicted bounding box for that frame it is classified as no match. We then compute the percentage of frames per track where there was a matching prediction from our system and can build a plot to evaluate the evolution of the performance for the tracks.

5.3.2 Algorithm Variants and Parameter Tuning

Table 5.8: The different variants of the algorithm to be tested using the validation dataset for each tank. In each variant, the corresponding parameters are tuned.

Variant	Description
Baseline	Hungarian algorithm using color and position
1	Addition of the color history feature
2	Addition of the temporary tracks feature
3	Addition of the movement prediction feature

In the first round of tests, where we will try to tune the parameters of the system, each variant that is tested (listed in Table 5.8) includes all the previous variants of features and tuned parameters. The first test is the base variant with the tuning of the color and distance similarity thresholds. These are the

thresholds used when matching detected objects with the existing tracks. If the distance is higher than the distance threshold or the histograms have a bigger difference than the color thresholds, the match between the two is ignored. So, first we test the color threshold and maintain the same value for the distance threshold. We determine what is the best value and do the same procedure for the distance threshold, varying it while keeping a fixed color threshold.

Then, using the best color and distance thresholds, we add the color history by using histogram averaging. We test multiple weight combinations between the old histograms and the new, and select the one with the best results.

The next iteration of tests is about adding the temporary tracks feature. Until now, every track had the same priority as the other ones during its entire life span. With the temporary tracks, we check if giving less priority to newly created tracks can help decrease matching errors between tracks and detected objects.

The following variant is the inclusion of the movement prediction using a Kalman Filter. The parameters of the Kalman Filter were tuned beforehand for each environment independently. To do this, we used the ground truth trajectories of the validation dataset that were made by hand, as well as the detection results using the detection module that we fine-tuned in the experiment A. For each frame, given the predicted bounding boxes, we selected the bounding box that corresponded to each track. The first parameter calculated was the initial predicted estimate covariance matrix **P**. As the initial position is given by the user, we considered the position estimate covariance to be zero. We computed the distance between the center of every two consecutive bounding boxes from the ground truth trajectories, GT_k and GT_{k-1} and the number of frames separating those two bounding boxes Δt, resulting in a velocity in terms of pixels per frame, according to

$$v = \frac{GT_k - GT_{k-1}}{\Delta t} \tag{5.4}$$

Then, the velocity covariance is computed. Next, we computed the measurement noise covariance matrix **R**. We compared the distance between the center of each GT bounding box and the corresponding predicted bounding box that we had selected manually. This way we could estimate how noisy the measurements were. This measurement error m_e is given according to

$$m_e = m^t - GT^t \tag{5.5}$$

where m^t is the selected detection (measurement), and then the measurement noise covariance matrix is computed. After computing both covariance matrices **P** and **R**, we could now compute the process noise by comparing the state estimate before update step, $\hat{x}_{t|t-1}$, and after the update step, $\hat{x}_{t|t}$, using

Table 5.9: Kalman tuning

Parameter	Coral Tank	Main tank
P	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 9.2133 & 1.4947 \\ 0 & 0 & 1.4947 & 6.8746 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 6.7677 & 0.7922 \\ 0 & 0 & 0.7922 & 3.1849 \end{bmatrix}$
R	$\begin{bmatrix} 13.8895 & 1.1084 \\ 1.1084 & 13.8065 \end{bmatrix}$	$\begin{bmatrix} 359.3122 & -1.8805 \\ -1.8805 & 173.7857 \end{bmatrix}$
Q	$\begin{bmatrix} 1.3662 & 0.3048 & 0.1299 & 0.0245 \\ 0.3048 & 0.9799 & 0.0262 & 0.0851 \\ 0.1299 & 0.0262 & 0.0169 & 0.0023 \\ 0.0245 & 0.0851 & 0.0023 & 0.0103 \end{bmatrix}$	$\begin{bmatrix} 8.7114 & -0.2678 & 0.0755 & -0.0030 \\ -0.2678 & 0.1212 & -0.0029 & 0.0063 \\ 0.0755 & -0.0029 & 0.0014 & -0.0001 \\ -0.0030 & 0.0063 & -0.0001 & 0.0007 \end{bmatrix}$

the ground truth as measure, according to

$$p_n = \hat{x}_{t|t-1} - \hat{x}_{t|t} \tag{5.6}$$

and then the process noise covariance matrix **Q** is computed. We repeated this last matrix computation multiple times to stabilise the process noise value. In Table 5.9, the resulting matrices are presented.

6

Results

Contents

6.1 Results of experiment A 57

6.2 Results of experiment B 64

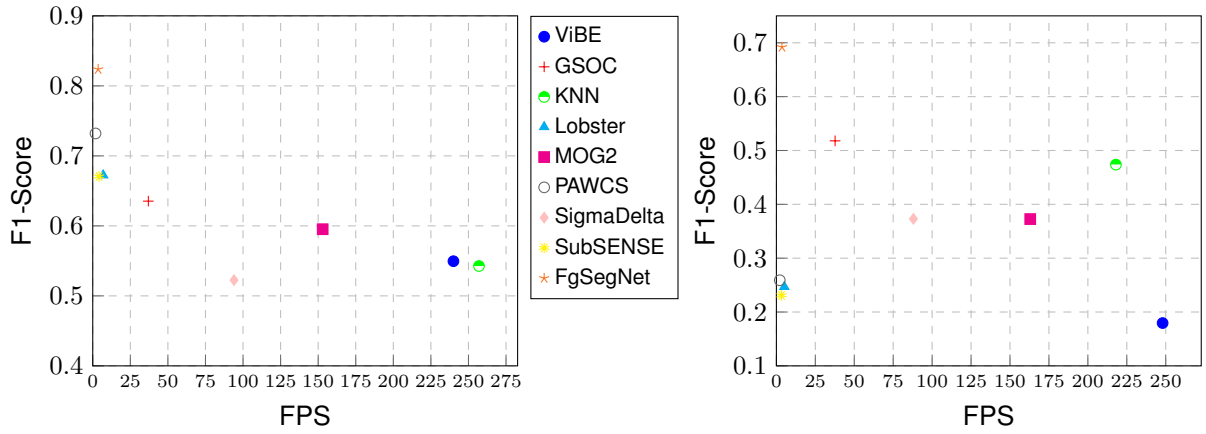


Figure 6.1: F1-score testing results for the algorithms in the coral reef tank on the left and the main tank on the right.

In this chapter, we will present the results of the experiments described previously.

6.1 Results of experiment A

We will present first the results of the background subtraction segmentation and then the YOLO detection.

6.1.1 Results A1: background subtraction

In this experiment we started by testing multiple algorithms in both environments (main tank and coral reef tank) and measuring the time it took to process the entire videos as well as computing the F_1-Score for the selected frames. The results are presented in Figure 6.1 for the coral reef tank on the left and for main tank on the right. For the coral reef, the approach with the best F_1-Score was the FgSegNet with 0.8326 followed by PAWCS, Lobster and SubSENSE. But between all these, there is not one that can perform above 7 FPS making them very slow in comparison with the speed of our videos (24 FPS). Despite not having a requirement to run the system in real time, we still want it to be as fast as possible while being very accurate. The next best algorithm is GSOC scoring 0.6353 at 37 FPS which is a very decent performance. On the main tank, the results are a bit worse. FgSegNet is again the best algorithm with 0.6918 of F_1-Score followed by GSOC with 0.5179. Every single algorithm performed better in the coral reef tank than in the main one. This was expected given that the main is at more depth causing more blur, there are schools present and the colors are more similar. This makes the object subtraction task a lot more difficult, resulting in worse results.

The second part of this experience was testing the two-pass variant, where we run the algorithms on the entire video first to build the background model and then evaluate them on the second run, to see if

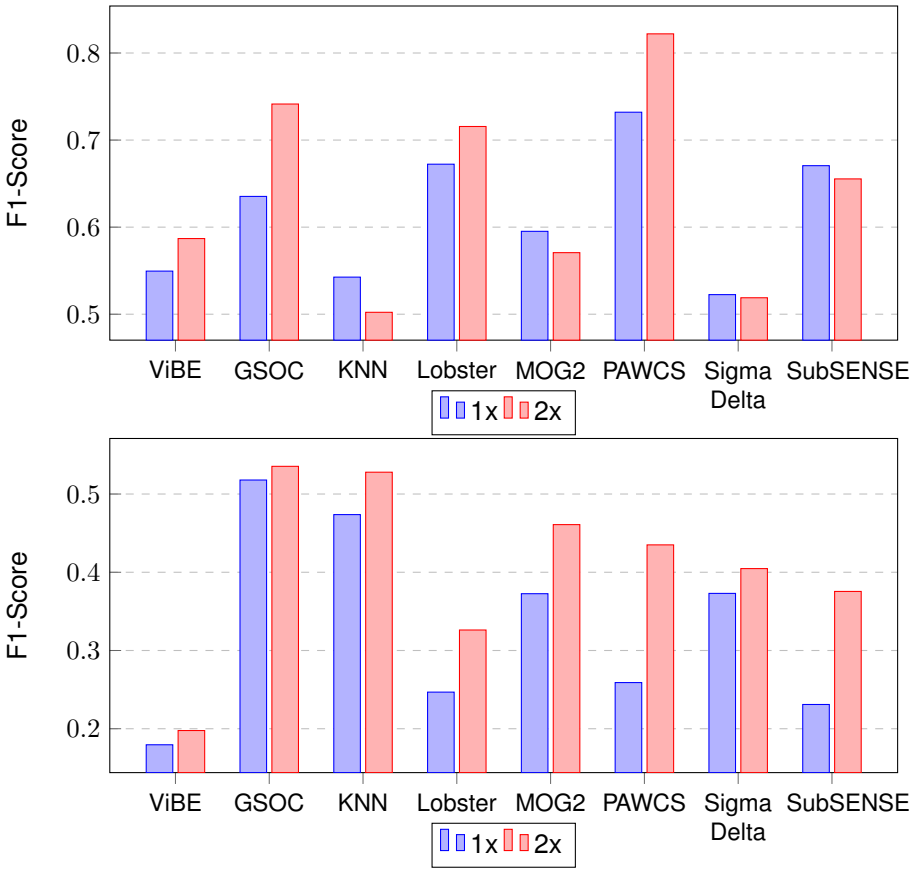


Figure 6.2: F1-score comparison for the single-pass and two-pass variants for the coral reef tank at the top and for the main tank at the bottom.

it could help the segmentation. The results and comparison with the single-pass variant are presented in Figure 6.2 with the coral reef tank at the top and the main tank at the bottom. In the coral reef tank, there was no global evidence of improvement in terms of F_1-Score. Some of the algorithms performed better and some performed worse. Still, we were able to achieve better top performances than what we had while running only once. The PAWCS was the top scorer with 0.8220 and GSOC came in second with 0.7414. Obviously, we have to take into consideration the extra time that is spent doing an extra pass through the entire video. In the main tank, on the other hand, there was a clear indication of improvements while running the video twice, as all algorithms benefited from it. The best algorithms were GSOC scoring 0.5354 at 18 FPS and KNN scoring 0.527 at an impressive speed of 109 FPS. The main tank is still performing worse than the coral reef tank.

In Figure 6.3, we can see an example of the results obtained on one of the frames used in the tests. The best algorithm for the one-pass version was FgSegNet and for the two-pass was the PAWCS. Both of them achieved very good results but they were slow. The two-pass version of GSOC was not as good as those two but the results were satisfactory at decent speeds, and is shown in the last row comparing

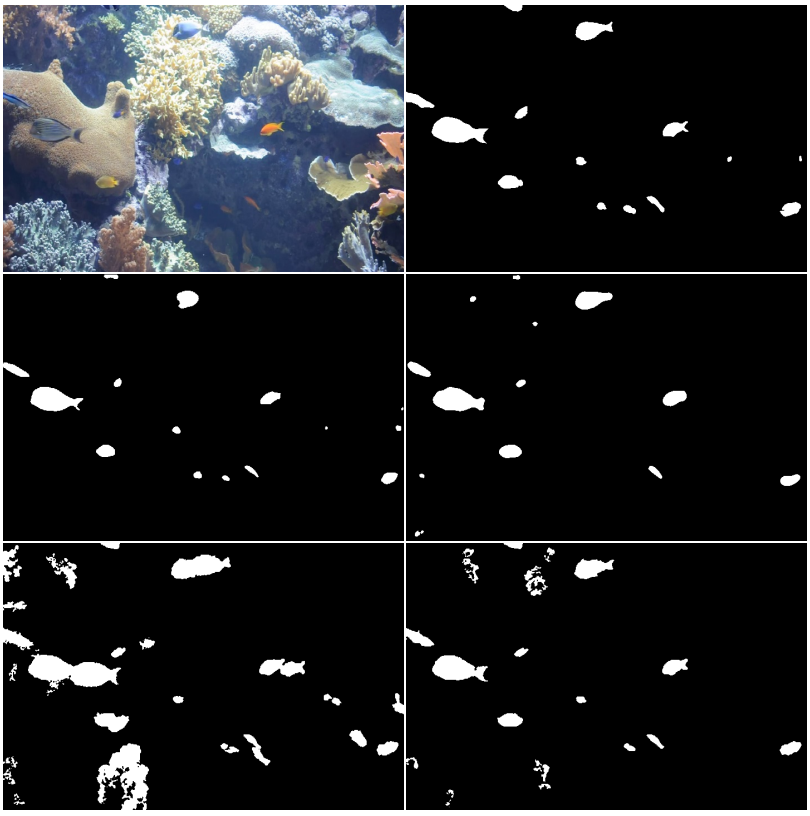


Figure 6.3: Results of background subtraction algorithms for the coral tank using the 24^{th} frame. On the top row there is the image on the left and the manually labeled mask on the right. On the second row, we have the result of FgSegNet on the left and the result of two-pass PAWCS on the right. On the third row we have the result of GSOC on the left and the two-pass GSOC.

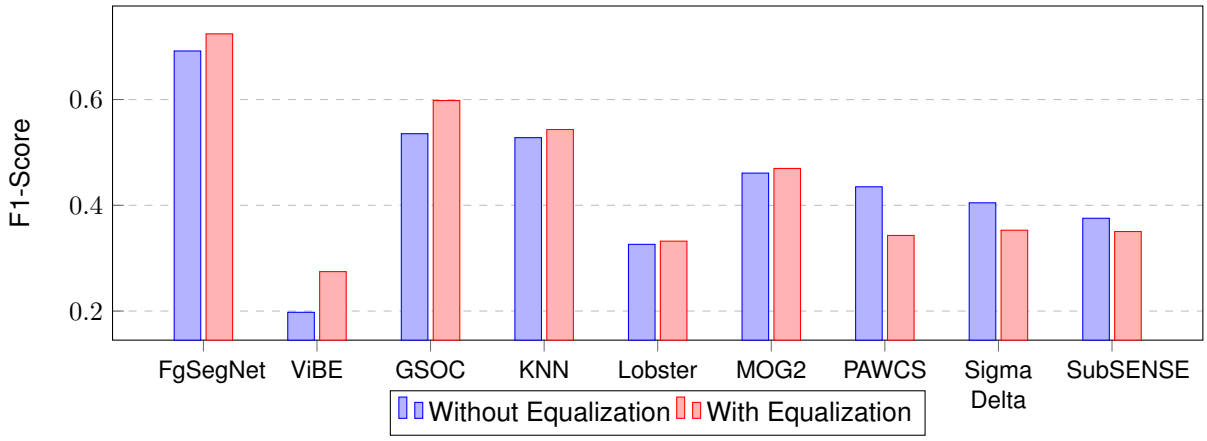


Figure 6.4: F1-score comparison for the use of color equalization for the main tank. The variant without equalization used with FgSegNet is the single-pass variant and the rest of them use the two-pass variant.

the one-pass version (left) with the two-pass version (right).

The last variant of this experience is applying the color equalization technique to the frames before running them through the algorithms, once again doing a first run to build the model and then a second to evaluate. This variant is only used in the main tank video and the results can be seen in Figure 6.4. It is possible to compare a variant of the algorithms using color equalization and not using it. In the variant without color equalization, the two-pass variant was used for all algorithms, except for the FgSegNet where the single-pass variant was used, as there is no two-pass variant for the network. Although the network has to be trained previously, it was considered as single-pass variant. The processing speed was decreased a bit as it also includes the equalization part. Once again, the best algorithms were FgSegNet with the best score but low FPS followed by GSOC.

In Figure 6.5, we can see how the background subtraction algorithms work in the main tank. On the first row, it is shown the result of applying the color equalization technique (on the right) to the original frame (on the left). Then we have the mask on the left of the second row followed by the FgSegNet, and the two-pass GSOC and two-pass KNN on the last row. All these examples use the equalized frame as the best results were achieved by using it. As we can see, the results are much worse than in the coral tank, especially in such early frames such as the one displayed (24^{th} frame, even when using the two-pass variant that computes a background model in advance.

6.1.2 GSOC Tuning

Overall, GSOC was one of the best algorithms with a good trade-off between speed and F_1-Score in most of the experiment variants, for both the main tank and the coral reef one. Therefore, that was the chosen algorithm to be used and now follows the tuning of its parameters. We tuned each environment separately. This parameter tuning was performed on the same testing dataset that was used in the

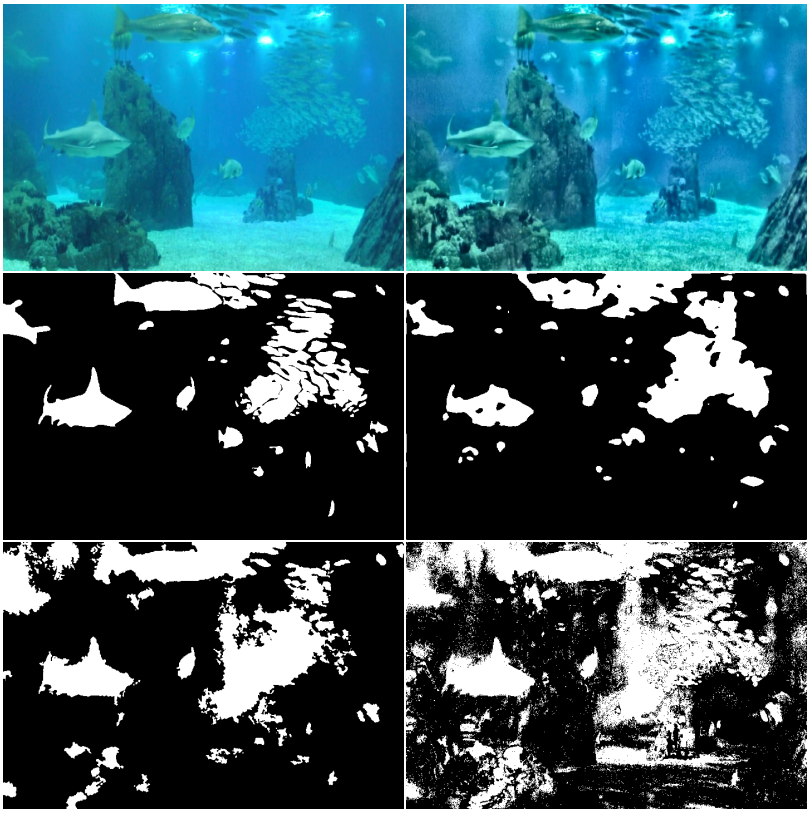


Figure 6.5: Results of background subtraction algorithms for the main tank using the 24^{th} frame. On the top row there is the image on the left and the frame after color equalization on the right. On the second row, we have the manually labeled mask on the left and the result of FgSegNet using the equalized frame on the right. On the third row we have the result of two-pass GSOC on the left and the two-pass KNN on the right, both using the equalized frame.

Table 6.1: Default values for the GSOC parameters.

Parameter	Default Value
Number Samples per pixel	20
Propagation Rate	0.01
Replace Rate	0.003
Hits Threshold	32

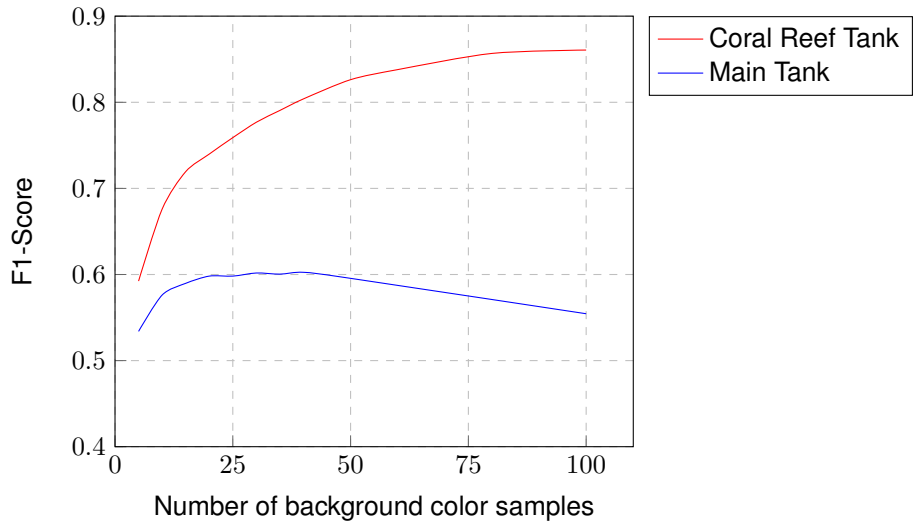


Figure 6.6: Evolution of F1-Score with the number of background color samples for each environment using GSOC.

previous tests. For the coral reef tank, we use the GSOC variant that runs the video twice. For the main tank, we go with the variant that applies the color equalization and runs the video twice. We studied each parameter individually, getting the value that produces a good result and using that value while tuning the remaining parameters. The parameters tuned are listed in Table 6.1 with the corresponding default values that had been used so far.

We start with the number of samples used per pixel. This parameter is the most impactful one. In Figure 6.6 is displayed how the F_1-Score varies with the number of samples for each of the environments. As the figure shows, we are able to improve the score in the coral reef tank by a decent amount by using more samples than we had been using so far, whereas in the main tank we can not achieve better results. There is a noticeable difference between the optimal number of samples to be used in the two environments. Using more samples means more processing time. In 6.7 we can see the how the processing time for a five minute video increases with the higher number of samples.

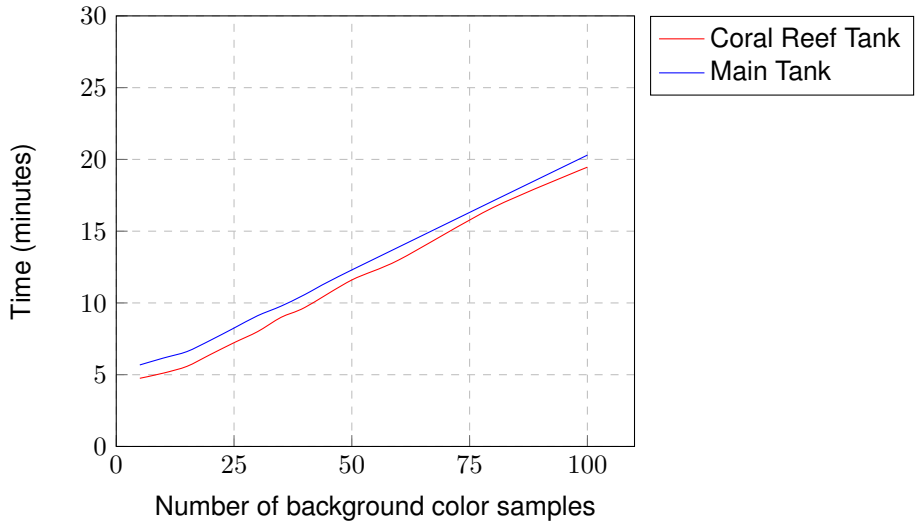


Figure 6.7: Evolution of processing time with the number of background samples for each environment (five minutes video each) using GSOC.

Table 6.2: Values obtained in the parameter fine tuning for GSOC

Parameter	Main	Coral reef
Number Samples per pixel	20	80
Propagation Rate	0.005	0.005
Replace Rate	0.005	0.001
Hits Threshold	32	40

To get the most out of the algorithm, in the main tank we use 20 samples and in the coral reef tank 80. There are three other parameters and all of them have less of an influence on both the score and the processing time. The first one is the propagation rate, $P_{propagation}$, that was tested with values between 0.003 and 0.1. The next is the replace rate, $P_{replace}$, and the values tested ranged from 0.001 to 0.05. Finally, the last parameter is the hits threshold, h_{thr}, and was changed in the interval 8 to 128. The best configuration achieved for both environments is presented in Table 6.2.

The final F_1-Score for the main tank is 0.5969 and for the coral reef tank is 0.8721.

6.1.3 Results A2: bounding box prediction

The results of this experiment are presented in Table 6.3. In the coral reef tank there are almost as many good detections as fish missed by the detector, leading to a lower Recall. In terms of Precision, it

scored higher as there were fewer detections that did not correspond to a real fish than good detections. Overall, most misses corresponded to the smaller fish that were not detected and also the ones that were under the coral in an area with low light and most bad detections were consistently around the same areas of the coral reef. In the main tank there were even less bad detections resulting in a high Precision while Recall was again low. In this scenario, the very few bad detections were mainly on high areas where the far away fish and the lights mixed together.

This CNN predicts with a certain confidence score that there exists an object of a certain class (in this case the only class which it can output is fish) at a certain location. So one way of improving the Recall would be to lower the confidence threshold from which we want to accept the output as valid, and it could also cause the Precision to lower. But in our case, these results were already obtained using a very low confidence threshold of 0.1 and still was not enough to detect the missing targets.

Once again, the main tank was evaluated with more caution by being more selective of the fish to be included as well as the areas to evaluate, whereas in the coral reef tank all fish were selected, so even though the results seem better in the main tank, it does not fully translate to performing better there than the coral reef tank.

6.2 Results of experiment B

In this section, we present the results for the tracking using two different object detection techniques, where we first tune the parameters using each one and then compare their final performance using the testing dataset.

6.2.1 Tracking using GSOC detector

In the coral reef tank, for which the evaluation is presented in figures 6.9 and 6.10, starting with the baseline version where we tuned both the color similarity threshold c_{thr} and distance threshold d_{thr}, we already had a decent performance. From 17 tracks, 13 of them had good predictions (predicted bounding box overlapping with the GT bounding box for more than 33%) in more than 80% of the frames, but there were some tracks where the tracking was too low, as seen in 6.10. We first varied the d_{thr} between

Table 6.3: Performance of YOLO object detector for both environments.

	Main	**Coral reef**
TP	36	40
FP	7	18
FN	26	39

	Main	**Coral reef**
Precision	0,8372	0,6896
Recall	0,5806	0,5063

50 and 80 and then the c_{thr} between 20% and 40%. The best performance was achieved by allowing associations in a 65 pixels radius relative to the last position and a minimum of 30% histogram similarity with the last detection.

Then we tested the integration of **color history**, i.e. averaging the histograms over time, by varying the weight of the histogram of the new detection. This value was tested between 10% and 90%. The baseline tuned earlier corresponds to having 100% weight for the new histogram as older histograms were fully replaced by the new one. Overall, this feature didn't bring much change according to Figure Figure, and we can also see that by looking at the performance of each track in the Figure 6.10, where there were only small changes in some of them between the first and second plots. The configuration chosen was 20% weight of the new histogram (and 80% to the older histograms).

By adding **temporary tracks**, we had a noticeable improvement on the overall performance by looking at the Precision and Success plots in Figure 6.9. Individually, two of the lower performing tracks so far were able to make predictions for a longer portion of the track, getting more than 80% of the frames with good predictions. One example of the tracker benefiting from having temporary tracks is shown in Figure 6.8. The fish being tracked is identified as number 1. Without temporary tracks, bad segmentation lead to an association error that was never able to recover later on, as another track started being associated with the fish (track 11). With temporary tracks, even though there was bad segmentation on certain frames, as the newly created tracks were temporary and had less priority in the association step, the original track kept being associated with the correct fish.

Finally, by adding **movement prediction** using the Kalman Filter, there were again some improvements and the only track that was failing almost completely was able to get more than 40% of the bounding boxes predicted with higher overlapping than 33% with the GT boxes.

Overall, the system was able to make a good prediction in the majority of frames. Most of the bounding boxes were centered with an error of less than 20 pixels in relation to their true position and the overlap was not perfect but still more than 50% IOU in around 80% of the frames. In the coral reef tank, some of the fish move quickly sometimes so having movement prediction helped the system recover tracks after failing for some frames, as the movement kept being predicted and kept lower distance than the one allowed for an association. The temporary tracks also helped in some cases were the segmentation was fragmented for the same fish and prevented a newly created tag for one of those segments "stealing" the correct match for the fish we were following when the segmentation was unified again.

In the main tank, the overall performance of the system is worse. The results for this tank are presented in Figures 6.12 and 6.13. For the baseline version, the best configuration for the color and distance thresholds could only get 6 out of 15 tracks with more than 80% of the frames with good matches. On the other hand, there were 8 tracks with less than 20% of the frames where a prediction

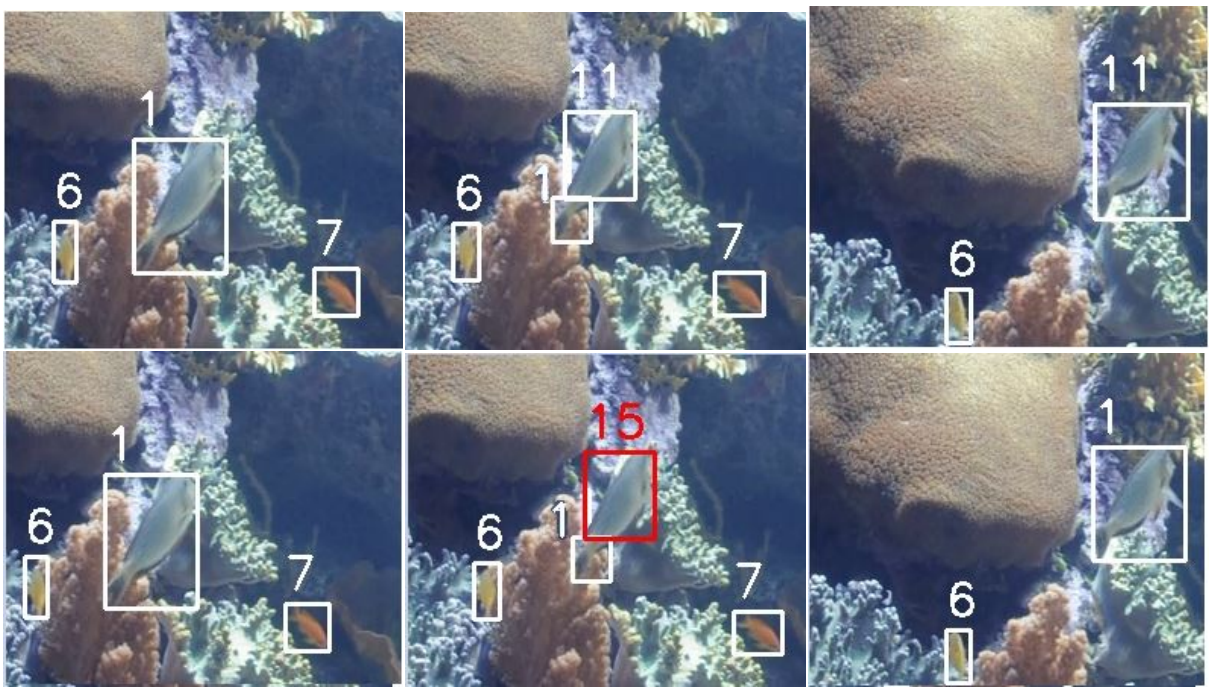


Figure 6.8: Example of the use of temporary tracks in the coral reef tank. In the top row, without using temporary tracks, the original target (identified as number 1) lost its target as another track that was new got the association. In the bottom row, using the temporary tracks (shown as red), the original track was able to keep track of the original target.

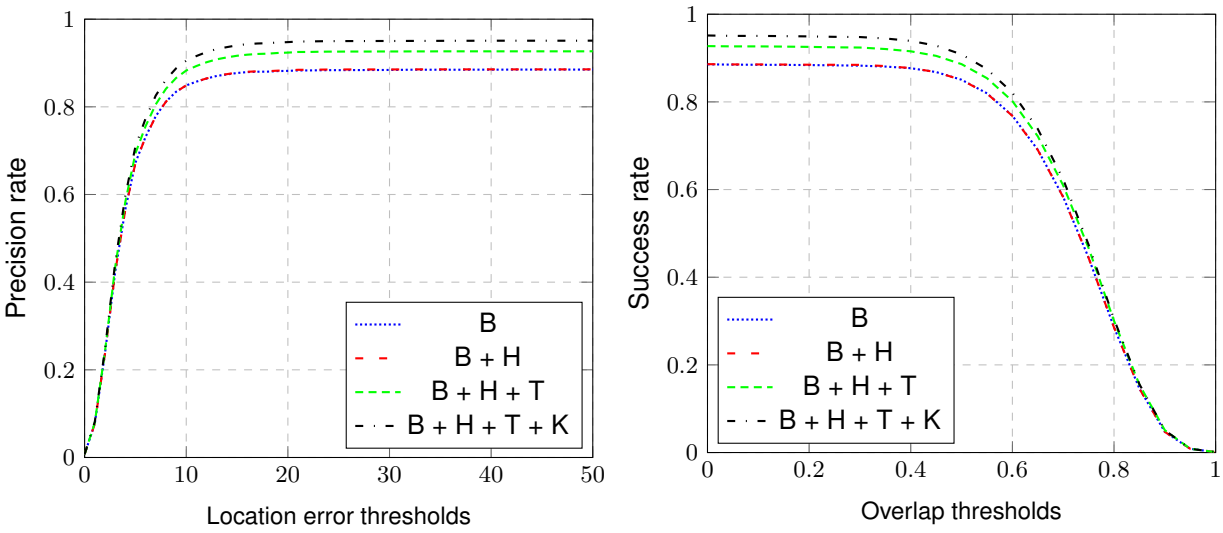


Figure 6.9: Precision and Success plots using GSOC for the training dataset in the coral reef tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter.

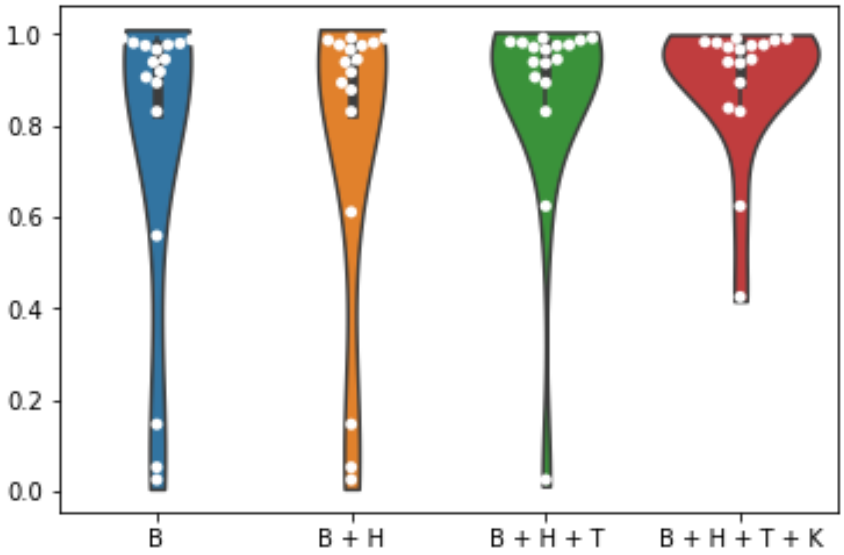


Figure 6.10: Distribution of the ratio of predicted boxes in each track with IOU higher than 0.33 for the different system implementations using GSOC for the training dataset. Each white dot in the same system implementation represents a different tested track in the coral reef tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter.

was classified as a good match. This resulted in a very low Precision and Success plot as there were a lot of frames with no prediction at all. The maximum distance threshold was set to 65 pixels and the minimum color distance threshold to 20%.

When adding the **color history** using histogram average, it helped some of the tracks and harmed others, but overall we can see an improvement in terms of Precision and Success, according to figure 6.12 (from the blue line to red one). There were still a high number of failing tracks, with some of them having less than 5% of frames with good predictions. The weight of the new histogram was tuned to 20% (and 80% for the old one).

Then we added **temporary tracks** and for the first time so far, the overall performance got worse than the previous iteration, which was the best until now. Before testing the final version of the system with every feature included, we decided to test a new variant where we used the Kalman Filter for movement prediction and turned off temporary tracks. Even though it performed better than using temporary tracks, it still was not as good as the version only using histogram averages, in overall terms.

In the end, using every feature including the ones that did not perform well, we were able to get the best performance. Some of the worse performing tracks got a bit of an improvement getting closer to 20% and 30%, but these values were still very low to be considered good results. In Figure 6.11, we can see the effect of the movement prediction on the fish identified as track 1. On the top row, with movement prediction disabled (meaning that the track position used to compare with the detections is the last known position) after a period of failed target detection, the original target gets associated with a different track (track 35), meanwhile the original track got associated with a fragment of other fish (the

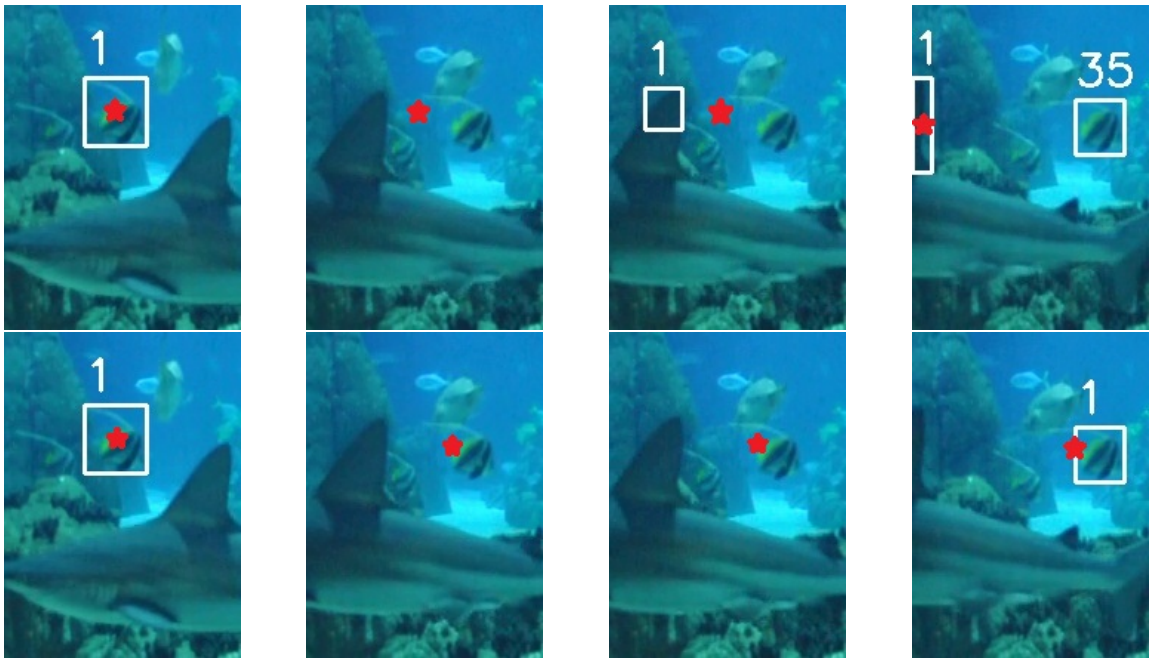


Figure 6.11: Example of using Kalman Filter in the main tank for movement prediction for the track 1. The red star is the position considered for the position similarity. In the top row, without using movement prediction, when detection fails the position remains the same in those frames, allowing incorrect associations afterwards. In the bottom row, using movement prediction, the position keeps being predicted until it eventually detects the fish and recovers the original target.

shark's fin). Using movement prediction, the position kept being updated following a path relatively close path to the real one and once the target was again detected, the association was successful.

So, overall the main tank was a lot more difficult to track and one of the main reasons is the poor object detection obtained by using the GSOC background segmentation technique. Also the fish have a lot more similarities in color, with a few being different, and they also blend in with the blue background, which is one of the reasons why the segmentation performed poorly too. We were still able to track correctly some of the trajectories.

6.2.2 Tracking using YOLO detector

The next experiment is tuning the system while using YOLO object detector. Once again, this part of the experiment was performed in a validation dataset.

In the coral reef tank, represented in Figures 6.14 and 6.15, we can immediately see a big decline of the results compared to the background subtractor GSOC. Starting with the baseline, we can see that precision is never more than 50% for an error of less than 50 pixels. It is also easy to see that there were more tracks with less than 50% of good predictions. The color threshold was tuned to 20% (compared to 30% using GSOC) and the distance threshold to 65 pixels (same as GSOC). Allowing a less similar histogram to be matched can be explained by how both approaches differ in building the histogram. In

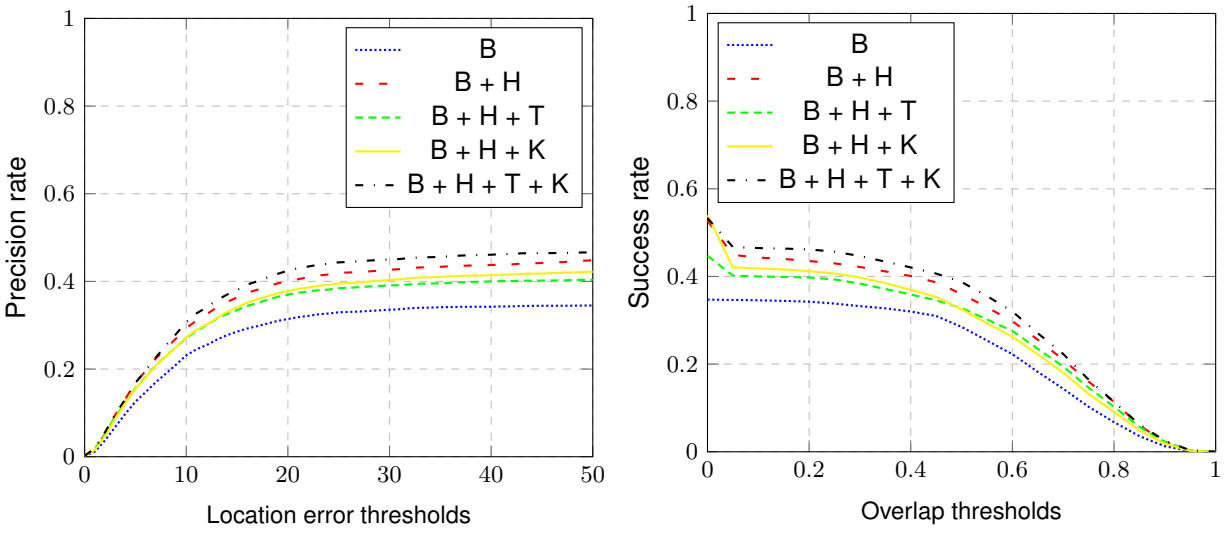


Figure 6.12: Precision and Success plots using GSOC for the training dataset in the main tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter.

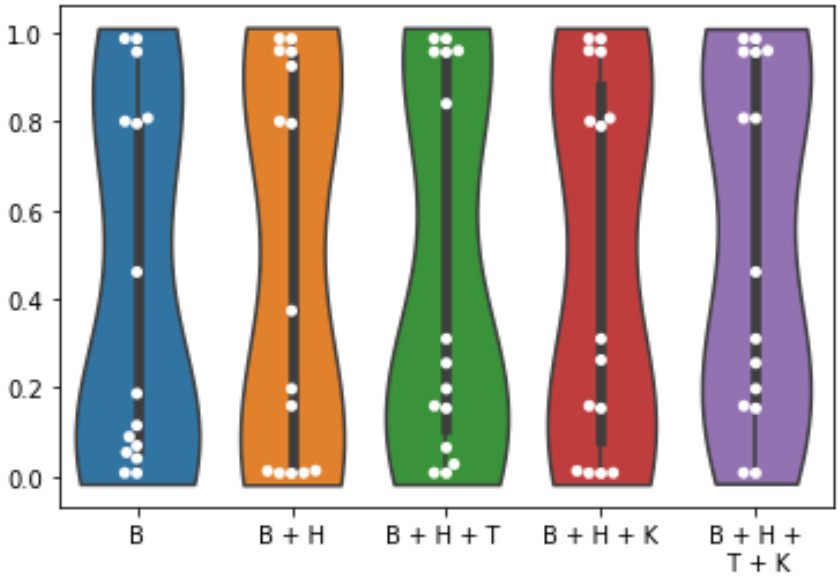


Figure 6.13: Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system implementations using GSOC for the training dataset. Each white dot in the same system variant represents a different tested track in the main tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter.

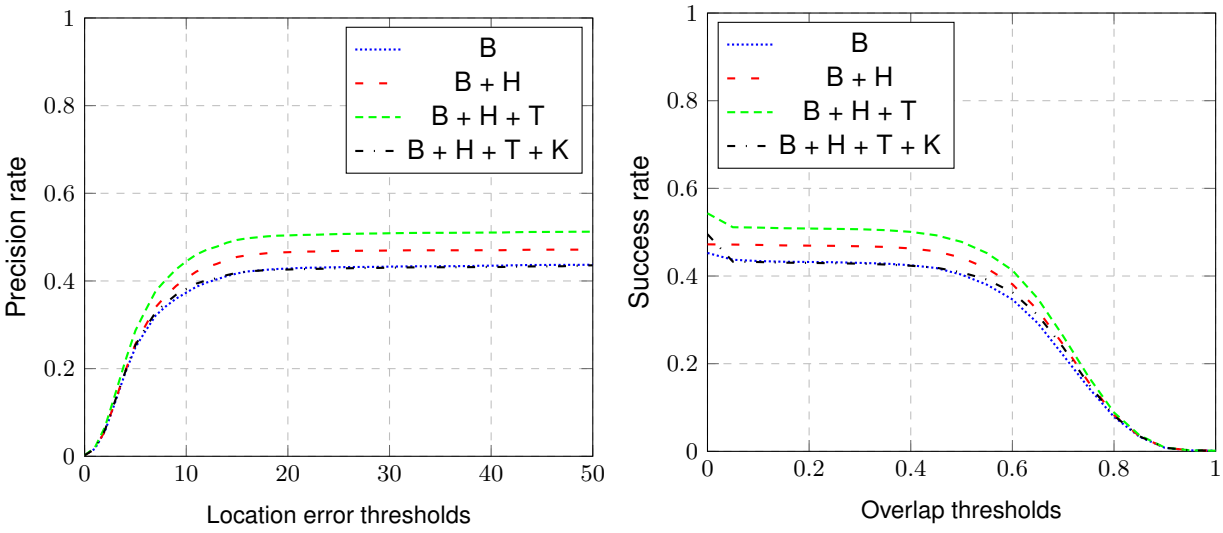


Figure 6.14: Precision and Success plots using YOLO for the training dataset in the coral reef tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter.

GSOC, only the pixels of the mask are used in the histogram whereas in YOLO all the pixels in the bounding box are used. So there will be sometimes more pixels belonging to the background present in the histogram in YOLO, so the color restriction has to be lower.

When adding **color history**, there was an increase in performance and the weight of the new histogram was tuned to 20%, the same value as the other version. Even though it improved the system, it was still a very low performance compared to using background subtraction.

Then we added **temporary tracks** and it showed better results once again in terms of overall performance. We had 10 tracks making good prediction in more than 50% of their frames but there were still some that only had less than 10% of good predictions.

Finally, by adding the **movement prediction** through Kalman Filter, the overall performance came down, achieving only the same performance as the baseline version in both Precision and Success. An explanation to this is the detector was not very effective to identify fish while they were swimming perpendicularly to the frame plane. So when the fish decided to make a curve with those characteristics, the detector would not identify them. With no detection and no association with the track, the position prediction would keep following the movement it had before and eventually get out of range of the fish that was still in a similar position of when it got lost. So in this environment it is best to turn off movement prediction while using YOLO.

Comparing the overall results in the validation dataset, it was clearly better to use the background subtractor GSOC instead of the object detector YOLO in the coral reef tank.

The results for the main tank are presented in Figures 6.16 and 6.17. This time, there is a much smaller difference between using the two types of detection, and there are also similar performances for

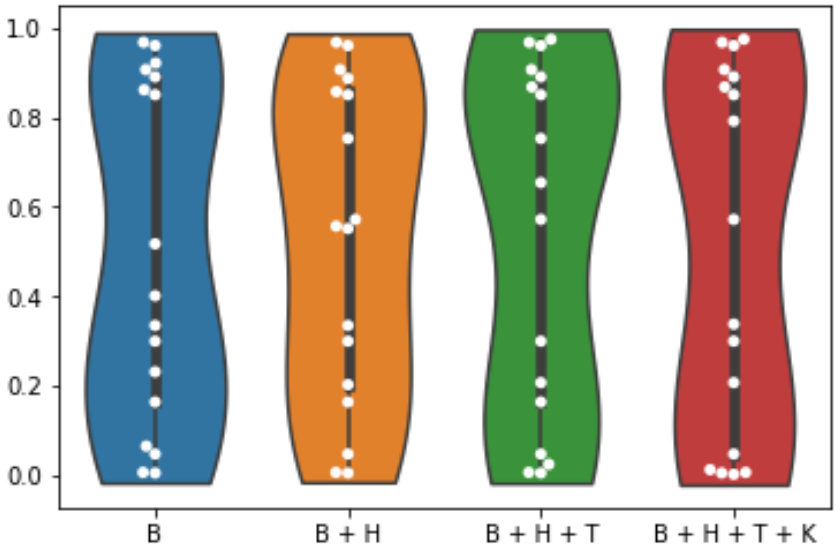


Figure 6.15: Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system using YOLO for the training dataset. Each white dot in the same system variant represents a different tested track in the coral reef tank.B - baseline, H - average histogram, T - temporary tracks, K - kalman filter.

each iteration with the different features.

The best version of the baseline was obtained with a color threshold of 30% (against 20% using GSOC) and a distance threshold of 80 pixels (65 with GSOC). When detecting multiple fish close together, in some cases, we were able to detect them separately instead of a big segmentation like with GSOC so we could increase the color similarity threshold a little bit and increase the radius in which we accept a match.

After adding **histogram averages**, both the Precision and Success stayed practically the same. This was possible by tuning the weight of the new histogram to 90%, which is very close to not having an average at all like before. If we used a smaller weight to new histograms, the performance would be lower. This means that probably we could not use the history of the detections, but we use it we nevertheless.

Adding **temporary tracks** had a marginal improvement to the overall performance. On the other hand, adding **movement prediction** on top of it made it marginally worse than the other three versions. So, like in the coral reef tank, the system performed better when using histogram averages and temporary tracks but with the movement prediction turned off, while using Yolo as the object detection technique.

6.2.3 Tracking: GSOC vs YOLO

This experiment uses the testing dataset with trajectories different from the ones in the validation dataset used so far. With these results we make the final direct comparison between the use of both detection

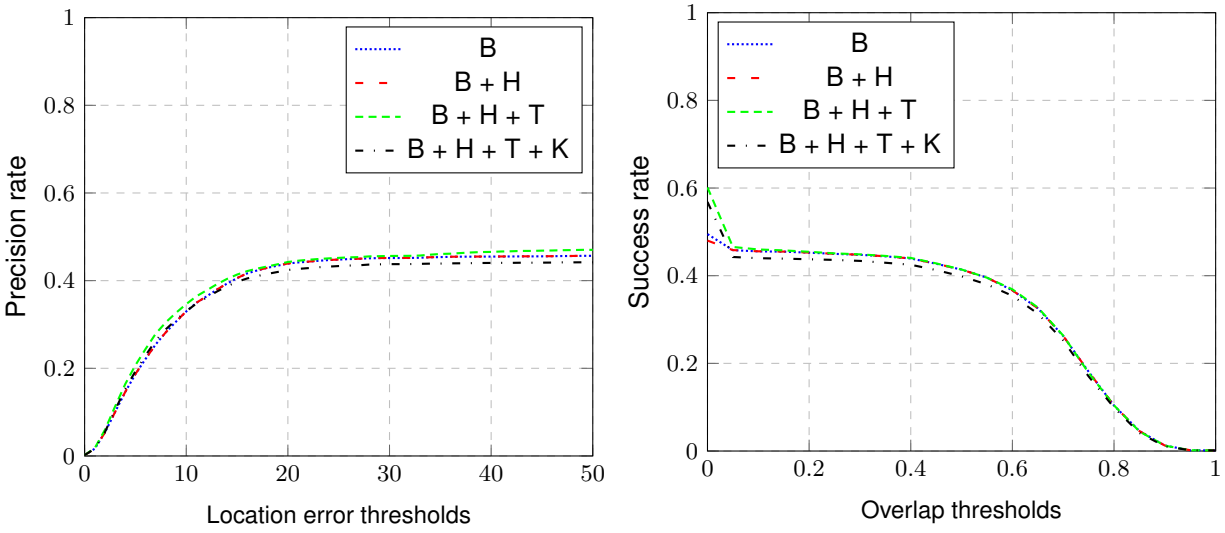


Figure 6.16: Precision and Success plots using YOLO for the training dataset in the main tank for the different system implementations; left: as a function of location error; right: as a function of overlap. B - baseline, H -average histogram, T - temporary tracks, K - kalman filter.

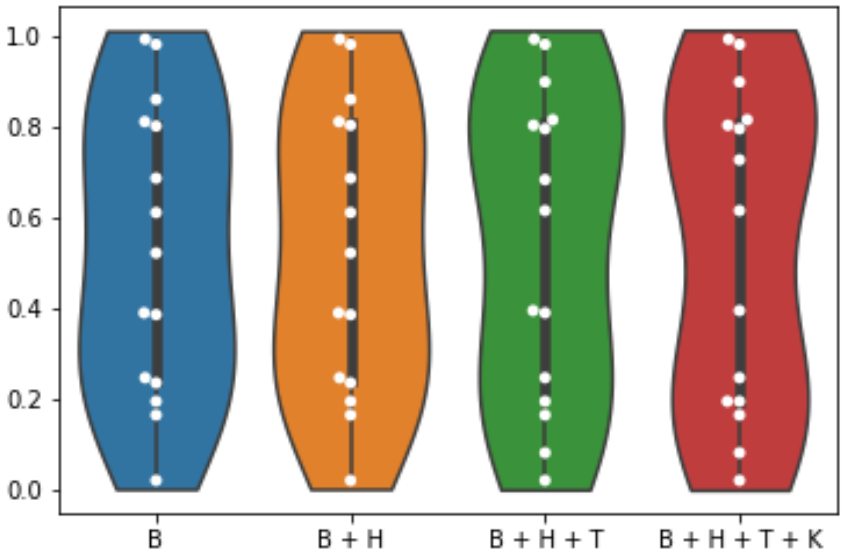


Figure 6.17: Distribution of the ratio of predicted boxes with IOU higher than 0.33 for the different system implementations using YOLO for the training dataset. Each white dot in the same system variant represents a different tested track in the main tank. B - baseline, H - average histogram, T - temporary tracks, K - kalman filter.

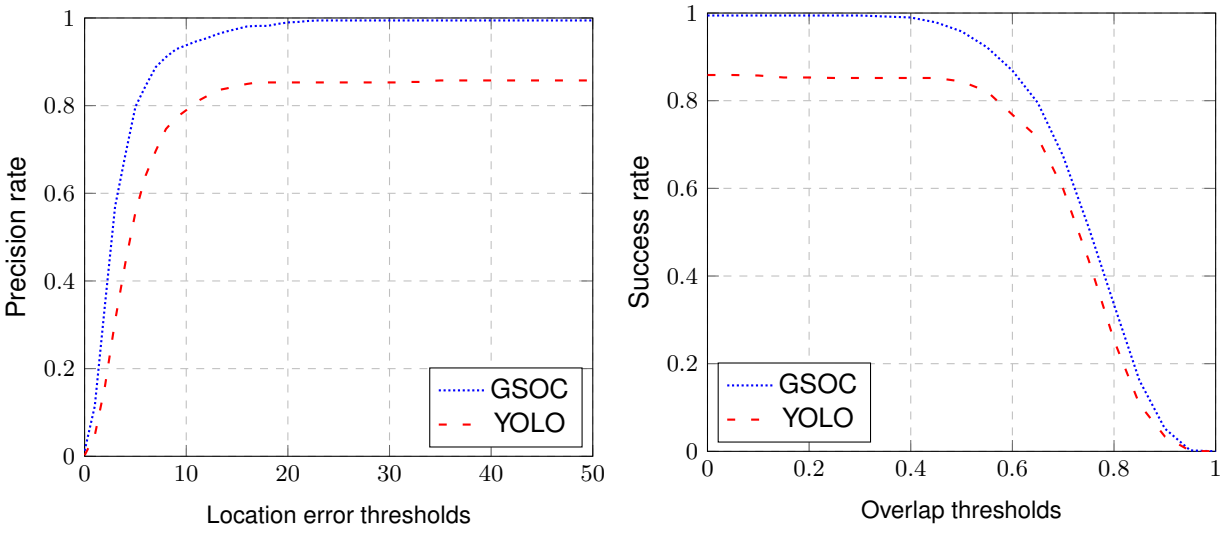


Figure 6.18: Precision and Success plots for the system implementations with each object detection technique for the testing dataset in the coral reef tank; left: as a function of location error; right: as a function of overlap.

techniques for each tank, using the tracking configurations obtained in the last sections. The final results for the coral reef tank using both detectors are laid out in Figure 6.18.

We can see that GSOC achieved the best results in the tracks of this small testing dataset in both metrics. In terms of Precision, there were around 90% of frames where the system predicted a bounding box with a center error of 10 or less pixels. Regarding the Recall, there were more than 80% of frames with an overlap of 60% with the corresponding ground truth bounding boxes. With YOLO, there was a noticeable decrease in the results but overall it was still a good performance. For all 4 tracks of this testing dataset, both versions were able to track them from the start to the end, meaning that the difference in results is probably caused by more failed detections or failed associations in some frames along the way by the YOLO version than by the GSOC version.

For the main tank, the results are shown in figure 6.19. In this tank, in line with what happened with the validation dataset, the results were worse when compared to the coral reef tank. This time the best tracking was achieved while using YOLO as object detector. There were more than 30% of frames in which the predicted box did not exist or its center was too distant, in both versions. The Success plot shows us that there were a similar number of frames without a good overlapping with the ground truth. These values can be explained by the loss of the targets being tracked in both system versions. Using YOLO, there were 2 tracks followed from start to end, 1 track that was lost near the end and the last one failed around the end of the first half of the track. As for GSOC, it also tracked 2 tracks from start to finish, but one track failed really early on and the other did not get to half-way too.

So, overall, is better to use the background subtraction algorithm in cases like the coral reef tank, where the fish are not always together (but can still cross paths) and are smaller. This means that when

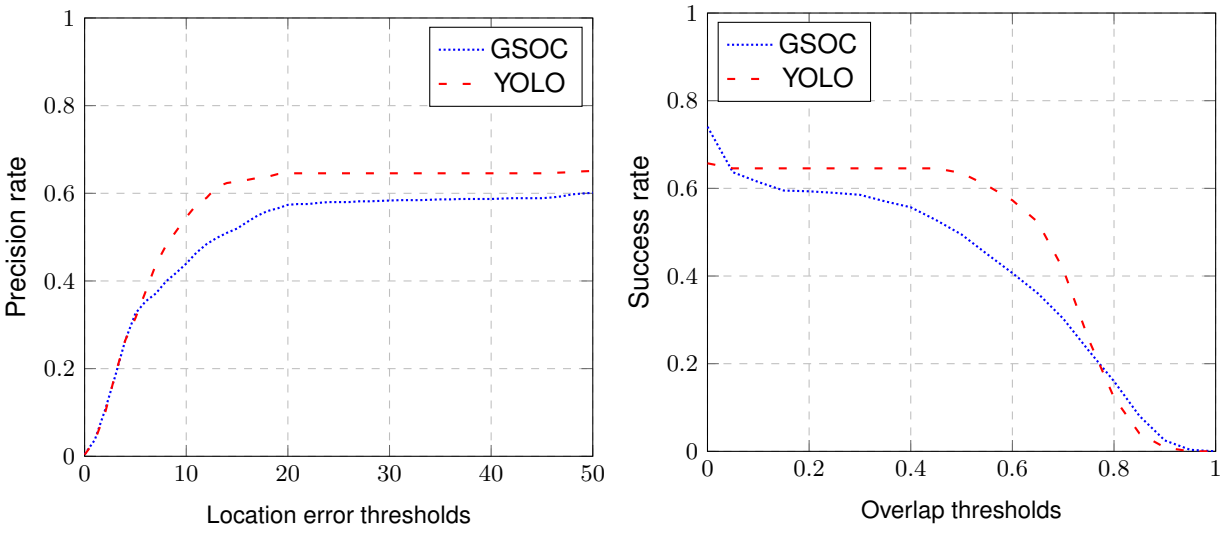


Figure 6.19: Precision and Success plots for the system implementations with each object detection technique for the testing dataset in the main tank; left: as a function of location error; right: as a function of overlap.

they turn, the area of the body captured by the camera is also very small. As the segmentation is not dependent on the form of the fish, they can still be detected in very different shapes and orientations as long as the detection algorithm can discriminate the fish's colors from the background ones. On the other hand, there are environments like the main tank, with a huge number of fish present, where they frequently swim together and in front of others that are further away from the camera. In this types of scenarios, it is better to use an object detector like YOLO that is capable of identifying fish separately even if they are touching each other in the frame space.

In terms of tracking features, all the best versions used histogram averaging, that was used as a history of the recent histograms, and temporary tracks, that gave less priority to tracks that were recently created. As for movement prediction, it was an improvement while using background subtraction but it decreased the performance while using the object detector YOLO.

7

Conclusion

Contents

7.1 Future work 77

In this work, we attempted to solve the problem of tracking applied to animals, in this case fish. This is a very relevant topic given the many concerns today about preserving our oceans and the animals that live in it. Having an automatic method that can successfully track fish can be a huge help to the organizations that monitor and study them, like Oceanário de Lisboa. There are many difficulties imposed by the natural conditions of the marine life such as multiple fish present in the same area, the camouflage behaviour that the animals use, a lot of visual similarities between different individuals of the same species and sometimes between different species, that are hard to be detected by non-specialists. We worked with two types of marine scenarios, one that portrays a coral reef ambient and one that mimics open waters. Even though the environments were closed tanks, they were created by biologists to recreate real life conditions.

Our work can be divided in two main parts: the first one is a comparative study between different types object detection techniques. This included a comparison between background subtraction algorithms, both traditional and through the use of neural networks, and a technique that predicts the bounding boxes directly. One of the problems of doing such studies is the big amount of data required as those datasets had to be manually created, so there had to be a balance between the size of the dataset and the time spent creating it. For both environments, the best performing background subtraction algorithm was the GSOC that was later selected to be used in our detection module.

The second part of our work is the development of the full tracking system. It is composed of an optional pre-processing step, followed by a detection module and finally the tracking step. Throughout the development and testing, the tracking features were developed to try to solve some occurring issues. The system included two versions of the detection module, one using GSOC and another using Yolo, the CNN detector that predicts the bounding boxes directly. This way, we could compare the impact of the different types of detection approaches on the tracking results. The best version of the main tank was achieved using Yolo while the coral reef tank performed better using GSOC.

Overall, the results were satisfactory, but there are some limitations to our approach like the inability to handle the big fish schools swimming around as they make it very hard to track each individual fish. The detection does not work well with fish that are very far away or the very small ones in the coral reef tank so tracking these is not possible.

7.1 Future work

One thing that could still be done to improve the tracking performance would be the addition of CNN features in the similarity matrix computation on the data association step of the tracking algorithm. By running the frames through Yolo, we could extract these automatically generated features from selected layers and compare the features of the tracks with the features from the detections through some sort of

similarity function. Alternatively, some kind of hand-crafted feature based on the visual patterns shown by the detected objects and the tracked could be used, instead of only using color histograms as the spatial arrangement is lost.

It is also important to research a mechanism to detect fragmentation in the segmentation of the fish, that could be used to rebuild a new bounding box when such events happen. This detection is no easy as big changes in the size of the bounding box between frames does not always means that there was a fragmentation problem. One example of that is when two fish were very close being detected as a single box and then separate creating two smaller boxes.

Finally, some research can be done in the combination of both detectors (GSOC and YOLO) into one, as each of them works well in situations that the other does not, so they could complement their detections.

Bibliography

[1] M. Babaee, D. T. Dinh, and G. Rigoll. A deep convolutional neural network for video sequence background subtraction. *Pattern Recognition*, 76:635 – 649, 2018.

[2] O. Barnich and M. Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image Processing*, 20(6):1709–1724, 2011.

[3] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[4] G. Bilodeau, J. Jodoin, and N. Saunier. Change detection in feature space using local binary similarity patterns. In *2013 International Conference on Computer and Robot Vision*, pages 106–112, 2013.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] M. Braham and M. Van Droogenbroeck. Deep background subtraction with scene-specific convolutional neural networks. In *2016 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 1–4, 2016.

[7] François Chollet et al. Keras. https://keras.io, 2015.

[8] A. Dehghan and M. Shah. Binary quadratic programing for online tracking of hundreds of people in extremely crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):568–581, 2018.

[9] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. In David Vernon, editor, *Computer Vision — ECCV 2000*, pages 751–767, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[10] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 175–181, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[11] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar. Changedetection.net: A new change detection benchmark dataset. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2012.

[12] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.

[13] M. Hofmann, P. Tiefenbacher, and G. Rigoll. Background segmentation with feedback: The pixel-based adaptive segmenter. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 38–43, 2012.

[14] K. Iqbal, M. Odetayo, A. James, R. A. Salam, and A. Z. Hj Talib. Enhancing the low quality images using unsupervised colour correction method. In *2010 IEEE International Conference on Systems, Man and Cybernetics*, pages 1703–1709, 2010.

[15] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[16] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis. Real-time foreground–background segmentation using codebook model. *Real-Time Imaging*, 11(3):172 – 185, 2005. Special Issue on Video Object Processing.

[17] D. Konovalov, A.Saleh, M. Bradley, M. Sankupellay, S. Marini, and M. Sheaves. Underwater fish detection with weak multi-domain supervision. *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019.

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[19] Y. Li and J. Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, pages 254–265. Springer International Publishing, 2015.

[20] Long Ang Lim and Hacer Yalim Keles. Learning multi-scale features for foreground segmentation. *Pattern Analysis and Applications*, 23(3):1369–1380, 2020.

[21] Antoine Manzanera and Julien C. Richefeu. A new motion detection algorithm based on sigma-delta background estimation. *Pattern Recogn. Lett.*, 28(3):320–328, February 2007.

[22] Douglas J. McCauley, Malin L. Pinsky, Stephen R. Palumbi, James A. Estes, Francis H. Joyce, and Robert R. Warner. Marine defaunation: Animal loss in the global ocean. *Science*, 347(6219), 2015.

[23] Camilo Mora, Derek P Tittensor, Sina Adl, Alastair GB Simpson, and Boris Worm. How many species are there on earth and in the ocean? *PLoS Biol*, 9(8):e1001127, 2011.

[24] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[25] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[26] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[27] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[29] Andrews Sobral. BGSLibrary: An opencv c++ background subtraction library. In *IX Workshop de Visão Computacional (WVC'2013)*, Rio de Janeiro, Brazil, Jun 2013.

[30] P. St-Charles and G. Bilodeau. Improving background subtraction using local binary similarity patterns. In *IEEE Winter Conference on Applications of Computer Vision*, pages 509–515, 2014.

[31] P. St-Charles, G. Bilodeau, and R. Bergevin. A self-adjusting approach to change detection based on background word consensus. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 990–997, 2015.

[32] P. St-Charles, G. Bilodeau, and R. Bergevin. Subsense: A universal change detection method with local adaptive sensitivity. *IEEE Transactions on Image Processing*, 24(1):359–373, 2015.

[33] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 246–252, 1999.

[34] R. Wang, F. Bunyak, G. Seetharaman, and K. Palaniappan. Static and moving object detection using flux tensor with split gaussian models. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 420–424, 2014.

[35] M. Wu and X. Peng. Spatio-temporal context for codebook-based dynamic background subtraction. *AEU - International Journal of Electronics and Communications*, 64(8):739 – 747, 2010.

[36] R. Zhou, K. Zhou, M. Wu, and J. Teng. Improved interactive multiple models based on self-adaptive turn model for maneuvering target tracking. In *2018 Eighth International Conference on Information Science and Technology (ICIST)*, pages 450–457, 2018.

[37] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 28–31. IEEE, 2004.

[38] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773 – 780, 2006.