Focusing a subset of objects in a stream of images

João Teixeira
IST, 84605

Instituto Superior Técnico, Lisboa, Portugal
https://tecnico.ulisboa.pt
joao.santos.teixeira@tecnico.ulisboa.pt

Abstract. To study animals and their surroundings, there is no longer the need to have a specialist on the field observing them and registering what they see. Instead, recording systems can be used and with existing technology, good enough quality videos are not very expensive to obtain. This is where topics related to computer vision are useful and enable us to process the videos with many different objectives. One of them, being the theme of this work, is object tracking. Videos can be used to track animals (in our case fish) and study their trajectories, patterns, relationships, etc. To do this, we first detect our targets by using object detection techniques and then we track them. Some challenges arise like occlusion between the objects or size variations that have to be dealt with in order to achieve a good tracking performance.

Keywords: Object Tracking · Object Detection · Fish · CNN · KCF · Interactive Multiple Model

Table of Contents

1 Introduction

By simply observing a target, we can identify and study its behaviours. But this task can be very time consuming given that some behaviours don't occur often and usually there has to be a specialist observing it to make a valid analysis. Fortunately, nowadays it's very easy to make recordings over long periods of time. Although the specialist still has to analyse the entire video, there is no need for the person to be at the same place as the subject, so more things can be captured. To overcome the time issues, visual tracking has been proposed in order to reduce the need of manual observation of videos. The evolution in technology has made possible to create systems that capture video with quality, whether that's a simple smartphone recording or a closed circuit television system that covers big areas in real-time. Computer vision can be a powerful tool in many areas like security (to track persons of interest in a crowd or to automatically identify threats), driver-less vehicles (identify the surrounding vehicles and their actions), traffic control (control traffic lights based on the number of cars), human/animal observation, etc.

The main focus of our work is tracking fish in closed tanks. Special marine animal centers like Oceanário de Lisboa, in addition to providing exhibitions to the public, are also responsible for conservation of species and studying them. To observe every behaviour would mean to have a biologist observing each tank every minute. But with a system that records and tracks fish, biologists can focus more on the video sequences that actually portray interesting behaviours with the benefit of being able to easily save those parts to later share with the rest of the scientific community. Having the fish in closed environments has some positive aspects. First of all, the fish aren't going anywhere since they can only swim in area they are confined to, meaning that for small tanks there is the possibility that the camera recording the tank might be able to catch the entire length of the it. Secondly, as the tanks are made of transparent glass, the recording system can be positioned on the outside of the tank, not disturbing the fish inside and not causing an impact on the fish behaviour due to its presence inside the tank.

1.1 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.

1.2 Structure

In Section 2, we review the state-of-the-art approaches in topics relevant to our work. In Section 3 we propose a solution that addresses the problem of tracking fish in our scenario. To evaluate our proposition, in Section 4 an evaluation method is defined. The scheduling is defined in Section 5 and we conclude our work in Section 6.

2 Related Work

Over the years, a lot of object tracking articles have been published, being one of the most studied topics of computer vision. Some research works tried to solve specific problems, like tracking people or cars, while some others try to create a robust algorithm that is usable in different types of scenarios with varying environment challenges. One work that was used as inspiration to ours [6], developed a system to track and classify fishes in tanks. The system was divided in three main components: fish identification through background subtraction, tracking of the identified fish and classification of the type of each fish. Although our work doesn't focus on the classification, it will try to deal with the occlusions between fish, one of the main problems encountered that directly affected tracking as it depended on having objects detected separately in order to track them ,and also improve the tracking itself by using other movement prediction approaches and more adequate techniques when associating detected objects with the currently tracked ones. The detection module will also be improved by using other methods as the one used provided good enough results but it still left room for improvements.

Pre-processing

To track objects, it is important to understand how we can extract information from the videos. Videos are collections of ordered frames/images. One of the most important characteristic of an image is its colors. To represent a color, we need to use a color space. Depending on the color space, an image can be monochromatic or colored. An example of a monochromatic color space is the grayscale. Using grayscale, each pixel of an image is represented by a single value that corresponds to the intensity of the light, going from black (lowest value of intensity) to white (highest value of intensity). Everything in between is a different shade of gray.

There are multiple colored color spaces, being RGB the most known. RGB uses three channels (Red, Green, Blue) to represent the color for each pixel. RGB is an additive color model, which means that each color is the sum of the value of the three components. 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 HSV, which describes a color using three components: Hue, Saturation and Value. Hue is the variation 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.

In [17], different color spaces were compared to see their influence on a background segmentation algorithm. Some of the tested color spaces included RGB, YCbCr, HSI (which has some similarities to HSV) and normalized RGB (rgb). The experiments consisted in making small changes in an RGB color and then converting it to another color space to see how different it was from the original color. To test the noise, a small value was added in the three channels of the RGB for both a dark color and a bright color. For a bright color, the influence was low, but in the case of a dark color, a small variation in RGB resulted in small variation only in YCbCr. To test the light variation, the channels of an RGB light color were multiplied by 0.7 and the channels of a dark color multiplied by 1.4. Only thr channel I in HSI changed (given that it is the channel for the light) and YCbCr had also a significant change in the light channel Y and some minor changes in the other channels. Overall, YCbCr was the least sensitive to noise, but sometimes it is more sensitive to shadows than RGB, as some experiments on images showed.

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 [15] and [12]. 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 simpler 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) only uses the values from a square neighbourhood for each pixel to build the histograms to equalize.

In [15], a system to detect fish used another color space, CIELAB, where they applied histogram equalization. CIELAB, or L*a*b*, has a channel L* for the lightness and channels a* and b* for the color. The first color channel 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. After they transformed the color space to CIELAB, they applied a histogram equalization technique called Contrast Limited Adaptive Histogram Equalization (CLAHE), that is a variant of it's simpler version AHE, 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 and with bad

contrast. 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 [12], Unsupervised Color Correction Method (UCM), also tries to enhance the quality of underwater images. The first step is to equalize the RGB colors by dividing 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 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.

Background subtraction

Another computer vision topic relevant to this work is background subtraction. These algorithms try to model the background in order to detect changes when moving objects appear in the scene. This way, it is possible to separate the foreground from the background and extract those objects. There are some very different types of approaches to model the background, including probabilistic, sample-based, codebook-based and deep-learning.

One of the most studied approaches is the Gaussian Mixture Models (GMM) first introduced in [8] but it has seen many different variations and improvements over the years. One of the most relevant works is [25]. 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. Each pixel is modeled by a mixture of K Gaussians, given the observed color intensities over time. 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 (the size of $\omega_{i,t}$) 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.

The authors of [22], while working with underwater videos to detect fish, developed a covariance based model to identify the objects (fishes) in the water. In that approach there is a model for both the background and the foreground. For each pixel and the neighbouring pixels in a $w \times w$ neighbourhood, are retrieved their coordinates (x,y), RGB values, H value from HSV and the first four statistical moments of the Local Binary Patterns (LBP) [21]. Then the covariance matrix of these features is calculated and using its eigenvalues, the variance (v)

is calculated by doing the square root of the sum of the squared eigenvalues. Now, that pixel is represented like (x, y, v), a three-dimensional space, and using the set of background samples $(\psi_b = b_1, ..., b_n)$ and the set of foreground samples $(\psi_f = f_1, ..., f_m)$ and using Kernel Density Estimation (KDE), they calculate two different probabilities for a pixel p:

$$P(p \mid \psi_b) = \frac{1}{n} \sum_{i=1}^{n} \varphi(p - b_i) \tag{1}$$

$$P(p \mid \psi_f) = \frac{1}{m} \sum_{i=1}^{m} \varphi(p - f_i) \tag{2}$$

where $\varphi(x)$ is a Kernel Density Estimation function. For better performance, they quantize the previous 3D space into a $X \times Y \times V$ discrete space and use a discrete KDE kernel. Now, it's possible to use matrices to store the probabilities of a variance value for each pixel position. The matrix P_b stores the information corresponding to the probability (1) and P_f information of probability (2). To initialize the model, the first N frames are used. First, the vector (x, y, v) is calculated. Then, not only the $P_b(x, y, v)$ is updated but also the n adjacent cells (in both directions) for the same position, x and y, are updated decreasing the value updated as the distance between the adjacent cells and the original v cell increases. All P_f cells are set to a small value γ (the authors used $\gamma = 0.1$). To classify new pixels as background or foreground, it's only necessary to access both matrices in the correct position (x, y, v) and the pixel is classified as background/foreground based on the log-likelihood ratio. To update the background model, all new pixels are used and the same steps for initialization are followed. Then, a weighted mean is performed between the old model and the new. For the foreground model, KDE estimation is only applied to pixels classified as foreground and a small $\gamma = 0.1$ is added.

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 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 a radius R (which they considered to be 20). If the number of close 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 (3)

$$P\left(d, t + dt\right) = \left(\frac{N-1}{N}\right)^{(t+dt)-t} \tag{3}$$

where N is number of samples kept for each pixel. Such probability can be rewritten as (4)

$$P\left(d, t + dt\right) = e^{-\ln\left(\frac{N-1}{N}\right)dt} \tag{4}$$

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 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, very 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 of 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.

Another background subtraction algorithm is PBAS [11], 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 see if the new value is close to $\#min$ 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 $\{B_k^v(x_i), B_k^m(x_i)\}$. The authors evaluated and tuned the parameters using the dataset from the Change Detection Challenge [9]. The dataset is composed of videos with different challenges, including dynamic background and shadows. 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. It scored the highest average F1-score of 0.7532 and the lowest percentage of wrong classifications of 1.7693, averaging 48 fps. 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. The tests showed that using the gradient magnitudes, the F1-score increased by 2% and the percentage of bad classifications decreased by 0.13%.

Another approach is PAWCS [23] following the work of [14] and [27], while also using some ideas of ViBE [2] and PBAS [11]. 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 at time t is computed as

$$q(\omega, t) = \frac{n_{occ}}{(t_{last} - t_{first}) + 2 \cdot (t - t_{last}) + t_0} \tag{5}$$

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 if its previous value is lower than an exponential D_{min}. The persistence threshold is computed as

$$W(x) = \frac{q(\omega_0, t)}{R(x) \cdot 2} \tag{6}$$

where $q(\omega_0, t)$ is the first word of the local dictionary, taking into account that it is sorted by persistence value. The authors tested the algorithm against the dataset from 2012 CDNet challenge using some approaches evaluated during the challenge and some others. The F1-score was computed for each category and it outperformed all the algorithms in all 6 categories and had an overall F1-score 7% higher than the second best approach tested.

SubSENSE [24], by the same authors of [23], has the same principles of [23] 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.

More recently, the study of Convolutional Neural Networks (CNN) applied to background subtraction has also 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. One of the first approaches was [5]. The algorithm is separated into several parts. On the first part, the background is extracted 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 outperfomed 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 scene.

The authors of [1] also propose an algorithm using CNNs without the drawback of the last one. They extract background samples using SubSENSE algo-

rithm [24] (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, 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 [26] 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

$$bm = \begin{cases} 5 & \text{if } F_s \geq 0.25 \\ 5 + \frac{F_s - 0.02}{0.25 - 0.02} & \text{if } 0.02 < F_s < 0.25 \\ 90 & \text{if } F_s \leq 0.02 \end{cases} \tag{7}$$

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 two-layer MLP with a sigmoid function in the output layer. 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 decided if the pixel is foreground or background.

Tracking

Tracking can be described as a way to identify the same object in different frames of the video. Usually, the object is identified in a frame and a representation model of that object is created. In the subsequent frames, the algorithm tries to identify the area of the frame that best approximates the model created and some algorithms even perform a model update throughout the video. 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 and it is important to take into account as 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.

Tracking algorithms can be split into two main groups: single target trackers and multiple object trackers. One approach to track a single target is by using correlation filters like the Kernelized Correlation Filter (KCF) proposed in [10]. 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.

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 O of detected objects using some kind of object detector, one can try to associate each track T_i to a detected object O_j. To do this, we have to use a similarity measure to compare each object from the current frame to the ones from the previous frame. After that, the Hungarian algorithm is used to maximize the similarity value for each association between the tracks and the objects. This is the approach used in [3]. After the objects are detected, a Kalman filter [13] 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:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{8}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{9}$$

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.

The Kalman filter performs well in a linear motion scenario, but in maneuvering targets others approaches may give better estimations. In [28], 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

$$x(k) = \sum_{i=1}^{M} \mu_i(k)\hat{x}_i(k) \tag{10}$$

where M is the number of models being used. As the authors of [28] 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.

Trying to solve the tracking problem in extremely crowded scenarios, the authors of [7] 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:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} f(\mathbf{x}) = & \overbrace{c_a^T \mathbf{x}}^{\text{appearance}} + \overbrace{\zeta c_m^T \mathbf{x}}^{\text{target motion}} + \overbrace{\eta c_{nm}^T \mathbf{x}}^{\text{neighbour motion}} \\ & + \underbrace{\mathbf{x}^T C_{sp} \mathbf{x}}_{\text{spatial proximity}} + \underbrace{\mathbf{x}^T C_g \mathbf{x}}_{\text{grouping}} \end{aligned} \tag{11}$$

where x is a matrix encoding the possible combinations between the targets and the candidate locations. Are also considered constraints 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 and is one of the distinctive parts of the approach to try and solve the tracking in high-density crowd scenes where target interactions are used in order to optimize the tracking for all targets simultaneously. The fourth component is a spatial proximity constraint to stop the tracker from selecting locations that are very close to each other unless the appearance can be easily distinguished. 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 one. 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.

3 Proposed Solution

As stated previously, our goal is to track fish inside tanks. To do this, a recording device must be used to gather video recordings that will be processed later on. Our environment can be characterized as follows:

- A single fixed camera recording is positioned outside the tanks filming from the side
- Not all tanks are the same: our solution must be able to track accurately in scenarios portraying tanks with different width, depth and different types of environments, such as coral reefs, Pacific, Atlantic and Indian oceans

- The fish being tracked are very diverse: they have different shapes, colors and swimming patterns
- The camera's field of view might not capture the entire width of the tanks meaning that the targets might swim to areas not being captured and can no longer be tracked thereafter

Our environments can be seen in Figure 1.

To tackle the problem at hand, our approach will be made of three modules: pre-processing, object detection and tracking. In the first module, the input video will be modified by techniques to improve the quality of the videos taking into account that the colors of underwater footage differ from the true colors of the captured elements and can make them less differentiated. In the detection module, the moving objects are identified and some some features are extracted. The tracking module relates the existing tracks with the newly detected objects given by the detection module. Our system architecture is represented in Figure 2.

3.1 Pre-Processing

Before the frames are used to detect the objects, different types of histogram equalization will be tested to evaluate their influence in the performance of both the object detection and the tracking. Some of the techniques being tested include the ones reviewed previously like CLAHE [15] and UCM [12], remembering that the first one includes transforming the image to the CIELAB color space before applying the histogram equalization to the lightness channel and then transforming it back to RGB.

3.2 Object detection

This component, for each post-processed frame F_k at time k, will try to identify the moving objects. This is made by detecting the presence of an object for each individual pixel. Although in some cases a pixel representing the background is defined by the same pixel color every time, that is not always true and a good example of that in our environment is the coral reef that moves with the currents. In this case, even though there is movement occurring, the coral is considered to be part of the background and shouldn't be detected as an object.

The method chosen to detect objects is the same as the work of [1], where the SubSENSE algorithm [24] and the flux tensor [26] are used to train a convolutional neural network to perform background subtraction since this approach performed very well. To train the CNN, we need to use a dataset with the frames from the videos and the correspondent background images. So the first thing we need to do is generate the background. First, the SubSENSE is used and it outputs a binary image with the foreground objects. Then, for each pixel, a background pixel library BL stores the last 90 pixel values that were classified as background by the SubSENSE where the oldest are first replaced. The background is generated by averaging the last bm pixel values of the library. This variable bm is computed using a flux tensor that acts as a motion detector

Fig. 1: Different types of environments in Oceanário de Lisboa

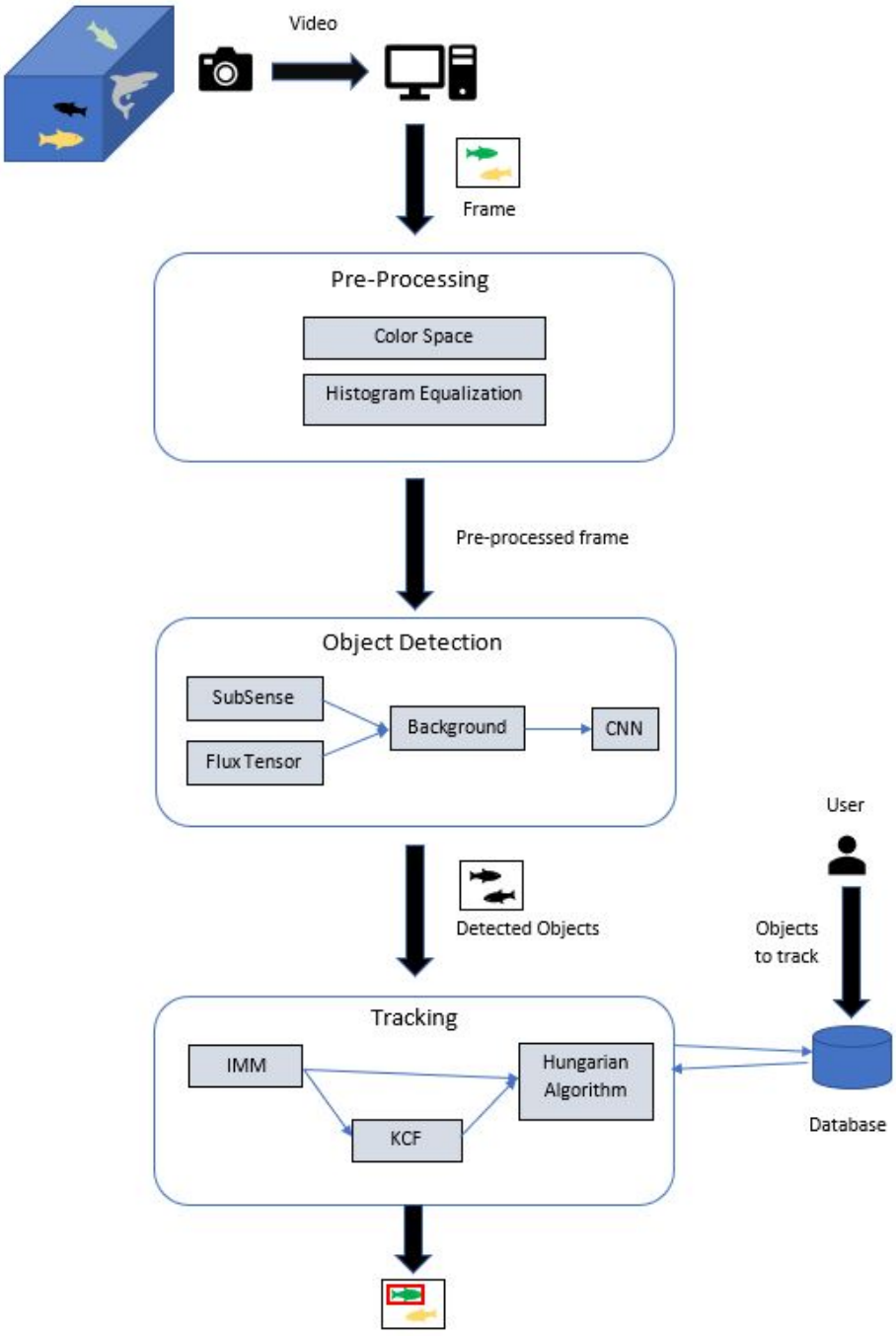

Fig. 2: System architecture

outputting a binary image where pixels are classified as moving or stationary. The percentage of moving pixels F_s is computed for the current frame and this value is used to increase or decrease bm according to (7). There is also a low pass filter applied to F_s:

$$F_s(t) = \alpha \cdot F_s(t) + (1 - \alpha) \cdot F_s(t - 1) \tag{12}$$

where

$$\alpha = \begin{cases} 0.2 & \text{if } F_s(t) < F_s(t-1) \\ 0.01 & \text{otherwise} \end{cases} \tag{13}$$

After this we can now generate the background using the last bm values for each pixel.

To train the network, we need to supply it with a patch from the image, the corresponding patch from the generated background and the ground truth. The RGB images are resized to 240×320, the intensities re-scaled to [0,1] and zero-padding is applied. Each patch is 37×37 and mean subtraction on the patches is performed before training without dividing by the standard deviation. The network has 3 convolutional layers with Rectified Linear Unit after each one and a 2-layer Multi Layer Perceptron followed by a Sigmoid Function. The input of the CNN is $37 \times 37 \times 6$ (three channels for the image patch and correspondent background patch). The convolutional layers apply a convolution of size 5×5, with 3 padding and a pooling filter of size 2×2. The size of the first layer of the MLP is 2048 and of the final layer is 1369 (37×37). The loss function, Binary Cross Entropy (BCE), is given by:

$$BCE(x, y) = -(x * \log y + (1 - x) * \log(1 - y)) \tag{14}$$

With the output of the CNN, we apply a spatial-median filtering with a size of 9×9 to remove outliers and after that, a threshold $R = 0.3$ is used to classify the pixels as background or foreground according to

$$g(z; R) = \begin{cases} 1 & \text{if } z > R \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Now that we have a binary image we need to extract the objects from it. To do this we have to apply a connected-components labelling algorithm. This algorithm will group together foreground pixels that are neighbours. We finally know the position of the objects and can compute their locations, bounding boxes, color histograms, among other things that can later be used in the tracking module, discussed in the next chapter.

3.3 Tracking

In this module we attempt identify an object throughout the video. A track is a collection of detected objects in the previous frames, each object can only belong to one track and at each frame F_k at most one object can be matched with each

track when there are no occlusions. When occlusions happen, a detected object may include two or more fish so more than one track can match a part of the same detected object. Given that objects disappear from the scene, sometimes tracks to be deleted when no match has been made for a certain period of time. A track can't be deleted the first time there is no matched object as the object might be occluded by other objects or elements of the scene or the object might not have been detected by the previous module.

Our system will combine position prediction and visual discrimination together to track the objects. For the position, an Interactive Multiple Model will be used following the work of [28]. The following three models will be used: constant velocity model, constant acceleration model and constant turn model. For the constant velocity model, a Kalman filter can be used since its the best estimator for linear motion and is computationally efficient. For the other two a Particle filter will be the choice although other variations of filters might be tested to understand which ones work better in our environment. For each currently tracked object, a position is estimated and then, after matching the objects with the tracks, the measurements are included in the estimation of the filters. Regarding the visual discrimination, the scale adaptive KCF (SAKCF) will be used as some of the tanks are very long and sometimes the fish are close to the glass and other times more distant. The scaling pool S has to include smaller scale variations, bigger scale and the original scale (the values of S are to be multiplied by the original window size). Different scales will be tested in order to get the least amount of values that give a good performance. Also a rotation pool R will be considered to include possible rotations of the object and work in a similar fashion to the scaling approach. In terms of image features to use with KCF, histogram of oriented gradients (HOGs) will be used but other features might be used, like raw pixel values, either replacing HOGs or run together with them, to test get the best tracking performance. Given that KCF tries to find the best matching part of the image in a certain position, we can use the positions from the detected objects as candidate positions and we can ignore objects that are very far away from the predicted position of the IMM for the track as it is very unlikely that a fish, from one frame to the next, changes it's position completely. Given that the object detector might fail to detect an object in the current frame, new candidate position can be considered according to the predicted position or the previous position of the object, in the cases where there are no detected objects near those positions. After that, an Hungarian algorithm is performed where each candidate position has a value for each object that includes the similarity in terms of positional proximity to the predicted position and visual similarity given by the KCF filter. As previously stated, sometimes objects are absent from the frame and we don't want to associate the corresponding track to any other object or target location so when all possible associations have a very low similarity score, no match should be made in the current frame. After some time of not being associated with any targets, the objects are considered to have disappeared from the scene and the corresponding track eliminated.

3.4 Interface

Additionally, an interface will be developed that will allow the user to make some corrections when the automatic tracking fails for more than a pre-defined number of frames, to allow the program to recover and only in case of total failure allow the user to make adjustments. In the end, the program will show the final result.

4 Evaluation

In our system, we combine object detection with tracking, in order to track fishes in closed tanks. Given that we want to evaluate tracking performance, we should use the same metrics used in object tracking competitions such as Visual Object Tracking (VOT) [16]. This competition is split into some different types of tracking, but since our system is supposed to be a long-term tracker, that is where we should focus. In the 2019 edition, the VOT-LT performed the evaluation of the submitted trackers based on the metrics recently defined in [20]. The authors defined tracking precision, tracking recall and tracking F-score, inspired in the the same metrics from other areas. To compute them, first we need to look at the tracker output. A tracker predicts the next position of the target with a certainty score. Looking at the ground-truth, in the case that the target is absent from the frame, the ground-truth set is empty, $G_t = \emptyset$ and when the tracker can't predict an output or the certainty score is lower than the threshold τ_θ, the output is $A_t(\tau_\theta) = \emptyset$. To compare the ground-truth with the prediction, an intersection over union (also known as Jaccard Index) should be computed between the two, $\Omega(A_t(\tau_\theta), G_t)$. It is given by the overlapping area of the two bounding boxes divided by the union of the two areas, resulting in a value between 0 and 1 (higher values mean more similarity between the bounding boxes). The tracking precision and tracking recall are given by

$$Pr(\tau_\theta) = \frac{1}{N_p} \sum_{t \in \{t : A_t(\tau_\theta) \neq \emptyset\}} \Omega(A_t(\tau_\theta), G_t) \tag{16}$$

$$Re(\tau_\theta) = \frac{1}{N_g} \sum_{t \in \{t : G_t \neq \emptyset\}} \Omega(A_t(\tau_\theta), G_t) \tag{17}$$

With Pr and Re defined, we can compute the tracking F1-score

$$F(\tau_\theta) = \frac{2Pr(\tau_\theta)Re(\tau_\theta)}{Pr(\tau_\theta) + Re(\tau_\theta)} \tag{18}$$

The F1-score is computed with multiple thresholds τ_θ and the highest score is chosen. To run this evaluation, we need a dataset with hand-labeled ground-truth bounding boxes of the fish's trajectories. Our dataset will aggregate videos from multiple tanks that present different types of challenges. The tanks vary in depth and width and the fish in them vary in amount, species, size, among other

things. VOT includes a dataset of a fish scene similar to one of ours, however it's a video sequence for the short-term competition. This means that the videos are short and the target is never absent from the frames. In spite of that, we can still use them to expand the testing set, with the plus that the annotations have already been made for those videos.

5 Schedule

The planned schedule for our work is presented in Figure 3. First, some preparations regarding the datasets and the CNN will be made as the rest of the work depends on them. Then the proposed system will be developed. As the evaluation method depends on the last module, we can only start the evaluation after it is available. At the same time, we will try to improve our system by tuning the parameters and eventually adding, removing or replacing some of the components. The final part is writing the thesis dissertation, although throughout the previous stages, some parts of the dissertation may be written.

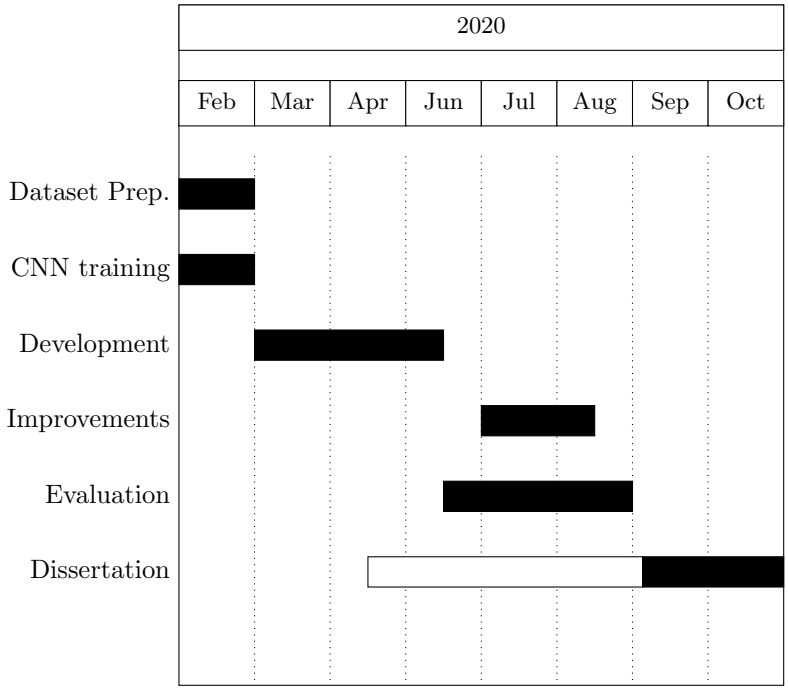

Fig. 3: Schedule

6 Conclusions

The goal of our work is to create a system that can robustly track fish swimming inside tanks. Some of the relevant challenges of this specific task were discussed. In order to achieve that goal, first the state-of-the-art was reviewed, and split into three categories: pre-processing, object detection and tracking. The first category includes techniques to improve the image quality, the second one to detect moving objects in the videos by trying to separate the background from the foreground (objects of interest) and in the final category, tracking approaches were reviewed, where some of the covered topics included data association, correlation filters and position predictions. Then, a system was proposed that combines some of the reviewed techniques for each module, adapting them to our problem, in order to robustly track the fish. To evaluate the system, an approach based on the position of the predicted bounding-boxes (or their absence when the fish is occluded) will be used. In the next phase, during the development, some changes may be required to improve the system, and some techniques may be added/removed.

References

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. 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.
6. J. Castelo. Video based live tracking of fishes in tanks. Master's thesis, Instituto Superior Técnico, 2019.
7. 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.
8. 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.
9. 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.

10. 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.
11. 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.
12. 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.
13. R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.
14. 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.
15. 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.
16. M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, 2016.
17. F. Kristensen, P. Nilsson, and V. Öwall. Background segmentation beyond rgb. In P. J. Narayanan, Shree K. Nayar, and Heung-Yeung Shum, editors, *Computer Vision – ACCV 2006*, pages 602–612, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
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. A. Lukežič, L. Zajc, T. Vojíř, J. Matas, and M. Kristan. Now you see me: evaluating performance in long-term visual tracking. *arXiv preprint arXiv:1804.07056*, 2018.
21. T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Proceedings of 12th International Conference on Pattern Recognition*, volume 1, pages 582–585, 1994.
22. S. Palazzo, I. Kavasidis, and C. Spampinato. Covariance based modeling of underwater scenes for fish detection. In *2013 IEEE International Conference on Image Processing*, pages 1481–1485, 2013.
23. 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.
24. 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.
25. 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.
26. 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.
27. 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.
28. 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.