Detecting and Counting Oysters

Behzad Sadrfaridpour\textsuperscript{1}, Yiannis Aloimonos\textsuperscript{1}, Miao Yu\textsuperscript{2}, Yang Tao\textsuperscript{3}, and Donald Webster\textsuperscript{4}

Abstract—Oysters are an essential species in the Chesapeake Bay living ecosystem. Oysters are filter feeders and considered the vacuum cleaners of the Chesapeake Bay that can considerably improve the Bay’s water quality. Many oyster restoration programs have been initiated in the past decades and continued to date. Advancements in robotics and artificial intelligence have opened new opportunities for aquaculture. Drone-like ROVs with high maneuverability are getting more affordable and, if equipped with proper sensory devices, can monitor the oysters. This work presents our efforts for videography of the Chesapeake bay bottom using an ROV, constructing a database of oysters, implementing Mask R-CNN for detecting oysters, and counting their number in a video by tracking them.

I. INTRODUCTION

Oysters are an essential species in marine ecosystems. They are filter feeders that clean the surrounding water and create habitat for the other species. However, oysters habitats have undergone significant drops both on national and global levels \cite{7}. There are ongoing efforts to restore oysters habitats across the United States \cite{7}, \cite{9}, \cite{21}, \cite{29}. A significant challenge is providing adequate and effective means for monitoring their progress. To improve the consistency and extensibility of monitoring of oyster habitat restoration projects, the authors in \cite{7} used an expert group in designing and monitoring oyster reef projects to standardize monitoring metrics, units, and performance criteria for evaluation. They recommended monitoring a set of universal parameters including “reef areal dimensions, reef height, oyster density, and oyster size-frequency distribution” and environmental variables including water salinity, temperature, and dissolved oxygen \cite{7}. Other oyster restoration projects also report the monitoring of these metrics and variables in part \cite{9}, \cite{21}, \cite{29}.

Currently, the measurement of the number of oysters and their physical dimensions for sampling is performed manually and, therefore, is restricted to small amounts, e.g., 100 oysters per sample site \cite{21}. Moreover, these measurements are not regularly available for small-size aqua/oyster farms. With advancements in robotics and artificial intelligence, there is a promising potential to improve the monitoring process of oyster habitat restoration. Drone-like ROVs (Remotely Operated underwater Vehicles) with high maneuverability are getting more affordable and, with proper sensory devices, can be used for sampling. Moreover, computer vision algorithms can be developed to detect oysters and calculate their physical dimensions and properties. More specifically, convolutional neural networks (CNNs) have been widely used for object detection, and we use them as state-of-the-art for oyster detection. To get the number of oysters in a video, we count the identified oysters. However, to prevent recounting, the detected oysters need to be tracked in consecutive image frames. We use the KCF tracker from OpenCV for tracking the already identified oysters.

Our goal is to step toward monitoring and counting but in a way that would also benefit the oyster harvesting problem. We use an ROV for data collection and implement object detection and instance segmentation algorithms that can be later used for automated harvesting of oysters. However, we do not address oyster picking and harvesting in this paper. Here, we explain how we collected the videos of oysters in the Chesapeake bay and used them for training the oyster detection CNNs. Next, we explain how we tracked the detected oysters to count the oyster in a video. The organization of the rest of the paper is as follows. Section II provides a related work for bottom habitat mapping and common object detection and segmentation algorithms used in precision agriculture. Section III discusses the object detection and counting algorithm. Section IV provides the results of our implementation and experiments.

II. RELATED WORK

Most of the related work for seafloor bottom mapping in subtidal waters is done using remote sonar sensing techniques followed by in situ sampling or underwater videography for ground-truthing \cite{24}. However, sonar data sets have low resolutions, and obtaining data sets with high-quality data is expensive. Moreover, their process, interpretation, and visualization demand substantial effort. Some works \cite{14} used the georeferenced optical data as the primary mapping tool. A significant problem of visual data is its interpretation \cite{5}. Compared to the physical data, the visual data (i.e., photographs) are limited in terms of measuring the physical parameters (such as size) of samples \cite{5}. However, with advancements in underwater robotics and image processing, the capabilities in obtaining and using optical data have much increased. For instance, recently, an approach for autonomous coral identification and counting \cite{23} and underwater 3D semantic mapping \cite{22} have presented. A similar approach can be adopted for the problem of oyster habitat mapping.
Generally, works for detection and tracking are categorized as Multi-Object Tracking (MOT), an active research topic in computer vision focusing on identifying and tracking objects of specific classes such as cars, animals, and pedestrians videos [10]. Usually, the objects are animated and have arbitrary trajectories, while here, the objects (oysters) are in-animated, and the camera is moving. Nonetheless, we can still employ MOT practices for our task. We employ tracking-by-detection, which includes three steps [25]: detection of objects, tracking of the previously detected objects, and association of the tracked objects with the detected objects.

Deep learning methods for object detection can be categorized as one-stage and two-stage algorithms. One-stage algorithms such as YOLO [26] SSD [20] and Fast R-CNN [12] solve the object detection using a single network [26] aiming for faster inference computation time while two-stage algorithms such as Faster-RCNN [27] solve the problem in two stages, one for proposing candidate bounding boxes and one for later classifying them as foreground classes and background, aiming for higher inference precision. Mask-RCNN [15] is an extension of Faster R-CNN, which also segments the foreground classes using a CNN at the end parallel to the bounding boxing recognition branch. Mask R-CNN demonstrated that it has the highest precision for common datasets [31]. In this work, the segmentation helps with generating the size maps of the oysters. Therefore, we used Mask R-CNN [15] for detecting oysters.

Various approaches have been applied for tracking and feature extraction [25]. These include deep learning methods, correlation filters (CF), auto-encoders, spatial attention, motion extraction, and feature pyramids. CF-based trackers are among the computationally efficient yet robust trackers [16]. We tried different trackers and chose KCF (Kernelized Correlation Filters) tracker [16] due to its robustness and efficiency.

III. METHODS

Three main steps required for counting the numbers of oysters in a video are detection, tracking, and association. We used Mask R-CNN for detection, KCF tracker for tracking, and pairwise matching of the detection and tracking bounding boxes based on Intersection over Union (IoU) for the association. First, we detect the oysters in the current frame and specify their bounding boxes as described in Section III-A. Next, we update the tracking bounding boxes from the previous frames described in Section III-B. The final stage is pairwise matching between the newly detected bounding boxes and the updated tracking bounding boxes and associating new trackers for the non-matched oysters (Section III-C). Figure 1 provides an overview of these steps.

A. Object Detection Network Structure

For detecting the oysters, it is most suitable to choose the object detection architectures such as R-CNN (Regions with CNN features) [13], Fast R-CNN [12], Faster R-CNN [27] and Mask R-CNN [15]. The input of these networks is the image, and the outputs are objects and their corresponding bounding boxes or masks of their instance segmentation in the image. R-CNN uses selective search algorithm to generate around 2k Region of Interests (RoIs), wraps them to a fixed square size, and finally runs them through a CNN. CNN is trained for classifying the images and predicting the boxing offset for the original RoIs. Generating the RoIs is computationally expensive and is the bottleneck of the R-CNN method. Fast R-CNN is an improvement of R-CNN. Similar to R-CNN, it calculates the RoIs first using the selective search algorithm. However, it first runs the entire image on a backbone CNN to get a high-resolution feature map corresponding to the whole image. It generates RoIs on the feature map of the image rather than the original image. It then uses pooling to reshape and unify the size of RoIs. The output of Fast R-CNN is similar to R-CNN. Faster R-CNN improves Fast R-CNN by using a separate network, Region Proposal Network (RPN), for proposing the RoIs from the output of the backbone CNN, feature map(s) of the image. It uses RoI pooling to reshape the proposals and then classify and predict the bounding boxes offset similar to Fast R-CNN. Mask R-CNN works similar to Faster R-CNN but extends it with another network for segmenting the RoIs. Our goal is to be able to generate the size and frequency map of the Chesapeake Bay oysters. Therefore, we used Mask R-CNN that outputs instance segmentation of objects in addition to bounding boxes and classes. Figure 2 shows the structure of this network. Different CNNs such as VGGNet, ResNet and with different depths are being used as the backbone of Mask R-CNN. Moreover, Feature Pyramid Networks (FPN) [17] are also being used as an extension to the backbone network. FPN builds a pyramid of feature maps with different sizes by making two series of layers (bottom-up and top-down) with lateral connections. Each feature map then goes through the rest of the network, i.e., RPN, classifier, bounding box regressor, and segmentation. The final segmentation is made by merging the masks outputted for all feature maps. The modified versions of ResNets that extracted features from the final convolutional layer of the 4-th stage, called C4, were used in the original Faster-RCNN implementation. Another choice for the backbone network is ResNet with dilations in 5-th stage [11], which is called DC5. In this work, we used ResNet50-C4, ResNet50-DC5, ResNet50-FPN, ResNet101-FPN, ResNet101-C4, and ResNet101-FPN as the backbone of the Mask R-CNN and compared the results.

B. Tracking

The KCF tracker achieves efficient tracking by generating a matrix of cyclic shifts of the sample vector $x =$
[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^n$. The cyclic shift of \( \mathbf{x} \) is defined as
\[ \mathbf{P}_i = [\mathbf{x}_n, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}] \]. Matrix of all cyclic shifts of \( \mathbf{x} \), the circulant matrix, is defined as
\[ \mathbf{X} = \mathbf{C}(\mathbf{x}) = \begin{bmatrix}
\mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \\
\mathbf{x}_n & \mathbf{x}_1 & \ldots & \mathbf{x}_{n-1} \\
\mathbf{x}_{n-1} & \mathbf{x}_n & \ldots & \mathbf{x}_{n-2} \\
\vdots & \vdots & \ddots & \vdots \\
\mathbf{x}_2 & \mathbf{x}_3 & \ldots & \mathbf{x}_1
\end{bmatrix} \].

(1)

The KCF tracker uses the cyclic shift method for \((M \times N)\) image block \( \mathbf{x} \) to train a filter \( f(\mathbf{x}) = \omega \phi(\mathbf{x}) \), where \( \phi(\mathbf{x}) \) is the mapping Fourier space. The training samples for \( \mathbf{x} \) are all of the cyclic shifts \( \mathbf{P}_i \) where \( i \in \{0, \ldots, M-1\} \times \{0, \ldots, N-1\} \). The corresponding score \( y_i \in [0,1] \) for \( \mathbf{P}_i \) is generated by a Gaussian function based on shift distance. The classifier for detection of the tracker is trained by minimizing the regressor error as [19]:
\[ w = \arg \min_w \sum_i (\omega \phi(\mathbf{x}) - y_i)^2 = \lambda \parallel w \parallel^2, \]

(2)

where \( \lambda \geq 0 \) is the regularization parameter for the model’s simplicity. The hypothesis is trained effectively by using fast Fourier transform. In the Fourier domain, instead of a single grayscale channel feature input, the KCF algorithm can use multi-channel histogram of oriented gradients (HOG) features as input, simply by summing them over [16].

C. Association

As shown in Fig. 1, for each frame in the video sequence, the detected bounding boxes (of oysters) are compared with the tracked bounding boxes to add new trackers, update trackers or remove trackers. The bounding boxes are matched in pairs using their IoU values. For two bounding boxes, IoU is defined as the ratio of area of their overlap over the area of their union. We define the IoU overlap threshold of 0.2 for matching the boxes. If the IoU of a detected box with a tracking box is greater than the threshold, we match them. Moreover, we add new tracking boxes for the non-matched detected boxes and update the matched tracking boxes. We keep track of any unmatched tracking box and remove it after a certain number of mismatches. Each tracker is assigned a unique number, and the output frame is the updated bounding boxes. Thus the total number of oysters is the number of trackers assigned for the video sequence.

IV. EXPERIMENTS AND RESULTS

This section describes our underwater image acquisition, implementation and training of the Mask-RCNN network, and, finally, results for detection and tracking.

A. Image Acquisition

We conducted a preliminary study on the underwater machine vision of oysters in the Chesapeake Bay, as shown in Fig 3a. We mounted an led light and a waterproof camera on a long pipe and took underwater photos. Underwater illumination is a challenging problem and had a high impact on the quality of the collected video sequence. Next, we used a Remotely Operated underwater Vehicle (ROV) for taking underwater videos. Figure 3(b) shows our underwater videography setup. The ROV is BlueRobotics BlueROV2 [1], with two horizontal thrusters and four vertical thrusters providing six degrees of freedom, three translational (heave, sway, and surge) and three rotational (pitch, roll, and yaw). It is connected via a tether to a remote machine located on the ground or boat. An open-source software, QGroundControl (QGC) [3], is installed on this machine as the graphic user interface (GUI) of ROV for sending commands to ROV and receiving camera and other sensory information on the ground station machine (remote machine). The ROV can be controlled autonomously or manually. Here, we control it manually by sending control commands to QGC using a wireless controller. A camera and two led lights are mounted in front of the ROV. The camera feed streams to the remote machine, and the operator uses it for the underwater navigation of the ROV.

We equipped the ROV with an additional camera, a GoPro HERO7 Black, and led lights to view the seafloor. We set up this camera to store the videos locally and stream to the remote machine using FFmpeg software [2] installed on the local machine. The images taken from the Chesapeake Bay...
Fig. 4: Images of an arbitrary location of the bay bottom but with different altitudes as the ROV moves toward the bottom from (a) to (d). Due to the high turbidity of the water, the bottom is only visible within around three feet.

Fig. 5: Sample images of (a) washed oysters and shells after dredging and (b) oysters and shells underwater have high turbidity, and the bottom is not visible without external lights. With our lighting setup, we could only see oysters within around three feet from the bottom, and thus we take images within that range. Figure 4 shows the images of the bay bottom for different heights from the Bay bottom.

B. Oyster data set and Annotation

Images from the bay bottom recorded videos using our customized ROV were used for constructing the oyster data set. There are two main challenges for ground-truthing the live oysters in the images. First, both live oysters and empty shells are lying down in the bay bottom. Live oysters have two valves (or shells) that are slightly open enough to ventilate or feed. Animal orientation is a significant factor for detecting live oysters. If animals are cup-side up, this orientation will mask whether they have two valves or one and how much the gaps are between two valves. Second, the oysters and shells are heavily covered in sediment, and the animal is heavily fouled with epibionts like bryozou. Therefore, the actual color texture of oysters is not visible (Figure 5). To overcome these challenges (without manipulating animals, which is out of scope of this article), we rely on two things. First, since we have the video data we can use it to analysis an image for ground truthing when necessary. Second, we assume that the object is an live oyster wherever its orientation is cup-side up.

To make the annotation process more convenient, we annotate the data set in two phases using the image annotation tool labelme [30] and another tool we made for converting to masks to labelme software format. First, we selected fifty sample images and generated mask images of the oysters. We used a pre-trained Mask R-CNN (described in Section III-A) as an initial object detector network and trained it using these fifty images. Next, we selected another set of images and ran them through our initial object detector to get preliminary predictions of oysters. We used our tool for converting the predictions of the trained network to labelme file format. We reviewed and adjusted the generated annotations for making the ground truth mask images of live oysters. We combined all the images and annotated them for model training of the object detection networks. About sixty-five percent of the data set was used for training, and the rest was used for evaluating the performance of the trained networks. Figure 6 shows a sample annotation of live oysters.

C. Implementation and Pre-Trained Models

Different implementations of Mask R-CNN are already available online. We used Detectron2 [31] that can be used as a library; trains fast, and has COCO data set [18] pre-trained models available. COCO is one of the benchmark data
E. Evaluation of the Trained Mask R-CNN

ResNet101-FPN, and with six different backbones of ResNet50-C4, ResNet50-FPN, ResNet101-FPN, ResNet101-C4, and ResNet101-DC5, trained our models for object detection and segmentation algorithms [18]. For a detected object (here oyster), IoU is the ratio of the predicted object’s overlap area and the annotated ground truth over their union area. It is used for detecting the number of true positives (TP), false positives (FP), and false negatives (FN). TP is the number of objects detected positive correctly, FP is the number of objects detected incorrectly, and FN is the number of missed objects. For example, if the IoU threshold is set to 50 percent, the predicted object is a true positive if IoU is greater than 0.5; otherwise, it is a false positive. If the object is not predicted, it is a false negative. Precision (P) is the ratio of TP over the number of the predicted objects, and recall is the ratio (R) of the number of TP over the number of ground truth objects. For an IoU threshold, if we plot the precision over recall curve, the area under the curve is the \( AP \) for that threshold. In this work, we report mask \( AP \) values to evaluate our trained model using the predicted masks.

Table I compares the different \( AP \) values for the oyster segmentation of our trained models for the validation set. \( AP_{50} \) and \( AP_{75} \) are the average precision calculated for IoU thresholds of 0.50 and 0.75, respectively. \( AP \) is the mean of \( \{ AP_{50}, AP_{55}, \ldots, AP_{95} \} \). The best \( AP \) and \( AP_{75} \) were achieved for the network with ResNet50-C4 backbone with the values of 55.9 and 64.1, respectively. The best \( AP_{50} \) was achieved for ResNet50-DC5 backbone with a value of 81.8.

As mentioned in IV-B, there are two challenges for detecting live oysters, (a) animal orientation and (b) sediments on oysters. These two challenge heavily impact the performance of the Mask R-CNN network. For annotating the ground truth, we used the video sequence to inspect the previous and next images when it was challenging to distinguish live oysters from empty shells. If it is challenging for the annotator to distinguish between the live oysters and the reefs on the bottom, the same problem applies to the network which learns from those annotations. Nonetheless, looking at the inferred images of the oysters, the results seem promising. Our ultimate goal is to make an underwater robot capable of picking up (harvesting) oysters. Therefore, as long as the network can detect most of the live oysters, we can overlook the false positives. Figure 7 shows samples of inferred oysters. Figure 7b shows instance segmentation of Fig. 7a. This image is taken from a relatively far distance, and due to turbidity, it is relatively more blurry. There are more false positives in the inferred image compared to the other images in Fig. 7. However, the number of missed detected oysters (false negatives) is small (two oysters in the top left). Figure 7d showcases a moderately high performance of the network. There is no false positive in this image but there is one false negative (the oyster beneath the oysters on the right). Figure 7f demonstrates a moderately average performance of the network. There are three false positives in this image (from the center to the right), and there is no false negative. Figure 7f illustrates a moderately high performance of the network. There is no false positive in this image but there one false negative (the oyster beneath the oysters on the right). Figure 7h depicts a relatively high performance of the network. There are three false positives in this image (right bottom), and there is no false negative.

| Backbone     | \( AP \) | \( AP_{50} \) | \( AP_{75} \) |
|--------------|---------|-------------|-------------|
| ResNet50-FPN | 53.2    | 77.5        | 59.2        |
| ResNet50-DC5 | 55.0    | 81.8        | 63.3        |
| ResNet50-C4  | 55.9    | 80.3        | 64.1        |
| ResNet101-FPN| 53.2    | 77.5        | 59.6        |
| ResNet101-DC5| 53.4    | 78.8        | 60.1        |
| ResNet101-C4 | 51.6    | 76.8        | 58.7        |

TABLE I: Instance segmentation mask \( AP \) values

Fig. 8: Stitched images of videos (generated via ODM [6]).

sets for object detection and segmentation and has around 330k images (with more than 200k labeled) of 80 object categories. Since we have a small data set, we leveraged pre-trained models for transfer learning. We used the underlying convolution kernels of the COCO pre-trained networks for our network training, which capture low-level features common in most objects such as edges and angles and only trained for the high-level features special to the oysters. As discussed in Section IV-B, we used 1290 instances of oysters, 827 instances for training, and the rest for validation.

D. Detection Results

All the training and evaluation have been executed on a single machine with CPU Intel Xeon E5-2680 2.4 GHz, 2 NVIDIA Tesla P100 GPUs, and 128GB memory. We trained our models for object detection and segmentation with six different backbones of ResNet50-C4, ResNet50-DC5, ResNet50-FPN, ResNet101-FPN, ResNet101-C4, and ResNet101-DC5.

E. Evaluation of the Trained Mask R-CNN

Average Precision (\( AP \)) over different IoU (Intersection of Union) thresholds are the common metrics used for object detection and segmentation algorithms [18]. For a detected object (here oyster), IoU is the ratio of the predicted object’s
Fig. 9: The first and last frames of the sample videos. For each tracker bounding box, the number on the top row shows the tracker identification number. If the detection and tracker matches, the number on the second row shows the oyster inference percentage.

|                | Tracking-by-detection | Manual counting |
|----------------|------------------------|-----------------|
| sample 1       | 27                     | 21              |
| sample 2       | 25                     | 20              |
| sample 3       | 118                    | 94              |
| sample 4       | 215                    | 176             |

TABLE II: Number of oysters in the sample videos

F. Tracking and Counting Implementation

We used OpenCV [4] for implementing KCF tracker. We picked two sample videos to demonstrate our oyster counting method. To better demonstrate the results, we picked two short videos with 1000 and 300 frames (around 16 and 5 seconds for 59 fps) and used OpenDroneMap (ODM) [6] to extract a single stitched image for each of these videos. Note that generating such images takes a much longer time than counting the oyster using our method. Figure 8 shows the stitched images. Figure 9 shows the first and last frames of the sample videos as well as the results of the proposed method. We count the oysters in a sample video ourselves manually and compare them with the results from implementing our method, as shown in Table II. We also repeat these for another two sample video sequences without generating their stitched image. The results show an average accuracy of 79.8%. There are three main sources of error: (i) miss-classification of empty shells as oysters, (ii) recounting of oysters, which was reduced by tracking the unmatched tracking bounding boxes, and (iii) the oyster instances that the detection network missed.

V. CONCLUSION

Oysters are an essential species in marine ecosystems, and restoration of oyster habitats is vital. Our goal is to help to monitor oysters. In this work, we demonstrated how we customized a drone-like ROV, BlueRobotics BlueROV2, with a camera, GoPro HERO7 Black, for videography of the Chesapeake Bay bottom. Moreover, we annotated the visual data captured from the bay bottom to make an oyster data set. We chose the state-of-the-art CNN for target tracking and instance segmentation, Mask R-CNN, and employed detectron2 for training and evaluation of the oyster data set. We compared the results for different backbones in the network structure. Our trained networks’ prediction results showed an Average Precision (AP50) of 81.8 for our oyster data set. We also combined the detection CNN with an object tracker for counting the number of oysters in videos. Target detection and instance segmentation of oysters is the first step towards making a robotic tool which environment-friendly and can harvest oysters more efficiently. The code is available online at GitHub/bsadr [28].

ACKNOWLEDGMENT

The authors wish to thank Mr. Bobby Leonard for providing his boat and giving us tours in his oyster lease for taking videos in the Chesapeake Bay. This work was supported in part by the Maryland Robotics Center postdoctoral fellowship program. This work was supported in part by USDA NIFA sustainable agriculture system program under award number 20206801231805.
REFERENCES

[1] BlueroV2. https://bluerobotics.com/. Accessed: 2020-02-22.
[2] Ffmpeg. https://www.ffmpeg.org. Accessed: 2020-02-22.
[3] Qgroundcontrol. http://qgroundcontrol.com. Accessed: 2020-02-22.
[4] OpenCV - open computer vision library. https://opencv.org/. 2020.
[5] John T Anderson, D Van Holliday, Rudy Kloser, Dave G Reid, and Yvan Simard. Acoustic seabed classification: current practice and future directions. ICES Journal of Marine Science, 65(6):1004–1011, 2008.
[6] OpenDroneMap Authors. Odm - a command line toolkit to generate maps, point clouds, 3d models and demos from drone, balloon or kite images. https://github.com/OpenDroneMap/ODM, 2020.
[7] Lesley P Baggett, Sean P Powers, Robert D Brumbaugh, Loren D Coen, Bryan M DeAngelis, Jennifer K Greene, Boze T Hancock, Summer M Morlock, Brian L Allen, Denise L Breitburg, et al. Guidelines for evaluating performance of oyster habitat restoration. Restoration Ecology, 23(6):737–745, 2015.
[8] Michael W Beck, Robert D Brumbaugh, Laura Airoldi, Alvar Carraza, Loren D Coen, Christine Crawford, Omar Defeo, Graham J Edgar, Boze Hancock, Matthew C Kay, et al. Oyster reefs at risk and recommendations for conservation, restoration, and management. Bioscience, 61(2):107–116, 2011.
[9] Brittany N Blomberg, Terence A Palmer, Paul A Montagna, and Jennifer Beseeres Pollack. Habitat assessment of a restored oyster reef in south texas. Ecological Engineering, 122:48–61, 2018.
[10] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep learning in video multi-object tracking: A survey. Neurocomputing, 381:61–88, 2020.
[11] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In Proceedings of the IEEE international conference on computer vision, pages 764–773, 2017.
[12] Ross Girshick. Fast r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015.
[13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 580–587, 2014.
[14] Raymond E Grizzle, Melissa A Brodeur, Holly A Abeels, and Jennifer K Greene. Bottom habitat mapping using towed underwater videography: subtidal oyster reefs as an example application. Journal of Coastal Research, pages 103–109, 2008.
[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.
[16] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. IEEE transactions on pattern analysis and machine intelligence, 37(3):583–596, 2014.
[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2117–2125, 2017.
[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.
[19] Shuai Liu, Dongye Liu, Gautam Srivastava, David Polap, and Marcin Woźniak. Overview of correlation filter based algorithms in object tracking.
[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.
[21] Katherine McFarland and Matthew P Hare. Restoring oysters to urban estuaries: Redefining habitat quality for eastern oyster performance near new york city. PloS one, 13(11), 2018.
[22] Md Modasshir, Sharmin Rahman, and Ioannis Rekleitis. Autonomous 3d semantic mapping of coral reefs. Technical report, EasyChair, 2019.
[23] Md Modasshir, Sharmin Rahman, Oscar Youngquist, and Ioannis Rekleitis. Coral identification and counting with an autonomous underwater vehicle. In 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 524–529. IEEE, 2018.
[24] German Y Ojeda, Paul T Gayes, Robert F Van Dolah, and William C Schwab. Spatially quantitative seafloor habitat mapping: example from the northern south carolina inner continental shelf. Estuarine, Coastal and Shelf Science, 59(3):399–416, 2004.
[25] Ioannis Papakis, Abhijit Sarkar, and Anuj Karpate. Gcnmatch: Graph convolutional neural networks for multi-object tracking via sinkhorn normalization. arXiv preprint arXiv:2010.00067, 2020.
[26] 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.
[27] 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.
[28] Behzad Sadrfaridpour. oyster-detection. https://github.com/bsadr/oyster-detection, 2020.
[29] Seth J Theuerkauf, David B Eggleston, and Brandon J Puckett. Integrating ecosystem services considerations within a gis-based habitat suitability index for oyster restoration. PloS one, 14(1), 2019.
[30] Kentaro Wada. labelme: Image Polygonal Annotation with Python. https://github.com/kentaroWakelab/labelme, 2016.
[31] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.
[32] M.D. Williams. Estuaries: Redefining habitat quality for eastern oyster performance near new york city. PloS one, 6(11):e28851, 2011.