Computationally Efficient Object Detection Strategy from Water Surfaces with Specularity Removal

by

Danish Faraaz Syed

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2021 by the
Graduate Supervisory Committee:

Wenlong Zhang, Chair
Yezhou Yang
Pavan Turaga

ARIZONA STATE UNIVERSITY

August 2021

ABSTRACT

Floating trash objects are very commonly seen on water bodies such as lakes, canals and rivers. With the increase of plastic goods and human activities near the water bodies, these trash objects can pile up and cause great harm to the surrounding environment. Using human workers to clear out these trash is a hazardous and time-consuming task. Employing autonomous robots for these tasks is a better approach since it is more efficient and faster than humans. However, for a robot to clean the trash objects, a good detection algorithm is required. Real-time object detection on water surfaces is a challenging issue due to nature of the environment and the volatility of the water surface. In addition to this, running an object detection algorithm on an on-board processor of a robot limits the amount of CPU consumption that the algorithm can utilize. In this thesis, a computationally low cost object detection approach for robust detection of trash objects that was run on an on-board processor of a multirotor is presented. To account for specular reflections on the water surface, we use a polarization filter and integrate a specularity removal algorithm on our approach as well. The challenges faced during testing and the means taken to eliminate those challenges are also discussed. The algorithm was compared with two other object detectors using 4 different metrics. The testing was carried out using videos of 5 different objects collected at different illumination conditions over a lake using a multirotor. The results indicate that our algorithm is much suitable to be employed in real-time since it had the highest processing speed of 21 FPS, the lowest CPU consumption of 37.5% and considerably high precision and recall values in detecting the object.

DEDICATION

To my mother, father and the rest of my family who have always motivated me to carry out my work to the best of my abilities and supported me throughout my life.

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to my advisor Dr. Wenlong Zhang for giving me the chance to explore such as interesting field of research at the RISE lab and for providing valuable feedback and advice whenever needed. Thank you for letting me be a part of the lab and involving me in the projects and research. I also want to thank my committee members Dr. Yezhou Yang and Dr. Pavan Turaga for their support and instructions.

A special thanks to Shatadal Mishra for his constant motivation, inputs and advice throughout my time at the lab. Without his guidance, much of this work would not have been possible. I would also like to thank the lab members Kashyap Sathyamurthy and Pallavi Shrinivas Shintre for their help in constructing this thesis and formatting it. I am also thankful to all the other lab members for their friendship and collaborations.

I would like to thank my parents for their immeasurable love, support and for always believing in me. Thank you for making me the person that I am today. Last but not the least, I would like to thank god for helping me get through all the hard times and ending my time at the university on a good note.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

A major requirement for most autonomous robots is perception, which is gives it the ability to see and interpret its environment. While visual perception comes easily to humans, it is not the case for robots as they require a number of complex sensors and computational capabilities. A heavily researched field in robot perception is computer vision. It seeks to develop algorithms that enable computers to extract high-level information using cameras as the primary sensors and images/ videos as the data. These computers are usually integrated with robots such as manipulators, Unmanned Aerial Vehicles (UAVs), Unmanned Surface Vehicles (USVs) and ground vehicles for autonomously running tasks without a human operator. Object detection is a commonly used computer vision technique that is used to recognize an object in the frame and output its corresponding bounding box. This allows us to understand where the object is located in the scene. The task of object detection is usually solved by methods such as background subtraction [1], feature extraction [2], optical flow [3], deep-learning detectors [4] and so on.

Real-time object detection is the task of doing object detection in real time with fast processing speeds and good accuracy. [5] presents a visual system on a self-driving car to detect pedestrians, bicyclists and other vehicles. The work demonstrated in [6] shows the performance of a real-time people counting system using a set of virtual lines on the image. It is also possible to employ detect faces and recognize facial expression as seen in [7]. A deep learning detector is used for object detection on a UAV [8] as a warning system during emergencies.

Performing real-time object detection on challenging environments such as water surfaces is a difficult task as discussed in [9]. Specular reflections on water surface pose the most hindrance to object detection. A number of studies have been conducted to eliminate these specular reflections from water surfaces for collection better images and videos. In [10], specular reflections are removed from images collected by a UAV over a ocean surface. It proposes a two step strategy of removing highlighted regions in the images and then restoring them using a local information around those regions. A model for elimination of reflections is proposed in [11]. However, these works require multiple still images and are not practical for real-time applications. In this thesis, we also use a specularity removal algorithm for better visibility of the floating objects.

However, there have been works conducted for real-time object detection on water surfaces. [12] discusses a real-time floating object detection method on water surfaces using Faster R-CNN. In this work, they use a gamma correction algorithm [13] to enhance the contrast of images without affecting any information contained in the images. However, the work here is employed on a still camera mounted on a high processing GPU. Similarly, the work in [14] demonstrates water surface target detection based using an improved YOLOv3 model on images collected by a UAV. A maritime object detection approach using an electro-optical sensor for navigation of autonomous ships is proposed in [15]. In this thesis, we develop a real-time object detection algorithm using a monocular RGB camera mounted on an multirotor for autonomous detection collection of trash objects on water surfaces. To the authors best knowledge, this is the first presentation that combines specularity removal and object detection in an algorithm.

Multirotors are classified as UAVs with two or more rotors. These extra rotors provide added stability for tasks such as hovering and also the ability to perform

quick and agile maneuvers. A multirotor is composed of different hardware and software subsystems that transfer data within the flight controller to perform the tasks necessary for its operation. There are three main sensors used in a multirotor. Inertial Measurement Unit (IMU) is a fusion of a number of other sensors such as a gyroscope, accelerometer and magnetometer. An IMU is generally used to measure the angular rates, the orientation of the robot and can be used for attitude and position estimation. A GPS is used to provide more precise information of the location and is also used for position estimation. Light Detection and Ranging (LiDAR) sensors are used for collision avoidance and navigation. Position control and attitude control are performed by their respective controllers which uses the desired velocity commands from the high level computer. [16] and [17] propose an attitude controller and a precision controller in a multirotor with disturbance rejection.

Due to their performance capabilities, effective speed and access to tight-bound areas, multirotors are increasingly being used for a number of operations where humans cannot enter. [18] and [19] demonstrate modified multirotors with flexible arms for maneuverability through narrow and cluttered environments. [20] utilizes a multirotor to collect aerial photography and keep track of landscape changes. Another emerging field of research is to use them for aerial package delivery [21]. During emergency situations, they can also be employed for search and rescue operations to provide better visual images and cover a wider area as discussed in [22].

Performing real-time object detection on a robot such as a multirotor increases the difficulty of an already demanding task. This is because of the constraints and CPU limitations when doing object detection on the on-board processor of a multirotor. If the CPU consumption of the detection algorithm is quite high, the processing speed of the algorithm would fail and it some cases, may even fry the processor.

While there have been a number of works conducted for real-time object detection strategies on water surfaces, they do not consider employing the algorithms on an on-board processor with no GPU a limited CPU capability. In addition to this, none of the works demonstrated the effective use of an object detection algorithm on a moving camera over water surfaces. The emphasis of this thesis would be to develop a real-time working detection strategy to be deployed on a robot with a down-facing camera, specifically multirotors.

1.1 Problem Statement

One of the most common forms of pollution in water bodies such as lakes and canals is floating plastic debris as seen in Figure 1. These debris can arise due to littering, fishing activities and garbage disposals. While plastic in oceans get carried to other places due to wind and ocean currents, trash in stagnant water bodies such as lakes usually have nowhere to go and start accumulating over time. The plastic debris are usually floating trash items in the form of bottles, caps, cartons, bags and beverage cans.

While the trash are of great concern by themselves, they become more harmful once the plastics start breaking down into microplastics. The microplastics are too small to be effectively removed through sewage treatment and can have a ranging impact from affecting the quality of drinking water to causing harm to the aquatic organisms. Hence, it is of great importance to remove the plastic debris from the water bodies before their decomposition starts. Using a human worker for such an operation consumes more time, utilizes man power and can also prove to be fatal in some cases.

Figure 1: Floating plastic debris on a lake (File image courtesy Monica Volpin / Pixabay)

Autonomous robots such as UAVs and USVs are potential vehicles that can be used to collect trash objects and mitigate the problem of water surface pollution. Considerable work has been done wherein UAVs have been used for aerial grasping of objects from rigid surfaces. [23] demonstrates the control of a multitoror attached with a 7-degree of freedom arm that can be used to carry out various missions. Similarly, the work in [24] proposes aerial grasping of objects using a UAV with a gripper. [25] demonstrates an origami-inspired foldable robotics arm mounted on a UAV to retrieve objects located in deep and narrow spaces. An autonomous hexacopter with a soft grasper for autonomous detection and collection of objects is also seen in [26]. The work in [27] presents an hybrid Unmanned Aerial-Ground Vehicle (UAGV) that lands near the object and grasps it from the ground modality. However, a common limitation of all these works is that the graspers used have a limited workspace. Therefore, our proposed algorithm was run on a multirotor with an integrated net system for autonomous collection of floating trash objects from lake surfaces with an increased workspace [28]. The object detection algorithm is run on a high-level

computer integrated with the multirotor, which is an Intel UpBoard with 4 cores. The multirotor had a customized flotation system and a downward facing monocular camera as shown in Figure 2. It was equipped with a PIXHAWK flight controller, with an IMU used for attitude estimation and a GPS for position estimation. We also integrated a Teraranger Evo (Terabee, France) single-shot LiDAR for height estimation above the water surface for landing.

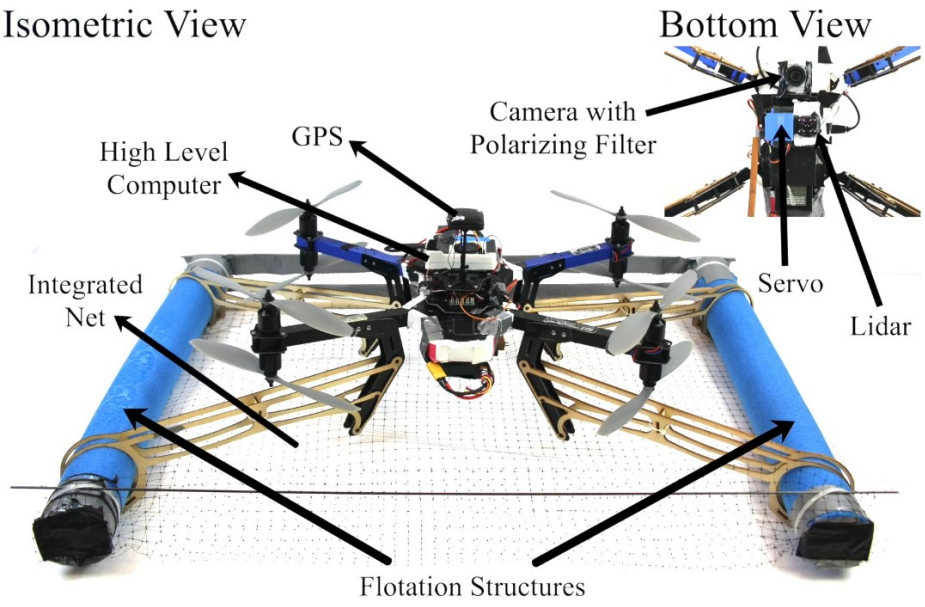


Figure 2: Isometric and Bottom view of the multirotor system with all the sensors and other components labelled

1.2 Challenges

While it seems fairly simple to detect floating debris on water surfaces, it is quite complicated and poses a lot of challenges when performed real-time. The major challenges that was faced when developing and testing our algorithm is discussed in detail below -

1. Specular reflections

   Heavy sunlight during morning and afternoons give rise to specular reflections on the water surface. The reflections are random and unpredictable in shape, size and intensities. In addition to this, Water currents due to the winds increases the volatility of these reflections making it harder to predict. These reflections are a cause of major concern during detection since some of these reflections completely overwhelm the object, making it almost impossible to detect.

2. Illumination conditions

   Another challenge in testing outdoors is the varying illuminations conditions that are caused by various factors such as intensity of sunlight, position of the sun and weather. Varying illumination conditions affects the reflective properties of the water surface as well as the object being detected.

3. Presence of the shore

   While the object detection looks for floating trash on the water surface, it should also take into consideration that when the object may drift near the shore and change the scene. During such cases, the frame changes from being in a completely dynamic background of the water surface, to partially covered by the static background of the shore .This implies that the algorithm should be able to adapt and work with scene changes.

4. Water surface disturbance

   The dynamic nature of water surface due to water currents, winds and propeller downwash from the drone makes it so that the floating object drifts constantly and is never static at one point. This also resulted in the object being drifted out of the camera's field of view for a few frames.

5. Properties of the object

Different objects have different materialistic properties such as shape, size, colour, reflectivity. Objects of darker colour are much difficult to detect since it is difficult to distinguish it from the background. In the same manner, highly reflective objects also pose a challenge to detection since they closely resemble the specular reflections on the surface of the water.

6. Scale changes and occlusion

   During the landing of the multirotor on the object, the object is subjected to a lot of change in aspect ratio and rotations. Hence, we could not expect a fixed scale and rotation angle to the object. In addition to this, partial occlusions of the object due to drift and shadows were also a challenge to be overcome.

7. Moving camera

   Since all the testing was performed by a camera mounted on a multirotor, the collected videos were subjected to a lot of wobbling due to the chattering and vibrations of the camera. The camera is never steady due to the motion of the multirotor in addition to the vibrations due to the propellers. An unsteady video feed would mean that the detection algorithm should be responsive to small and random changes in the location of the object in the frame.

1.3 Organization of This Thesis

This thesis is divided into two halves. The first half consists of two chapters. Chapter 2 gives a brief overview of the literature review on computer vision approaches used for object detection and how to select the right one depending on the problem statement. It is then followed by a detailed description of a preliminary tracker that

was developed and its validation. We also describe the failure cases of this tracker, reasons for failure and elaborate on why it was abandoned.

The second half deals with development of the object detection algorithm and its experimental results. In Chapter 3, we describe the selection of our image sensor and on why it was chosen. A detailed explanation of the collection of the dataset with different objects is described. The steps taken to eliminate the challenges faced are then discussed and the object detection algorithm is described in detail with its test results. We demonstrate the experimental results of the detector when compared with other existing detectors in Chapter 4. We also discuss the collection of the dataset for comparison and the process by which ground truth labelling was performed. It is then followed by the concluding remarks and a brief overview of the future work.

1.4 Research Objective and Contributions

The objective of this thesis is to come up with a robust object detection algorithm with the following properties -

- It must accurately detect floating trash objects on the water surface and output the corresponding location in the frame with no false detections. A wide variety of floating trash objects can be found, and hence the detector must be perform well on a diverse range of objects.
- Since we run the algorithm real-time on an onboard processor of an autonomous robot, it must have a high rate of processing and also consume as little CPU as possible. This allows for a smooth operation with necessary memory available for other processes to run as well.
- Due to varying illumination conditions and nature of an outdoor environment,

it must effectively handle specular reflections, occlusions, shadows and other obstacles in a dynamic background such as water.

- Water currents produced by winds and the propeller downwash from the multicopter makes it so that the object never remains static at a position for an extended period of time, and instead drifts constantly. Hence, it is of considerable importance for the detector to be able to detect an object even after it leaves the camera's frame of view and comes back into the scene without any failure.

With the above mentioned properties in mind, a reliable and robust object detection algorithm was developed and is presented in this thesis. The two main contributions of the thesis are as follows -

- A computationally efficient two-phase algorithm is developed for robust detection of floating trash objects on water surfaces. An edge-based contour detection approach is employed for effective detection of different objects under varying illumination conditions.
- A specularity removal algorithm is implemented in the code for effective removal of specular reflections on the water surface without affecting the features of the floating object.

1.5 Assumptions Made in This Thesis

Two major assumptions were made in this thesis.

1. There is only one object in the scene to be picked up for each trial being

conducted. This is stemmed from the fact that the multirotor that was used for our tests was equipped with the facility to pick up one object for each launch.

2. The scene is observed from a bird's-eye view perspective. A downward facing monocular camera is used, which is mounted on the multirotor. This placement of the camera had the best field of view and also made it easier to land the multirotor exactly on top of the object.

Chapter 2

LITERATURE REVIEW

2.1 Existing detection strategies

Object detection in computer vision is usually solved using two different strategies - traditional computer vision and deep learning. It is highly recommended to consider the advantages and disadvantages of both these approaches to decide on which can be better employed for the given problem statement before proceeding further.

2.1.1 Traditional computer vision

This approach employs the use of existing feature extraction techniques for object detection. Features are defined as small regions of interests in the images, with necessary information contained within them. This extracted information is then used to define and detect objects. Some traditional feature extraction techniques are detailed below.

1. Edge detection

   Edge detection is one of the oldest image processing technique developed to identify information in an image. The idea behind edge detectors is that there are sudden discontinuities in an image where there are edges. Hence, these algorithms work by detecting sudden change in pixel brightness intensities, which are characteristic of edges. Edges can provide a broad understanding of the object detected, such as the shape, size and orientation. Some commonly used

edge detection techniques are Sobel edge detector [29], Canny edge detector [30] and the Laplacian edge detector [31]. The output of Sobel and Canny edge detector for a given input image is shown in Figure 3 (b) and (c) respectively.

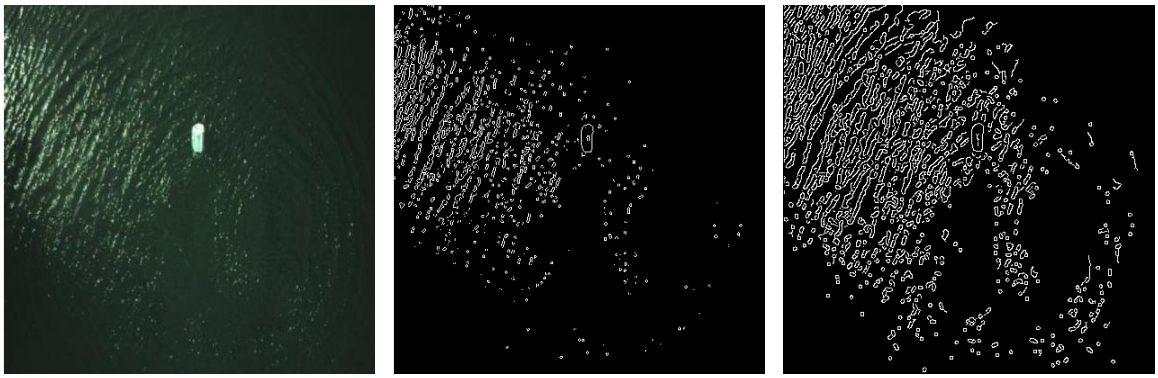


Figure 3: Edge detection. (a) Input image; (b) Sobel Edge detector; (c) Canny edge detector

2. Feature detector and descriptors

   Feature descriptors are algorithms that take an input image and output the location points of interest, also called as keypoints, and their corresponding descriptors based on some criterion. A descriptor is usually a vector of values that describes the neighboring patches of the keypoint. These vector of values could contain anything from raw pixel values to histogram of gradient orientations.

   Most feature descriptors are invariant to changes in scale, translation and rotations and hence are used for a wide range of applications such as object detection, 3D reconstruction and image matching. Scale-Invariant Feature Transform (SIFT) [32] is a feature detector and descriptor which detects centers of blob-like structures in an image using a Difference-of-Gaussians method. The SIFT descriptor on the other hand, is based on the histogram of gradient

orientations. A much faster working version of SIFT was developed, which is called Speeded Up Robust Features (SURF) [33]. Another commonly feature detector and descriptor is Binary Robust Invariant Scalable Keypoints (BRISK) [34], which used a corner detector for its keypoints detection and has a descriptor containing the signs of the difference between certain neighboring pixels. Oriented FAST and Rotated BRIEF (ORB) [35] was proposed as an efficient alternative to SIFT and SURF due to its resistance to noise and rotation invariance. ORB was shown to be computationally efficient, and it some cases two-times faster than SIFT. [36] proposed KAZE features which detects 2D features based on nonlinear diffusion filtering [37] rather than approximating the Gaussian scale space [38] as in the other feature detectors. KAZE features provided better accuracy and performance than the previous methods, but with added computational expense. Figure 4 (b) and (c) show the 20 strongest keypoints detected by SURF and BRISK respectively. The size of the drawn keypoint represents the strength of the feature that has been detected.

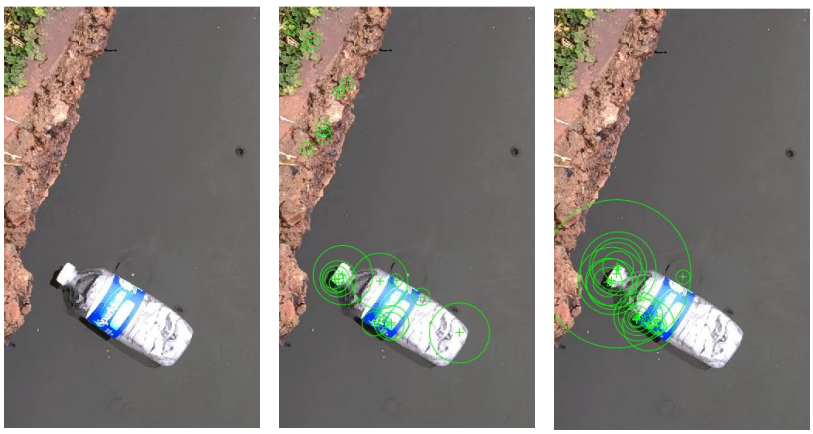


Figure 4: Feature detection. (a) Input image; (b) SURF Feature detector; (c) BRISK Feature detector

3. Haar Cascades

   Haar Cascades [39] is a face detection algorithm that uses its own feature called as haar features. These features are a set of rectangular blocks that are used to find face segments such as the eyes, nose and the lips. The concepts behind Haar Cascades is that there are areas within the face, such as the eyes, which are significantly darker than other area, such as the cheeks. The haar feature contains a set of adjacent rectangles which is transversed from the top-left of the image to the bottom-right. It then computes the difference between the sum of pixel intensities in each rectangle and compares it with a pre-defined threshold to identify the face.

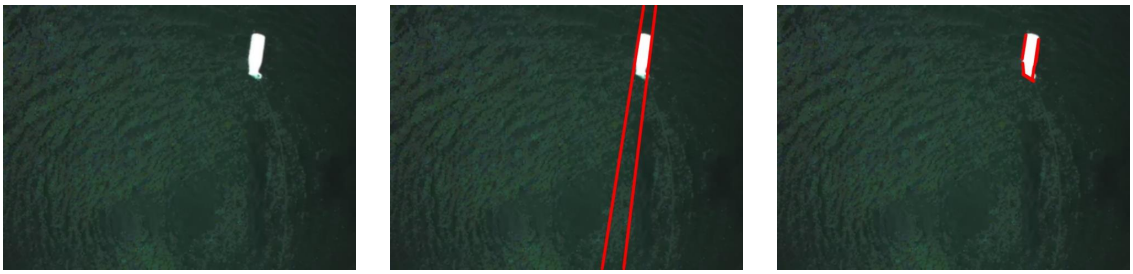


Figure 5: Hough Line Detection. (a) Input image; (b) Standard hough transform; (c) Probabilistic hough transform

4. Hough transform

   Hough transform [40] is a method that it used to detect lines, circles or ellipses in an image. It is necessary to perform a binary edge detection or thresholding on an image first, and then provide it to the Hough transform. To detect lines, it used the parametric equation of a line: $\rho = x\cos(\theta) + y\sin(\theta)$ to construct the Hough space. The Hough Space is a 2D plane with ρ as the vertical axis and θ as the horizontal axis. Depending on the resolution of the image, an

accumulator array is initialized to detect different kinds of lines. For every edge pixel, the value of θ is varied from 0 to π and each corresponding ρ is computed. The accumulator array is used to find the intersection of the curves and it is incremented for each non-zero pixel that is close to the corresponding line. Finally, a pre-defined threshold is used to find the lines in the image. OpenCV implements two different types of Hough transform. The standard hough transform is described above and it outputs a parametric equation for a line. The probabilistic hough transform on the other hand, is a more efficient version and can output the extreme ends of a line. Both versions of hough transform are show in Figure 5.

2.1.2 Deep learning

Deep learning is a subset of machine learning that can be used for various computer vision applications such as object detection and tracking, semantic segmentation, Simultaneous Localization and Mapping (SLAM) and object classification. For deep learning, the system is initially provided with a dataset of images which have been labelled according to what each image contains. The deep learning model is then "trained" on these images using neural networks that learn the patterns and features of each of the classes in the dataset.

The concept of deep learning is derived from the use of neurons to extract high level information i.e features from an image and then pass it on to a classifier that uses the information to make predictions. This process also occurs in our brains wherein the first set of neurons in our cortex extract simple features such as edges, whereas the neurons further down help in extracting high level information. Each set of neurons

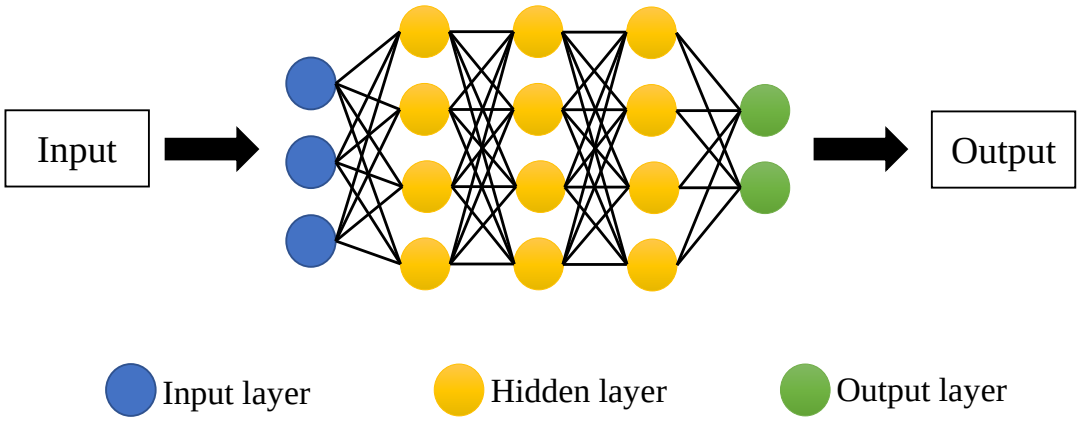


Figure 6: A simple deep learning neural network model

in a deep-learning model is called as a layer. A layer takes in an input, performs operations on it based on a set of functions and then passes it on to the next layer. The first layer of a model is called as the input layer, the last layer is called output layer and all the layers in between are part of the hidden layer. Usually, each layer performs one operation such as convolution, pooling, padding etc.

With the development of Convolutional Neural Networks (CNN's), deep learning saw a sudden jump is research and use due to its increased object recognition capabilities as discussed in [41]. CNNs use a matrix of values, called as kernels, to convolve the image and hence the name. There could be multiple convolutional layers, each trained to detect a specific pattern or feature, in a deep learning model. The kernel matrix transverses over the input image matrix and a dot product is carried out between the kernel and the area which is overlapped. The output is summed and then fed to an activation function. The output of the activation function indicates whether a neuron would be ON or OFF. Some commonly used activation functions are Linear, Sigmoid, TanH and ReLu, which is selected based on the task to be performed. A

sample deep learning network is shown in Figure 6 with 1 input layer, 3 hidden layers and 1 output layer.

At the end of a typical deep-learning model, a loss function is computed, which is used to decide on the output of the model. It contains a set of weights or errors. The more accurate a model is, the lesser its error would be and hence this loss function is to be minimized. There are many optimization functions that seek to do this, the most common of which is the gradient descent algorithm. The algorithm starts by a set of random weights and the loss function is computed for it. The algorithm then computes a slope for this function which indicates the set of values to be initialized to the weights to attain a lower loss for the next iteration. A learning rate is used to control the amount by which the weights can change for each iteration. The process is repeated until the loss function is stabilized at its lowest number. This weights used to get the minimized loss function is then used for testing the model with other inputs.

2.2 Selecting the right approach

Both traditional computer vision and deep learning have their trade-offs to be considered. Analyzing the problem statement thoroughly and having a clear understanding of its requirements will make it easier to choose the right approach.

One of the major drawbacks of the traditional approach is that as the number of classes to be detected increase, it becomes difficult to identity features that best describe all the different classes. Moreover, it is necessary to fine tune all the parameters by trial and error approach to attain the best working detection. Using deep learning models on the other hand, based on the input dataset, a wide variety of classes can

be detected. The problem of fine tuning the parameters is also eliminated since the training of the model is automatic and it fine tunes the parameters by itself. Deep learning models are also generally seen to have better accuracy in detecting objects than traditional approaches. The models can also be retrained on another dataset for other purposes, contrary to traditional approaches which are very specific to a set of features.

However, the above listed drawbacks do not mean that deep learning is the best approach for all scenarios. Most deep learning models require a huge training dataset before they are able to have a good accuracy. Training on a big dataset also entails that it is time consuming before the model can be deployed for use. A large enough dataset may not always be available and it consumes further time if a custom dataset is to be prepared from scratch. The trained models are also very dependent on the dataset they were trained on. Failure can occur if the image resolution of the training images are different from the test images. Similarly, overfitting is a very common issue that could happen wherein the deep learning models perform well on the train images but have poor results on the test images due to overfitting of the parameters.

For our problem statement, we do not have multiple classes of objects and also assume only one object in the scene at a given time. It would be a overkill to deploy deep learning models to solve this problem. Additionally, traditional approaches have higher processing speed than deep learning models. Since our tests were conducted real-time, higher processing speeds is a priority. Moreover, deep learning models have a higher CPU consumption which was not feasible for our case. Since we run all our algorithms on an e UpBoard with 4 cores, we had to limit our CPU expense for other processes to run. Hence, considering all the factors listed above, we employ a traditional computer vision approach to detect the objects.

2.3 Preliminary strategy

2.3.1 OpenCV

Open source Computer Vision library (OpenCV) is the most popular open-source library that is used to solve computer vision applications in real-time. It has number of image processing functions, detectors and trackers that can be used for almost all computer vision applications. Tracking algorithms have a lower computational expense than that of detectors since they do not process the entire image but only the portion where it estimates the location of the object. However, a major drawback of these trackers is that they do not recognize an object when it leaves the frame and comes back again. In such cases, these trackers fail and should be initialized again. Since trash objects on water surfaces can easily drift out of the field of view of the camera, we required our tracker to be able to recognize the object after it comes back into the scene.

2.3.2 Trackers in OpenCV

The OpenCV library contains a number of trackers that can be implemented within a few lines of code. These trackers are the Boosting tracker [42], Multiple Instance Learning (MIL) tracker [43], Kernelized Correlation Filter (KCF) tracker [44], Tracking-Learning-Detection (TLD) tracker [45], Median Flow tracker [46], Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT) tracker [47] and the Minimum Output Sum of Squared Error (MOSSE) tracker [48]. The working of some of the most commonly used trackers is described below.

The MIL tracker employs the concept of "tracking by detection" wherein an online classifier is trained to segment the object from the background. The classifier uses the current bounding box provided by the user and looks around the neighboring patches to compute several positive examples. It the sets up a number of positive and negative bags with each bag containing the corresponding potential positive and negative examples. This is based on the intuition that even if the tracking is not accurate, there may be a positive example inside the positive bag that contains the correct bounding box. Although this tracker can handle partial occlusions of the object, it cannot handle complete occlusions. In addition to this, this tracker does not always report failure properly since even small inaccuracies during tracking can affect the training of the classifier.

KCF tracker is a fast working tracker that utilizes concepts used by the Boosting tracker and the MIL tracker. It looks for overlapping regions within the collected positive examples and applies Discrete Fourier Transform [49] on the data, diagonalizing it to reduce storage and computation. This is then applied on a kernel ridge regression [50] to obtain the tracker. KCT tracker is one of the fastest tracker till date, running at hundreds of frames-per-second while being accurate. However, this tracker is prone to failure easily and does not recover from occlusions either.

TLD tracker, as the name suggests, divides the tracking problem into 3 sub-components - tracking, learning and detection. The tracking keeps a track of the object for each frame. The detector consider every frame as a separate image and localizes all the features that are seen in the current image and the ones before. It then uses these features to correct the tracker if necessary. The learning observes the false positive and false negative errors of the detector and estimates them to avoid them in the succeeding frames. TLD tracker has one of the best performance compared to the

other tracker. It is also able to recover from full occlusions satisfactorily. However, this tracker has the most computational expense due to all the processes running in the background.

The CSRT tracker solved the problem of short-term tracking by employing the use of Discriminative Correlation Filters (DCF). Both spatial and channel reliability concepts are introduced to the DCF. The spatial reliability map updates the filter support to the part of the object that is suitable for tracking. This enlarges and localizes the selected region, which is used for tracking of circular and non-rectangular objects. The channel reliability on the other hand, is estimated to weight the per-channel filter responses in localization. This tracker offers a comparatively high accuracy of tracking with decrease in processing speeds.

The trackers discussed above work differently than one another and have their pros and cons. However, a major concern with all these trackers is that all of them except for TLD tracker result in tracking failure during scenarios of complete occlusions. These trackers do not recognize the object once they back in the scene and have to be re-initialized again, which is not practical for real-time cases. While TLD tracker is able to do so, it has a very high CPU consumption with a low processing speed which makes it unsuitable to be run on a onboard processor. Hence, we had to look for a tracker that does not fail under occlusions, can recognize objects that leave the scene and come back again, and has good accuracy in tracking with significantly low CPU usage.

2.3.3 CMT tracker

Clustering of Static-Adaptive Correspondences for Deformable Object (CMT) tracker [51] is an open-source tracker which is compatible with OpenCV and can be tested out easily after a simple download. The code is available in both Python and C++. We prefer the later due to its lower CPU consumption. CMT tracker is a keypoint-based tracker that can re-recognize an object if it leaves the scene and comes back into the frame after a few moments. This makes it a very suitable tracker for our purpose. The main idea of the CMT tracker is to use keypoints for breaking down the object into multiple fragments. This allows it to be easier to compare the individual fragments with a descriptor rather than using the entire object as a whole.

The working of the CMT tracker is as follows. Initially, in the first frame, a bounding box containing the object to be tracked is defined. A set of foreground and background keypoints are computed inside this bounding box. The keypoints are their descriptors are detected using Binary Robust invariant scalable keypoints (BRISK), since it is invariant to scale and rotation changes. These keypoints will be later used for matching as the images iterate. A static appearance model is first employed wherein a global search is used to match the keypoints between the initial frame and the current frame. This model works well from re-detection of the object after occlusions. However, to account for changes in object appearances, an adaptive model is also updated after every frame. By assuming the time between consecutive frames to be small, a sparse optic flow is used to establish correspondences from frame $t-1$ to frame t using Lucas-Kanade algorithm.

When both the models yield the correspondences, the result of the static model is used since it is more robust. All the correspondences are then clustered by using

a standard agglomerative clustering algorithm [52]. It is assumed that the largest cluster of the correspondences contains the object whereas all other clusters are part of a clutter. Since keypoints with similar descriptors are difficult to match, they excluded the correspondences that are dissimilar to the largest cluster using a second matching round. Once the outliers are removed, the remaining keypoints are used to compute a rotated bounding box which contains the object being tracked and the process continues.

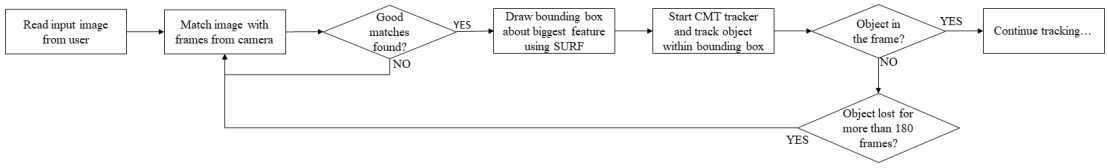


Figure 7: Block diagram representing the initial algorithm for detection and tracking of trash objects

2.4 Implementation of CMT tracker

Before initializing a tracker, it must be provided with the bounding box containing the object to be tracked. This can be done manually by a user drawing the bounding box over the frame or by providing the coordinates of the bounding box to the tracker. However, this is not feasible for our case since we perform the tracking real-time autonomously without any help from the operator. In such instances, a detector is first run to get the bounding box which is then subsequently used to initialize the tracker. Hence, the developed algorithm consists of two components - detection and tracking. The detection component employs the use of a Bruteforce matcher and SURF detector. The tracking is initiated using the CMT tracker once the object

has been detected. This algorithm is discussed in detail below and is represented by Algorithm 1. For better understanding, the flow diagram of the algorithm is shown by Figure 7 and a visual representation is provided in Figure 8.

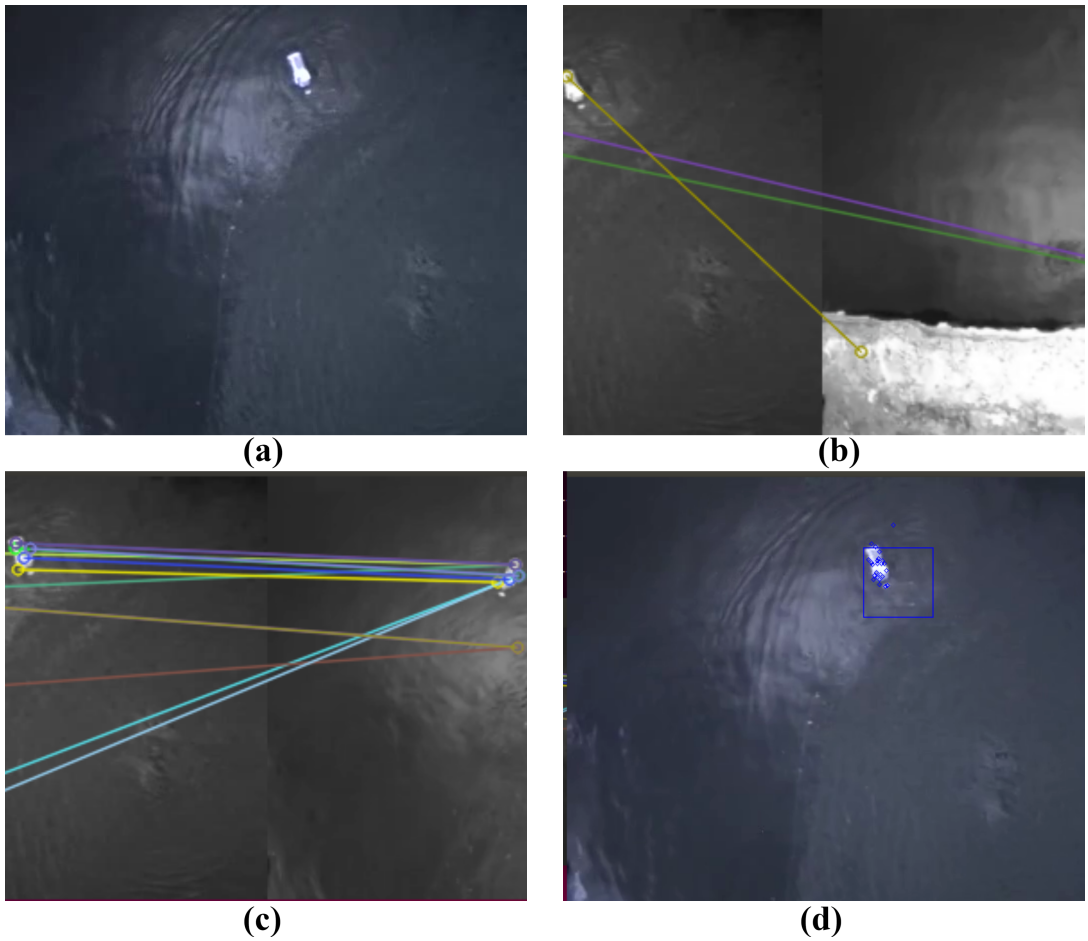


Figure 8: Visual representation of the algorithm. (a) Input image of the object to be tracked; (b) Feature matching between the input image and the video feed; (c) Good features have been matched i.e object has been detected; (d) Tracking starts

2.4.0.1 Detection

Detection is the first stage of this algorithm and is used to locate the object in the frame. An image of the object to be detected is provided to the algorithm. Detection of the object is done in two stages - matching the input image to video stream and then using a feature detector to draw a bounding box around the object. The video feed starts as soon as the multirotor takes off and until the collection of the object.

Matching the input image and video feed is done by using a Bruteforce matcher. It is a simple feature matcher that compares descriptor of one feature in an image with the descriptor of features in another image. Based on distance calculation, the closest features are returned as matches. To generate the features of the input image and video stream, we use Scale-Invariant Feature Transform (SIFT) to extract keypoints and compute their descriptors. SIFT, while being computationally inexpensive, was also able to detect features when the video stream contained only water and no objects on its surface.

Since the matching process is run for as long as the object is not found, it becomes essential to use the feature detector which gives a satisfactory amount of good matches without having an adverse effect on the computational expense. Hence, we choose SIFT to be our feature detector. Once all the matches has been found, a threshold was applied to filter out only the good matches. Since a few matches could be false positives, the algorithm was coded such that it confirms the presence of the trash in the input image to be in the video stream only if good matches have been recorded for six or more consecutive frames. Once this has been established, the matching is stopped.

SURF Feature Detector and Descriptor is then applied on the consecutive frame

Algorithm 1 Vision architecture for Detection and Tracking of trash item using CMT tracker

```
Input: Image of object to be detected and tracked
bounding box ← false
i ← read input image
while bounding box == false do
    f ← get frame
    Matching(i,f)
    if good matches found then
        SURF(f)
        bounding box ← true
    end if
end while
f ← get frame
initialize CMT
while true do
    f ← get frame
    Tracking(f)
    if Tracking lost for more than 180 frames then
        bounding box ← false
        while bounding box == false do
            f ← get frame
            Matching(i,f)
            if good matches found then
                SURF(f)
                bounding box ← true
            end if
        end while
    end if
end while=0
```

to find all features and descriptors within it. SURF is a speeded up version of SIFT and hence the name. While SIFT uses Difference of Gaussian to approximate Laplace of Gaussian, SURF does the same by using Box Filter.

A threshold hessian is applied to eliminate features that have low strength. Once the features and their corresponding descriptors have been computed, the feature

whose descriptor has the largest radius is identified and inscribed by a bounding box which is then passed on to the CMT tracker.

Since we are looking for objects on water surface, which is quite featureless, the feature with the largest descriptor radius is usually the object that is to be detected and tracked.Moreover, we consider the case where there is only one object in the scene. When compared with other feature descriptors like SIFT, ORB and KAZE, it was found that SURF provided the largest descriptor that encompasses the whole object to be tracked. This is why we use SURF even though its computationally more expensive than the alternatives.

SURF could also be used solely without the matching to find the largest object in the scene. But this approach poses various problems. First, when there is no object in the scene, using SURF to get the largest feature would usually result in some arbitrary features such as dust particles or water ripples being detected. In addition to this, we have no way of recognising an object once it comes back to the frame if only SURF is used for detection. Hence, this is why an image of the object to be detected is provided to the Bruteforce matcher.

2.4.0.2 Tracking

The CMT tracker is initialized using the bounding box computed by SURF. By default, the CMT uses the Features from Accelerated Segment Test (FAST) [53] feature detector and BRISK to detect the keypoints and compute the corresponding descriptors respectively within the bounding box. This combination works well for our purpose and hence they were not changed. The keypoints would be used for tracking the object as the video progresses and to handle occlusions as well. The multirotor

uses the bounding box coordinates of the CMT tracker to track the object and keep it exactly below the multirotor for increased success of collection. If the object being tracked is lost or occluded for a defined number of frames, the code reverts back to the process of matching and continues doing so until the object reappears.

2.4.1 Validation of the feature matching and tracker

To compare and evaluate the matcher and tracker of this algorithm, we collected a real-time video of bottle on a lake using a USB camera attached to a drone. The video was then post-processed before being used for the experimental validation.

Table 1: Comparison of different feature detectors on various objects

Bottle	SIFT	ORB	KAZE	SURF
Number of good matches	14	8	6	**15**
CPU usage (in %)	**47.05**	66.67	55.88	75

Table 1 below compares the results of SIFT with other alternative such as ORB, KAZE and SURF. For the water bottle, in terms of number of good matches, SIFT performs comparatively well with 14 good matches. The only other detector which yields better results is SURF which has 15 good matches. ORB and KAZE on the other hand, give just 8 and 6 good matches respectively. SIFT has the lowest CPU usage of 47.05% while ORB, KAZE and SURF take up 66.67%, 55.88% and 75% respectively. We choose SIFT rather than SURF due to its considerably low CPU usage with a slight trade off in number of keypoints detected.

We used two performance measures - precision and recall to evaluate the CMT

Table 2: Comparison of different trackers

	CMT	KCF	MIL	TLD
Precision	**0.675**	0.5283	0.4782	0.5905
Recall	**0.6304**	0.45	0.5286	0.6232

tracker with other trackers such as KCF, MIL and TLD. The comparison is shown in Table 2. CMT outperforms all other trackers when comparing both precision and recall.It tracks the object efficiently and is also able to recognise and restart tracking if the object comes back into the frame after a period of being hidden or disappearance. While KCF tracker seems to start of well with the tracking, it fails as soon as the object is slightly obscured. MIL tracker, on the other hand, fails in the sense that it starts tracking the ROI where the object was last visible and doesn't seem to recognise the object when it comes back into the scene. The TLD tracker comes to closest to performing as good as the CMT tracker. It even seems to recognise the object when it comes back into the frame. The only issue in case of the TLD tracker is that it is much more slower and computationally more expensive than the CMT tracker. While CMT tracker had the best performance when compared with the rest, this algorithm proved to have a lot of disadvantages which led to it not being used further.

2.5 Limitations and need for a better approach

Using the SIFT detector and descriptor with the CMT tracker had a successful tracking output for the first few times. But on testing it real-time with different objects and weather conditions, it failed. The issues with this algorithm is described in detail below -

1. Detection failure leads to tracker failure

   The working of the tracker is dependent on the bounding box provided by the detector. Although the feature matching runs until the object in the scene, the SIFT keypoint detector draws the bounding box over the biggest keypoint detected when the matching is satisfied. This may or may not be the object to be tracked. The detector is initialized when the feature matching is satisfied, but is independent of other features in the scene.

   During our testing, the SIFT algorithm sometimes detects the water ripples caused by winds and the propeller downwash, as the biggest object in the frame and draws a bounding box around it instead, though the trash object is in the frame. Subsequently, providing the bounding box of the water ripples to the tracker causes it to fail and the object is not tracked at all.

2. Issue of reflections

   A large number of specular reflections on the water surface that arise during sunny weather conditions also lead to detection and tracking failure. The reflections are so overwhelming that they are bigger than the object to be detected and sometimes completely obscure the object so much that it becomes impossible to detect it. We encountered failure of the tracking a number of times when we conducted real-time testing in sunny conditions.

3. Need to provide an input image

   Another major cause for concern of this algorithm is the need to provide an image of the object to be detected. This is not always practicable since we the algorithm needs an image of the exact object to be detected, which may not be available in real-time. If the image is not as desired, the feature matching would continue to run and the algorithm would never progress to tracking. In addition

to this, there are a wide variety of trash objects that could be encountered. It is not possible to have an image of every trash object that we need to pickup.

4. CPU consumption of the detector

   Detection is computationally more expensive than tracking. Using feature matching for detection consumes more CPU the longer it is run. For as long as the object is not in the scene, the feature matching is run. During real-time testing, the CPU consumption of the detector reached as high as 2 cores, which is not desirable to be on a UpBoard with 4 cores.

5. CPU consumption of the tracker

   Although the tracker is not more CPU expensive than the detector, there are some cases where it consumes a lot of CPU. One of the main causes is when the feature detector of the tracker detects a lot of keypoints and hence starts having a frame rate drop and increase in computational expense as it has to keep track of a lot of keypoints. This issue was seen when the tracker detects keypoints over the water ripples inside the bounding box.

With the challenges and drawbacks of the CMT tracker, it became essential to develop a better object detection algorithm that would be able to would be able to work with better accuracy and low rate of failure. It should also handle the issues faced by the previous algorithm, especially failure due to occlusions and out-of-view scenarios. The approach to designing another algorithm is described in detail in the next chapter.

Chapter 3

DEVELOPMENT OF A 2-PHASE OBJECT DETECTOR WITH SPECULARITY REMOVAL

This chapter will go over on how a better detection strategy was developed and on how it was able to overcome the challenges and disadvantages of the previous one. The first section of this chapter shall introduce the selection of the hardware setup and their use in collection and processing of test videos. The second half of this chapter will be used to discuss the development and working of the algorithm in detail.

3.1 Image sensors

Any sensor that is able to perceive an environment and convey the information in the form of an image is called an image sensor. A camera is the most common image sensor used for computer vision and machine vision applications. All of the videos and images used for the completion of this thesis were collected and processed using an oCam-1CGN-U (Withrobot, Rep. of Korea) 1 Mega Pixel USB 3.0 colour global shutter monocular camera as seen in Figure 9.

This camera outputs images of Bayer RGB format which can be converted to RGB by using a provided function that is integrated with OpenCV. The camera also has it's own viewer program that can be used for various purposes such as varying the exposure and gain values, colour correction, switching between manual and automatic auto-focus, and so on. The camera can process upto 54 fps at image resolutions of 1280 x 960 and 180 fps at resolutions of 640 x 480. In addition to having fast processing

Figure 9: oCam-1CGN-U

speeds, it also has a low CPU usage since it writes data to the main memory of the board without passing through the CPU using a USB 3.0 interface.

Based on the type of shutter, cameras can be classified into global shutter and rolling shutter. In global shutter cameras, the active-pixel sensors inside the camera scans the entire area of the image at the same time, whereas in rolling shutter, the image is scanned from the top to bottom, line-by-line. We use a global shutter camera for our purpose, since rolling shutter cameras have a lot of disadvantages when it comes moving objects. Figure 10 demonstrated the difference between a rolling shutter and a global shutter for a 4×4 pixels camera.

Rolling shutter effect is used to describe a list of the common issues found in rolling shutter cameras. Some of these issues are wobbling, skew and aliasing. Wobbling occurs when the image is taken from a unsteady support of when it is taken from a fast moving vehicle, resulting in a very fuzzy image. The skew effect occurs when either the camera or the object is moving from one side to another. Due to the nature of

rolling shutter, it distorts the image such that the object is tilted towards the direction of motion. Spatial and Temporal aliasing are caused due to parts of the object moving faster than the rest and partial exposure of the image, respectively. Both these cases cause portions of the image to be discontinuous with the rest. A global shutter camera does not encounter these issues and also has the added advantage of better consistency and accuracy in the image.

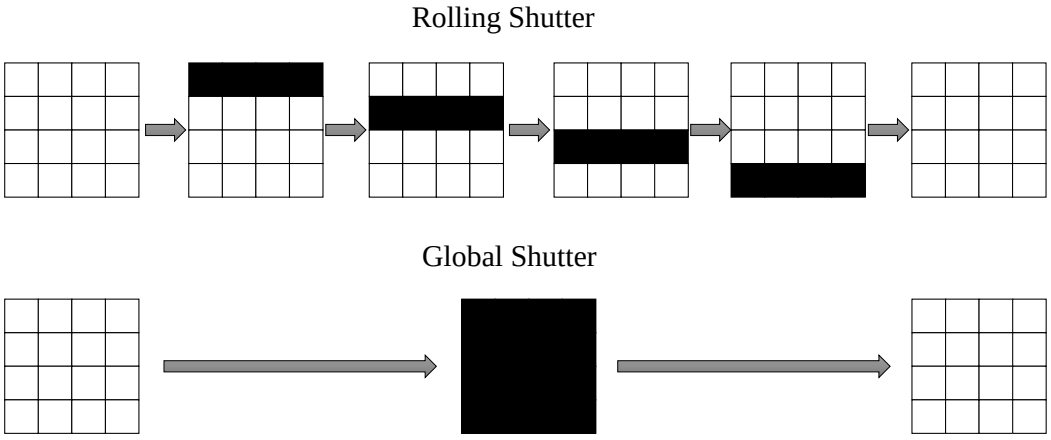


Figure 10: Difference between a rolling shutter and a global shutter camera

3.2 Collection of the videos

For testing the object detection algorithm, we require a good number of videos containing different objects at varying light conditions. Five different objects of different shapes, sizes and shades were used, namely - a white bottle, a white carton, a silver can, a red carton and a dark can as shown in Figure 11 with their respective

dimensions and weights. All of the videos were collected in a lake park at Gilbert, Arizona (lat:33.3589, lon:-111.7685) at different times of the day.

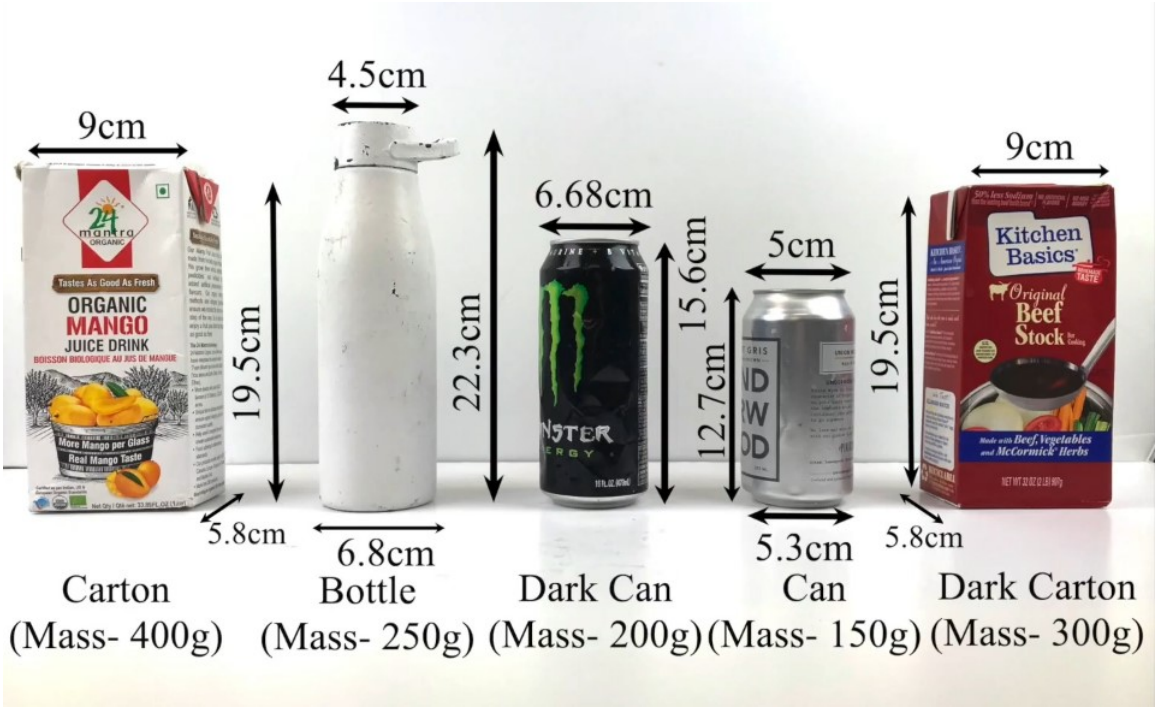


Figure 11: Objects used for collection of the dataset

First, the objects were tied tautly with a rope to prevent them from drifting away too much and loosing them. The objects were then thrown at a distance from the shore manually. The multirotor with the downward facing camera is taken off from its base station and then the video recording is started. The videos at 80 FPS with an image resolution of 640×480. After multiple trails, we also found that the exposure value of 15 and gain of 45 has the best output without making the videos too dark or too bright, and hence we set those values manually before recording. The multirotor takes the video from various angles and altitudes. A number of scenarios such as partial occlusions and drift of the object away from the camera's field of view is also recorded.

3.3 Eliminating the challenges

A key to coming up with a good object detection algorithm is to identify the potential challenges that will be faced during testing and incorporate methods to eliminate those challenges. In this section, we will describe how the algorithm tackled the different hurdles faced when performing object detection on water surfaces.

Partial occlusions of the object and drift due to the water currents is one challenge to be overcome. Tracking algorithms are not able to handle such occlusions and do not recover successfully when the object momentarily leaves the scene and comes back. A low-cost detection algorithm was developed instead. Generally, detectors are more CPU expensive than tracker. However, since minimal processing techniques were used, the CPU consumption was within the limits of the onboard computer.

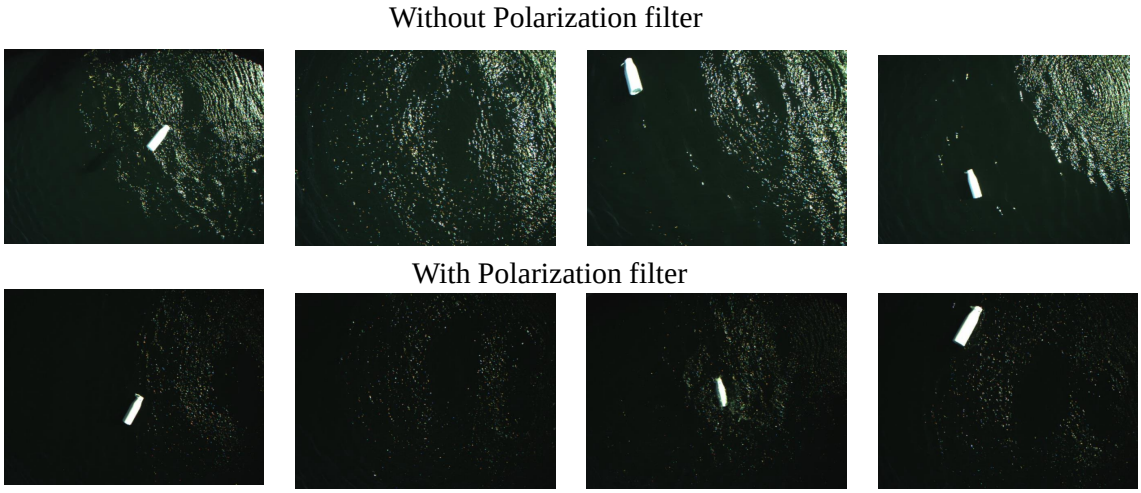


Figure 12: Reducing the effect of reflections using a linear polarization filter. Top: Snapshots of videos collected without polarization filter; Bottom: Snapshots of videos collected with polarization filter

As described in the first chapter, another major challenge when performing object

detection on water surfaces is the surface reflections and the reflections on the object. These reflections obscured the object and complicated the detection algorithm. To prevent this, we attaches a linear polarization filter in front of the camera lens which reduced the reflections and helped in enhancing the features of the object. The difference between not using a polarization filter and using a polarization filter is shown in Figure 12. The figure demonstrates two videos collected over a white bottle, one without a polarization filter and the other with a polarization filter. However, since the linear polarization filter could only subdue reflections coming in at a small range of angle and hence a few specular reflections on the water surface remained, especially during sunny days. To subdue these further, we incorporate a specularity removal technique in our object detection pipeline.

Objects of lower reflectivity have a lower contrast with the background, making them harder to detect. Additionally, when testing outdoors, multiple illumination conditions will be encountered. This would mean that the object detection algorithm must be independent of features that rely on the intensity of light, such as the brightness pixels. As the multirotor starts descending for collection of the object, the aspect ratio of the object changes. This called for the need of an algorithm that could handle scale changes and rotations. To tackle all these issues, we needed to look for features that would not be affected by changes in brightness, scale and rotations. Since the objects that we had tested on had well defined shapes, we used an edge-based detection method. The algorithm is further described in detail in the next section.

3.4 The algorithm

A two-phase object detection algorithm was developed and tested in multiple test videos. The algorithm had an estimation phase and a detection phase. The estimation is initially run and it starts looking for the object to be detected. Once it confirms that the object has been found, the algorithm proceeds to the second phase and performs specularity removal along with detection. The detection is then used for tracking of the object and to deliver the bounding box coordinates to the multirotor.

One of the hardest tasks was performing specularity removal in combination with object detection. A literature survey on different reflection removal algorithms and implemented them to test their performance and compatibility with our algorithm. [54] is one of the first approaches that we implemented. This work treats specular pixels as noise and attenuates them using a low-pass bilateral filter to remove them. While this approach had a good performance, the iterative use of the bilateral filter consumed too much of CPU resource and hence could not be used for real-time applications on videos.

The approach developed in [55] was then implemented, which computes the local intensity ratios of each pixel across the RGB channel and then, makes those ratio equal in an iterative fashion. Since this approach also uses an iterative framework and does not eliminate substantial reflections as we wanted, it is not feasible to implement this approach with our algorithm for the similar reasons as described in the previous paragraph.

Algorithm 2 Object detection algorithm

0: **while** read polarized camera frame **do**
0: **if** *Object detected* $== false$ **then**
0: $f \leftarrow getframe$
0: $Canny\_edge\_detector(f)$
0: $contours(i) = FindContours$(f)
0: $Obj = max\_area\{contours(i)\}$ {**Find contour with maximum area as we assume only one object in the scene**}
0: **if** No contour jumps in a window of 10 frames **then**
0: *Object detected* $\leftarrow true$ {**Store the initial bounding box coordinates**}
0: **else**
0: *Object detected* $\leftarrow false$
0: **end if**
0: **end if**
0: **if** *Object detected* $== true$ **then**
0: $f \leftarrow getframe$
0: $I_{min}(p) = \min\{f_R(p), f_G(p), f_B(p)\}$ {**Minimum intensity value of each pixel p across the RGB channels**}
0: $T = \mu_v + \eta * \sigma_v$ {**Compute intensity threshold value**}
0: $\tau(p) = \begin{cases} T, & \text{if } I_{min}(p) > T \\ I_{min}(p), & \text{otherwise} \end{cases}$ {**Compute offset to find pixels to be changed**}
0: $\hat{\beta}(p) = \begin{cases} 0, & p \text{ within bounding box} \\ I_{min}(p) - \tau(p), & \text{otherwise} \end{cases}$ {**Compute the specular component**}
0: $f_{sf}(p) = merge\{f_R(p) - \hat{\beta}(p), f_G(p) - \hat{\beta}(p), f_B(p) - \hat{\beta}(p)\}$ {**Subtract specular component from each of the three channels and merge them to get the specular-free image**}
0: $Canny\_edge\_detector(f_{sf})$ {**Detect edges on the specular-free image where object is not affected by specular reflections removal algorithm**}
0: $contours(i) = FindContours(f_{sf})$
0: $Obj = max\_area\{contours(i)\}$ {**Update the bounding box coordinates**}
0: **if** Object not detected for more than 10 frames **then**
0: *Object detected* $\leftarrow false$
0: **end if**
0: **end if**
0: **end while**=0

We then implemented the algorithm discussed in [56], which is based on the fact that reflections can be separated into their diffuse and specular components. We compute a specular component of the frame and then subtract it from the original frame to get the specular free image, which is free of reflections. Subsequently, we employed this algorithm in our code due to its low computational load and fast processing speed. To account for objects of different reflective properties, we have used an edge-based contour detection algorithm by initially estimating the edges of

the object to be detected. The complete algorithm is described in detail below and the visual representation is given by Figure :

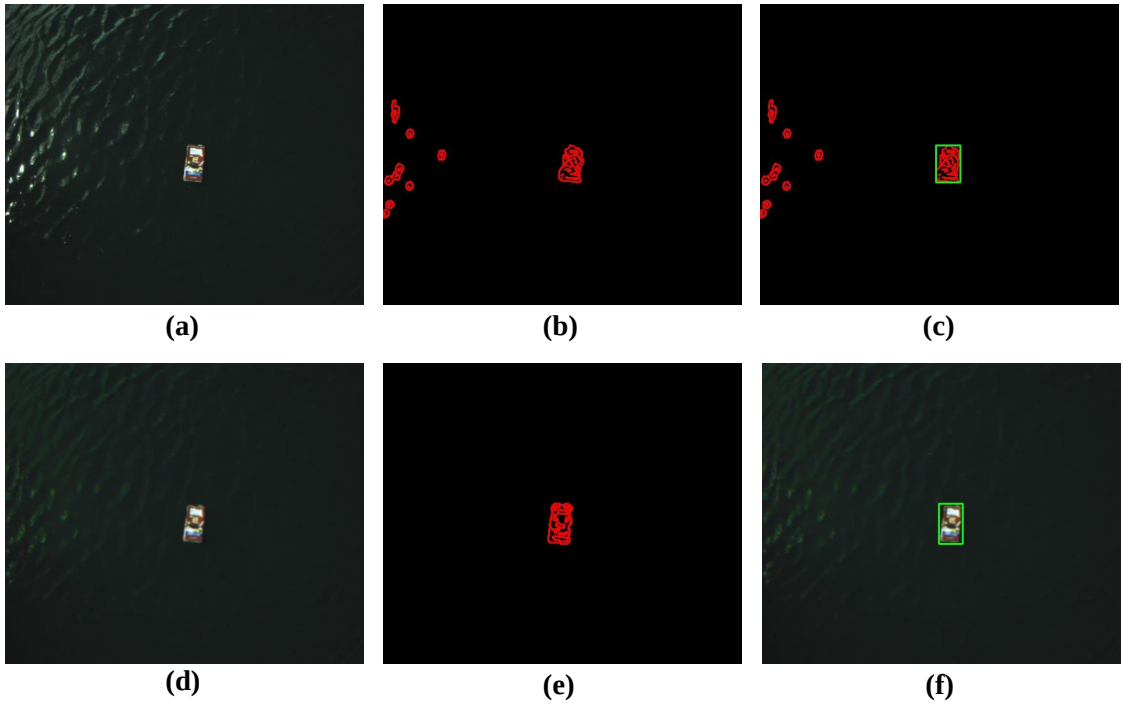


Figure 13: The two phase object detection algorithm. Top row: Estimation phase. (a) Current polarized video frame; (b) Contours extracted from the frame; (c) Bounding box drawn over largest consistently placed closed contour. Bottom row: Detection phase. (d) Specularity removal from all pixels outside bounding box; (e) Extraction of closed contours; (f) Detection of the object

- **Read polarized video frame (Line 1):** In the first phase of the algorithm, a thin filament of polarization filter is still used in front of the camera lens to suppress a major portion of the specular reflections. The polarized frame is shown in Figure 13 (a).
- **Estimate initial position of object (Line 3-11):** The polarized video feed is passed on to a canny edge detector which is used to extract closed contours from the frame which is seen in Figure 13 (b). A dilation operation is performed

on the edges detected by the canny edge detector. Dilation is a morphological operation [57] that is used to increase the thickness of the edges for better visibility. The operation consists of convolving the image with a kernel that scans a over the image and computed the pixel with the maximum value inside the overlap. It then replaces the value of the anchor pixel with the maximum value within the overlap. The dilation operation is given by -

$$\texttt{dst}(x, y) = \max_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x + x', y + y'),$$

where *dst* is the output image, *src* is the input image, x and y represent the pixel positions in the row and column respectively.

After this operation, a contour search method is employed to find the largest closed contour. However, due to the presence of reflections on the water surface, the largest closed contour can either be an object or a reflection. Since the reflections are volatile, their contours aren't consistently placed in the same region. If the largest closed contour is consistently placed in the same region for atleast 10 consecutive frames, it indicates the presence of an object in the scene, as shown in Figure 13 (c).

- **Store current bounding box coordinates (Line 8):** Once we get the initial closed contour of the object we compute its current bounding box coordinates. To ensure reliable object detection, we remove reflections from the next frame by considering the bounding box coordinates of the object in the current frame. By doing this, we guarantee that the reflection removal algorithm removes specular component of the entire image without affecting the features of the object.
- **Removing specular reflections (Line 15-19):** For the second phase of the algorithm, we employ the work by [56] starts by computing the minimum

intensity value $I_{min}(p)$ across the RGB channel for each pixel in the frame. If each RGB frame is of the size 640×480×3, I_{min} is of the size 640×480. An intensity threshold value T is then calculated to distinguish highlighted pixels:

$$T = \mu_v + \eta \cdot \sigma_v \tag{3.1}$$

where μ_v and σ_v are the mean and standard deviation of the minimum intensity value matrix I_{min} over all the pixels. The variable η is a measure of the specular degree of an image and is generally set to be 0.5. Based on the calculated intensity threshold T, an offset τ is then computed to determine which pixels have to be changed to suppress reflections.

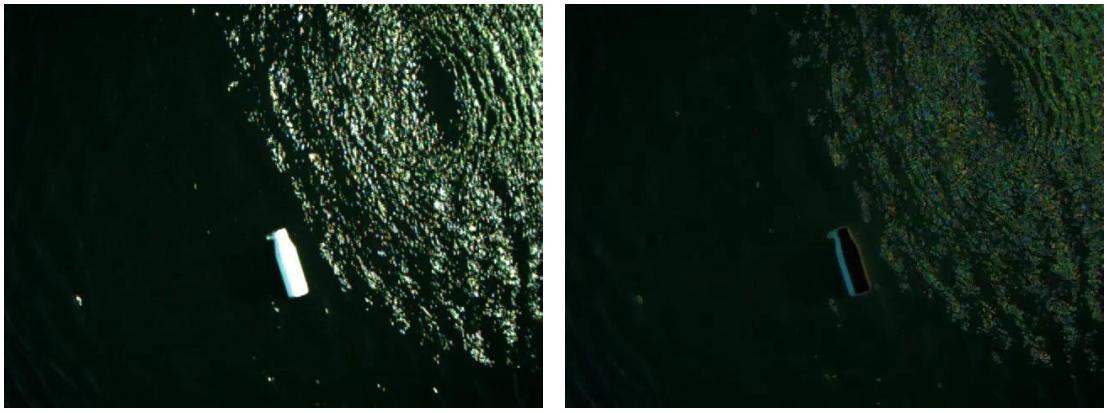
$$\tau(p) = \begin{cases} T, & \text{if } I_{min}(p) > T \\ I_{min}(p), & \text{otherwise} \end{cases} \tag{3.2}$$

The specular component $\hat{\beta}(p)$ of the frame is computed by subtracting the offset from I_{min}. For any pixel inside the bounding box coordinates, we set $\hat{\beta}(p)$ to 0 to preserve the features of the object to be detected.

$$\hat{\beta}(p) = \begin{cases} 0, & p \text{ within bounding box coordinates} \\ I_{min}(p) - \tau(p), & \text{otherwise} \end{cases} \tag{3.3}$$

Finally, the specular component is subtracted from the frame to get the specular-free image without removing reflections from the area where the object is located, which is demonstrated in Figure 13 (d). Since the object moves due to drift from one frame to the next, we scale the bounding box size by a small amount before applying the specularity removal over the rest of the pixels in the frame. The specularity removal is applied on the entire frame excluding the area where the object has been estimated using the bounding box coordinated. This is

because the objects have reflective surfaces as well. Since the specularity removal algorithm is eliminates reflections on all surfaces, if applied on the object, it affects the features of the object so much that it cannot be seen in the frame. The specularity removal algorithm assumes that the object is also a reflective surface to removed and tries to change the appearance of the object to match the background, resulting in a dark colour. Figure 14 demonstrates an example in which the specularity removal algorithm has been applied over the entire frame.



(a) **(b)**

Figure 14: Result of removing specular reflections over the entire frame without excluding the object. The object has been completely removed from the scene and cannot be detected, although all the surface reflections have been removed. (a) Input image (b) Output of the specularity removal algorithm.

- **Detect object and update bounding box coordinates (Line 20-26):** Since reflections have been eliminated in the rest of the frame and assuming that the object has not moved a lot within one frame, the region where the object lies is highlighted when compared with rest of the image. The contours of the specular-free image are again extracted using a canny edge detector,seen in Figure 13 (e), and the object is detected as shown in Figure 13 (f). The object

is easily detected in this phase since there are no reflections that are seen when extracting the contours. The current bounding box coordinates of the object is utilized for specularity-removal from the next frame, and the process iterates. If no contours are detected for a pre-determined time window, the algorithm reverts back to the maximum-area contour searching.

3.5 Testing

We carried out the testing of this algorithm using the videos collected in the previous sections. Tests were also conducted in real-time with the multirotor wherein, we deploy the multirotor with the net deactivated, manually fly it over the object and then start the object detection algorithm. Once the object has been estimated and the detection is initialized, we pass the bounding box coordinates of the detected object to the multirotor, which uses it to autonomously descend and lands on top of the object. The net system is then activated to pick up the object from underneath and the multirotor carries it back to the shore. Using the two-phase detection algorithm, we were able to achieve an overall success rate of 91.6%. A successful test run demonstrating the object collection from a water surface using the multirotor is shown in Figure 15.

Figure 16 demonstrates the working of the object detection algorithm for all the 5 objects at four different frames of the video. We can easily infer the accuracy of our detection and the similarity of the bounding box with the ground truth. We can also see the different environmental conditions such as the brightness of light, size of the object and the water ripples which factored in the videos for each of the object. The combined working of the polarization filter in front of the camera lens and the

a) Object detected b) Multirotor descends c) Multirotor lands on object d) Object picked up

Figure 15: Successful collection of an object from water surface by the multirotor

specularity removal algorithm has decreased the contrast of the background water surface by a lot, while also highlighting the features of the object. However, failure cases were seen when testing dark can. Since the video collected using the dark can was during a cloudy illumination condition, the edges object had low visibility, as the background was also quite dark. In addition to this, using the polarization filter and the specularity removal algorithm reduces the contrast of the image, which also resulted in the dark can being masked by the background in a number of frames. While the detector does not through any false positives, it failed to detect the object for a number of frames and reverted back to estimation even though the object was in the frame. This caused a decrease in the number of successful collection of the dark can by the multirotor.

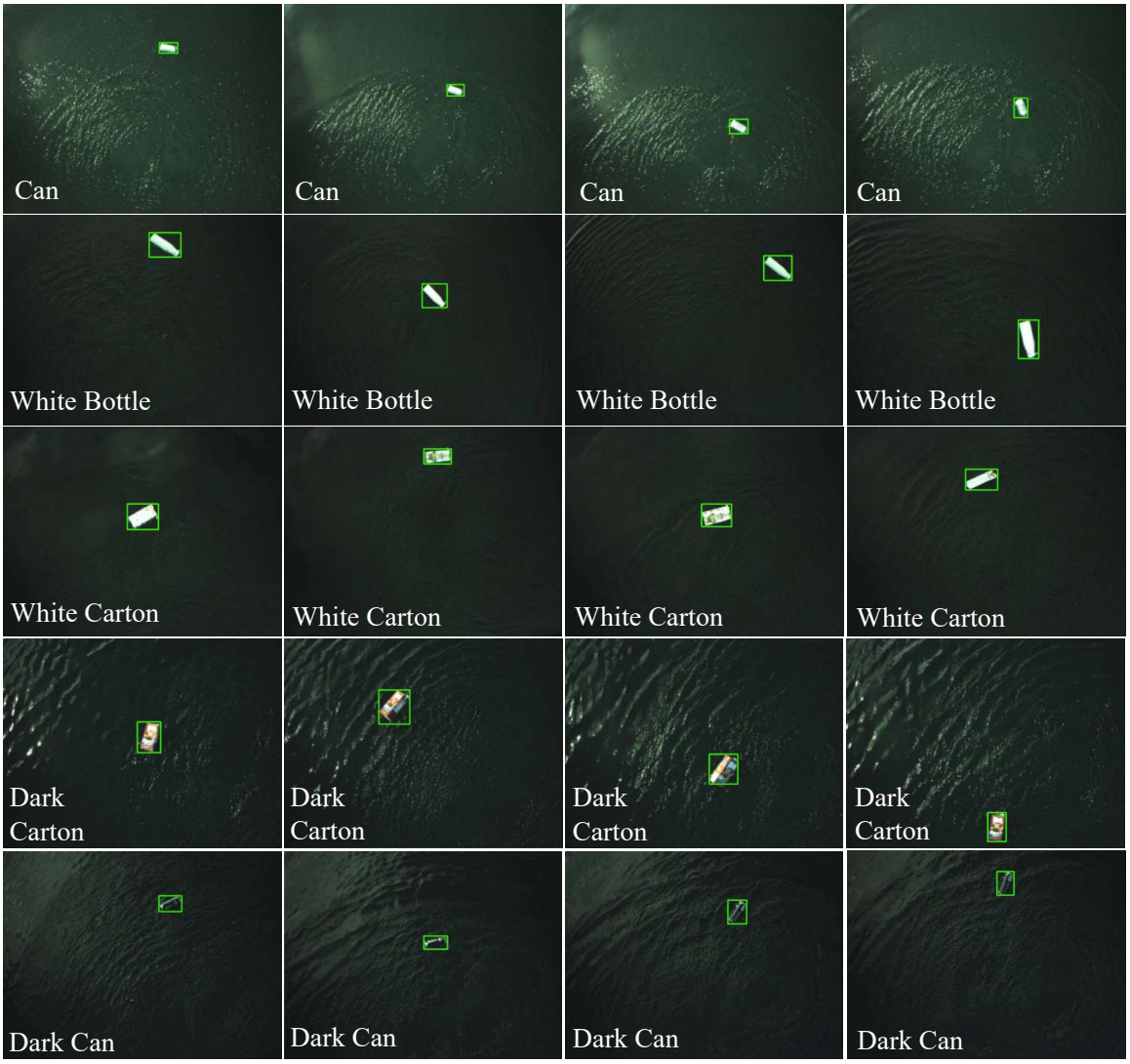


Figure 16: Proposed object detection algorithm's output on a cloudy day. Snapshots from a video are shown at different time instances for the following objects: (a) Row 1: Can; (b) Row 2: White bottle, (c) Row 3: White Carton, (d) Row 4: Dark Carton & (e) Row 5: Dark Can.

3.6 Limitations

Although the developed algorithm had a better accuracy and faster working than the previous algorithm, it still has a few limitations that need to be overcome.

1. Object estimation is not always reliable

   The object estimation phase looks for closed contours that are placed consistently in the scene for a pre-defined number of frames. This approach is not always reliable and can cause failure. If there is no object in the scene but there are other features that are consistently in the scene, the object estimation would mistake those to be the object instead and initialize the detection.

2. Specularity removal can cause failure

   The removal of specular reflections from the frame can cause a number of issues. If the bounding box provided by the estimation does not overlap the object, the specularity removal would regardless still be applied to all the rest of the pixels in the frame. The contours of the features inside the bounding box, usually the water ripples and reflections, would now be extracted and detected. The detection would not terminate and would instead proceed detecting the ripples.

3. Darker objects are harder to detect

   As was discussed in the previous section, because of the decrease in contrast of the video due to the application of the polarization filter and the specularity removal, the features of objects with darker shades becomes harder to detect since they match the shade of the background. The estimation phase would continue running for long periods until it has been able to extract the contours of the object. If the detection phase is initialized and the object drifts away from the field-of-view of the camera, the estimation phase is again run for a long period when the object comes back into the scene.

4. Lack of tracking the object

   As is the case with all detectors, they are quite different from tracking. Object detection cannot remember the object being detected. For instance, if the object

being detected drifts away from the frame and another object comes into the scene instead, the detector would start detecting the new object in the scene instead. This is not the case with object tracking since they remember the features of the object being tracker and would not start tracking other objects. Object trackers can be used to assign unique ID for different objects so as to identify and distinguish them from one another.

Chapter 4

EXPERIMENTAL RESULTS

In this chapter, we discuss our the performance comparison of our detection algorithm with other detectors such as You Only Live Once v3 (YOLOv3) [58] and Mobilenet Single Shot Detector (SSD) [59]. These deep-learning models are both used widely in the field of computer vision for real-time object detection and hence they were chosen for comparison with our algorithm. Both YOLOv3 and SSD have been pre-trained on the COCO dataset [60], which is a large scale object detection dataset containing over 200,000 labelled images and more than 80 classes. Using pre-trained deep-learning models reduces the time it takes for it to be trained on a custom dataset since it already has been updated with the model weights of the previous dataset.

4.1 The dataset

For choosing the dataset, we collected videos with 5 different objects namely - a white bottle, a white carton, a silver can, a red carton and a dark can as shown in the previous chapter under cloudy/ partially cloudy conditions. To maintain consistency, we collected all the videos at 30 FPS. The exposure and gain of the camera were set to 15 and 45 respectively. The videos were trimmed to 1 minute length to get equal number of frames for each object. The videos were then converted into individual frames. We use just the first 500 frames of each video for training and validation of the detectors, which resulted in a total of 2500 frames. The frames are collected and

then sorted randomly so that they can be split as 2000 images for the training dataset and 500 for the validation dataset.

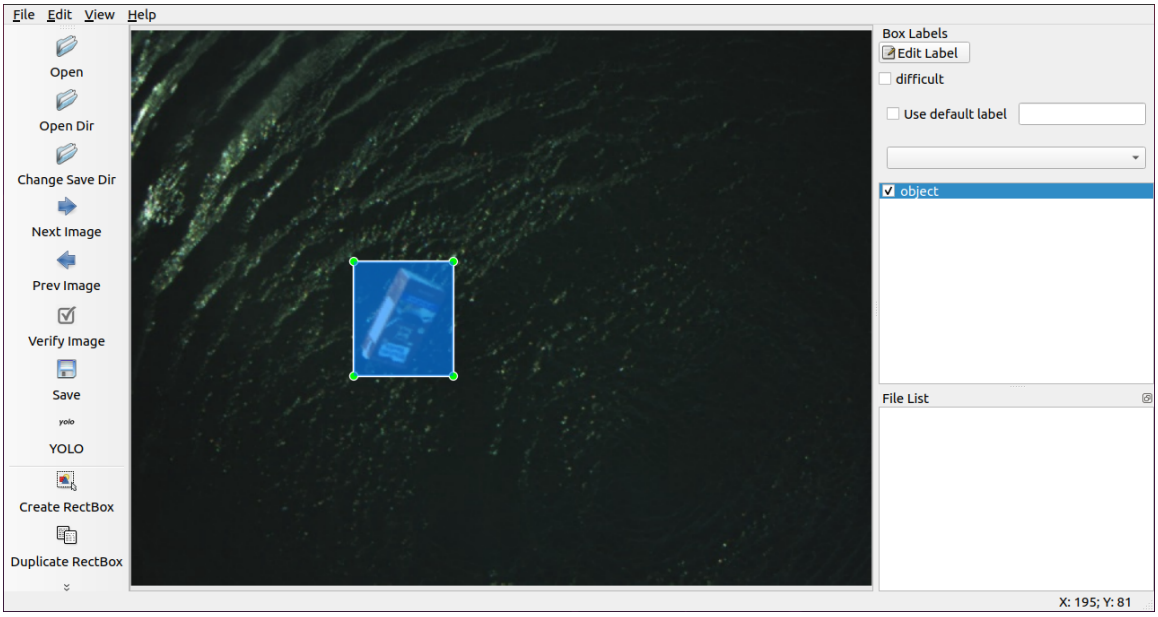


Figure 17: Annotating an image containing an object using LabelImg

The training and validation dataset were then labelled using LabelImg [61]. This is an open source graphical image annotation software that is used to prepare training dataset for deep-learning models. The images are loaded individually into LabelImg and annotated by hand. Since the objective of our algorithm is to just detect the object and not to classify it, we labelled all the frames under a single class called "object". Once the annotation is complete, the labels can be saved as XML files in PASCAL VOC format, or TXT files in YOLO format depending on the model to be trained. Annotating of an image using labelImg is shown in Figure 17.To increase the size of the training dataset, we generated synthetic data by performing various operations such as rotation, shearing and varying brightness on the training dataset. The brightness of the images was varied to capture different illumination conditions.

With the addition of the synthetic data, we increased the size of the training dataset by an additional three times from 1500 to 6000 images.

4.2 Ground truth labelling

The ground truth for all the videos was generated using Matlab's Ground Truth Labeler App. An input video is loaded and a label is drawn over the object that is to be tracked. The ground truth bounding box is then automated for the next few frames using the Kanade-Lucas-Tomasi (KLT) point tracker algorithm [62]. It works under two assumptions - The difference in time between two consecutive frames is small such that there is no large displacement of the object and each frame contains objects of different textures that smooth changing shades of gray on grayscaling it. Under these assumptions, it uses the Shi-Tomasi [63] algorithm to detect a set of good features to track in the frame. These features are then tracked over the frames by using optical flow vectors.

However, this algorithm fails when the object leaves the frame and so we have to annotate the ground truth manually for a few frames. The output is given as the top-left x, y coordinates, the height and width of the ground truth bounding box. This data is then compared with the bounding box parameters of the tracker output for evaluation. The function takes three parameters - the output bounding boxes of the tracker, the ground truth bounding box and a optional overlap threshold. The metrics were computed by using a overlap ratio of 0.3 since the multirotor could detect the object satisfactorily even when the overlap ratio is 0.3. Overlap ratio is computed as the intersection over union (IoU) between the bounding boxes. Intersection over union is defined as the ratio between the area of overlap between two bounding boxes

and the total area of the bounding boxes. Figure 18 demonstrated the ground truth generation for the video of the white bottle.

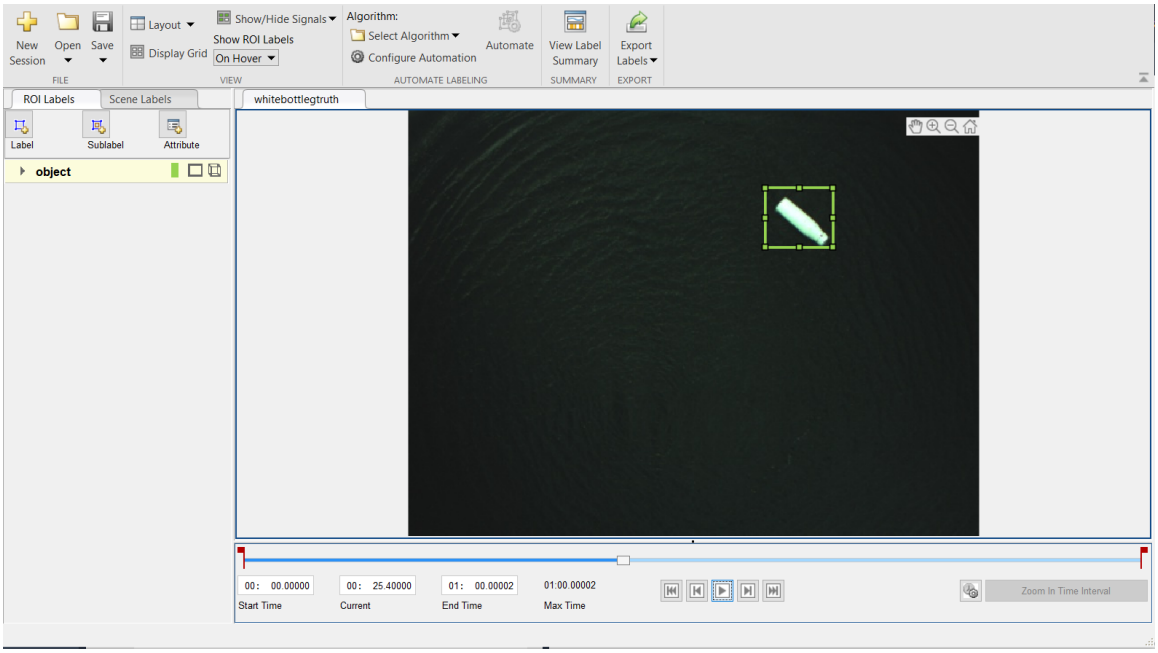


Figure 18: Labelling the ground truth using MATLAB's Ground Truth Labeler App

4.3 Training of detectors for comparison

Using the custom training data described in the previous section, we again train the COCO pre-trained YOLO and SSD. The training images were resized to 416×416 and the learning rate was set to 0.001. The momentum used was 0.9 and a batch size of 64 was initialized. The learning rate of the optimization algorithm was decreased by a fixed factor using a dacay rate of 0.0005. Both YOLO and SSD were trained on the dataset until they reached their average losses stabilized at 0.13 and 3.1 respectively. Once the training is complete, the models were validated against the validation images to make sure its performance is ready for testing.

4.4 Evaluation of the algorithm

We compare the algorithms by running them on a Intel Core i7-10750H CPU with 10 GB RAM and 4 cores. We do not utilize GPU for any of the testing, as the Intel Upboard embedded processor we utilize for real-time testing only uses CPU for running these algorithms. The metrics used for comparison are the processing speed in frames-per-second (FPS), the total CPU consumption in %, Precision, and Recall. To ensure accurate detection of the object, the estimation should run at a high rate so that the algorithm can track fast-moving objects efficiently. This can only be achieved by a high FPS. For computing the FPS, we initiate a timer when the frame is first processed, and the end where the processing is done. The difference between the time taken from the start to the end is the time taken to process one frame. The inverse of this is the FPS. We also need low CPU consumption to reduce the power consumption and subsequently improving the flight time. To simulate the working on real-time conditions, we restrict the CPU usage to just 1 CPU core. As is evident from Table 1, our algorithm is a clear choice when comparing the CPU consumption and processing speed. The average CPU consumption for our algorithm is 37.5%, whereas the processing speed is 21 FPS. Our proposed algorithm has the lowest CPU consumption and highest FPS in comparison because our detection algorithm neither employs a large model or performs several complex operations on the input image, as is the case in most deep-learning models. The YOLO framework on the other hand, utilizes 86.25% of the total CPU availability. The processing speed of YOLO is also not suitable for real-time since it is only 2.8 FPS. SSD had a better processing speed of 11 FPS when compared to YOLO, but it also had a high CPU consumption of 75%.

Precision is defined as the ratio of true positives to all positive instances, based on

Table 3: Performance comparison of different detectors

Objects	**Average FPS**			**Total CPU usage in %**			**Precision & Recall**		
	Ours	**YOLO**	**SSD**	**Ours**	**YOLO**	**SSD**	**Ours**	**YOLO**	**SSD**
White Bottle	21	2.8	11	37.5	86.25	75	0.96	0.97	0.57
Dark carton	21	2.8	11	37.5	86.25	75	0.91	0.96	0.52
Dark can	21	2.8	11	37.5	86.25	75	0.78	0.90	0.22
Silver can	21	2.8	11	37.5	86.25	75	0.91	0.90	0.86
Juice carton	21	2.8	11	37.5	86.25	75	0.97	0.96	0.80

the ground truth. Recall is defined as the ratio of true positives to the sum of true positives and false negatices, based on the ground truth.

$$Precision = \frac{TP}{TP + FP},\ Recall = \frac{TP}{TP + FN}$$

Precision and Recall are both equal in our case since the test videos and the ground truth video had an equal number of frames. The YOLO model had the highest precision and recall metrics for most of the objects. The precision and recall rates of our algorithm are slightly lower than YOLO except in two cases. For the dark can, the performance of our algorithm is lower since it doesn't detect the object on all the frames. This is because the visibility of the dark can had reduced since the background was dark as well. In the case of the silver can and juice carton, our algorithm actually has a slightly better performance. SSD had the lowest precision and recall metrics due to its inability to detect the objects for a large number of frames. It is also to be noted that the precision and recall values of none of the deep learning models is ever equal to 1 because they also fail to detect the object for a few frames. Considering all these metrics and the fact that testing has to be carried out in real-time on a portable processor, our proposed method works best for this scenario.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

To conclude, a two-phase detection strategy for robust detection of floating trash objects on water surfaces has been developed. The algorithm has a fast processing speed with low-computational expense, which is suitable to be deployed on on-board processors of robots in real-time. To reduce the effect of reflections during detection, we have used a linear polarization filter in front of the camera lens and have also applied a specularity removal algorithm. It has been demonstrated that the combination of both have eliminated the effect of reflections to the maximum.

The algorithm has an estimation phase to locate the object and a detection phase to initialize the detection. In the estimation phase, we look for location of the object by extracting out closed contours that stay in the scene for a defined number of frames. The detection phase uses the bounding box coordinates provided by the estimation phase to apply a specularity removal algorithm for all the pixels outside the bounding box. The object is then detected by extracting the contours again and computing the bounding box. The updated bounding box coordinates are stored for specularity removal in the subsequent frame and the code is repeated.

The detection algorithm has been compared with other deep-learning detectors on 5 different objects of varying shapes and sizes, and the performance comparison was shown. The performance metrics used were the FPS, CPU usage, precision and recall. Our object detection algorithm had the highest FPS of 21 and the lowest average

CPU consumption of 37.5%. The precision and recall metrics were also comparably high at 90.6%. The above data demonstrated how our algorithm was the best suited for real-time object detection than the rest.

5.2 Future work

Future work in the area method of object detection would be to increase the robustness of the current algorithm. Testing the algorithm on more objects in presence of harsher conditions and conducting real-time field testing with the multirotor would bring up more challenges and solving them can lead to a development of a better detector. Particularly, we need to look at eliminating the limitations of the algorithm discussed in the previous chapter. [64] describes a method for automatically adjusting camera exposure using the gradient information available in the scene. This was used to solve the issue of under/over-saturation in outdoor robotics experiments. In our case, such an auto-exposure algorithm can be applied to find the best exposure value that suppresses reflections and also maximize the object features.

Using background subtraction techniques for motion estimation did not work for our case since the water surface is a dynamic background and not static. However, the work presented in [65] demonstrates the use of adaptive background mixture models to model each pixel in the background as a mixture of Gaussians and updating the model by an online approximation technique. The pixels are then classifies using a Gaussian distribution to determine which fits the background the most. A real-time tracker that can handle light changes, cluttering motion and scene changes has also been demonstrated in this work.

Estimation of the object by extracting consistently placed contours is not always

reliable and robust. Detecting specular reflections in the video feed and avoiding those regions instead would be a more practical approach. [66] proposes a method for detection of specular reflections by analysing the Intensity-Saturation (IS) histogram obtained from a HSI color space of the image. Applying such a technique for our algorithm can eliminate the use of an estimation phase and would increase the robustness of our algorithm.

Using an RGB-D camera can also prove very beneficial for object detection of a more diverse range of trash objects such as transparent objects and objects with less defined features. The advantages of using depth cameras for computer vision applications has been discussed in more detail in [67]. [68] proposes an classification algorithm that uses SIFT features to recognize transparent objects in the scene using a RealSense depth camera. A monocular camera on the other hand, cannot obtain depth information from the frame and detect transparent objects. Since we see a number of transparent objects such as plastic bottle and bags floating on the surface of water, using a depth camera can improve the detection capabilities of the algorithm.

Another pressing issue to be considered, is the detection and tracking of multiple objects. When employed on a multirotor, it is power and time consuming to detect single objects and perform the entire process each time. Detection and tracking of multiple objects can save time and battery power. Most multi-object tracking approaches have the disadvantage of applying the tracking-by-detection technique which uses two models for target localization and data association respectively. However, having two models causes issues in real-time due to the increased CPU consumption. The proposed system in [69] uses one single model for both target localization and data association for simultaneous learning, resulting in increased efficiency. It also works at a relatively high FPS and has comparable accuracy to the state-of-the-art trackers.

Incorporating such a multi-object tracker in our algorithm can drastically improve the working of the system while also having a greater impact on the environment during trash collection.

REFERENCES

[1] Soharab Hossain Shaikh, Khalid Saeed, and Nabendu Chaki. "Moving object detection using background subtraction". In: *Moving object detection using background subtraction*. Springer, 2014, pp. 15–23.

[2] Guoli Wang, Xinchao Wang, Bin Fan, and Chunhong Pan. "Feature extraction by rotation-invariant matrix representation for object detection in aerial image". In: *IEEE Geoscience and Remote Sensing Letters* 14.6 (2017), pp. 851–855.

[3] Anshuman Agarwal, Shivam Gupta, and Dushyant Kumar Singh. "Review of optical flow technique for moving object detection". In: *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE. 2016, pp. 409–413.

[4] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. "Object detection with deep learning: A review". In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[5] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. "A multi-sensor fusion system for moving object detection and tracking in urban driving environments". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1836–1843.

[6] Javier Barandiaran, Berta Murguia, and Fernando Boto. "Real-time people counting using multiple lines". In: *2008 Ninth International Workshop on Image Analysis for Multimedia Interactive Services*. IEEE. 2008, pp. 159–162.

[7] Marian Stewart Bartlett, Gwen Littlewort, Ian Fasel, and Javier R Movellan. "Real time face detection and facial expression recognition: development and applications to human computer interaction." In: *2003 Conference on computer vision and pattern recognition workshop*. Vol. 5. IEEE. 2003, pp. 53–53.

[8] Nils Tijtgat, Wiebe Van Ranst, Toon Goedeme, Bruno Volckaert, and Filip De Turck. "Embedded real-time object detection for a UAV warning system". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2110–2118.

[9] Sarah Babbitt, Tanya Gatsak, and Bhashyam Balaji. "Challenges in object detection in above-water imagery". In: *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVIII*. Ed. by Ivan Kadar, Erik P. Blasch, and

Lynne L. Grewe. Vol. 11018. International Society for Optics and Photonics. SPIE, 2019, pp. 371–382. URL: https://doi.org/10.1117/12.2518879.

[10] Shengke Wang, Changyin Yu, Yujuan Sun, Feng Gao, and Junyu Dong. "Specular reflection removal of ocean surface remote sensing images from UAVs". In: *Multimedia Tools and Applications* 77.9 (2018), pp. 11363–11379.

[11] Xiaojie Guo, Xiaochun Cao, and Yi Ma. "Robust separation of reflection from multiple images". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2187–2194.

[12] Lili Zhang, Yi Zhang, Zhen Zhang, Jie Shen, and Huibin Wang. "Real-time water surface object detection based on improved faster R-CNN". In: *Sensors* 19.16 (2019), p. 3523.

[13] Gang Cao, Lihui Huang, Huawei Tian, Xianglin Huang, Yongbin Wang, and Ruicong Zhi. "Contrast enhancement of brightness-distorted images by improved adaptive gamma correction". In: *Computers & Electrical Engineering* 66 (2018), pp. 569–582.

[14] Bingbing Zhang, Xiaojun Qian, Rui Yang, and Zhen Xu. "Water Surface Target Detection Based on Improved YOLOv3 in UAV Images". In: *2021 9th International Conference on Communications and Broadband Networking*. 2021, pp. 47–53.

[15] Dilip K Prasad, Deepu Rajan, Lily Rachmawati, Eshan Rajabally, and Chai Quek. "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: a survey". In: *IEEE Transactions on Intelligent Transportation Systems* 18.8 (2017), pp. 1993–2016.

[16] Shatadal Mishra, Todd Rakstad, and Wenlong Zhang. "Robust attitude control for quadrotors based on parameter optimization of a nonlinear disturbance observer". In: *Journal of Dynamic Systems, Measurement, and Control* 141.8 (2019).

[17] Shatadal Mishra and Wenlong Zhang. "A disturbance observer approach with online Q-filter tuning for position control of quadcopters". In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 3593–3598.

[18] Dangli Yang, Shatadal Mishra, Daniel M Aukes, and Wenlong Zhang. "Design, planning, and control of an origami-inspired foldable quadrotor". In: *2019 American Control Conference (ACC)*. IEEE. 2019, pp. 2551–2556.

[19] Karishma Patnaik, Shatadal Mishra, Seyed Mostafa Rezayat Sorkhabadi, and Wenlong Zhang. "Design and Control of SQUEEZE: A Spring-augmented QUadrotor for intEractions with the Environment to squeeZE-and-fly". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 1364–1370.

[20] Jakub Mirijovsk and Jan Brus. "Utilization of a small-format aerial photography from drone pixy concept in the evaluation of the landscape changes". In: *International Multidisciplinary Scientific GeoConference: SGEM* 2 (2011), p. 345.

[21] Marc P Narkus-Kramer. "Future demand and benefits for small unmanned aerial systems (UAS) package delivery". In: *17th AIAA Aviation Technology, Integration, and Operations Conference*. 2017, p. 4103.

[22] Yunus Karaca, Mustafa Cicek, Ozgur Tatli, Aynur Sahin, Sinan Pasli, Muhammed Fatih Beser, and Suleyman Turedi. "The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations". In: *The American journal of emergency medicine* 36.4 (2018), pp. 583–588.

[23] Guillermo Heredia, AE Jimenez-Cano, I Sanchez, Domingo Llorente, V Vega, J Braga, JA Acosta, and Anıbal Ollero. "Control of a multirotor outdoor aerial manipulator". In: *2014 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2014, pp. 3417–3422.

[24] Paul EI Pounds, Daniel R Bersak, and Aaron M Dollar. "Grasping from the air: Hovering capture and load stability". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 2491–2498.

[25] Suk-Jun Kim, Dae-Young Lee, Gwang-Pil Jung, and Kyu-Jin Cho. "An origami-inspired, self-locking robotic arm that can be folded flat". In: *Science Robotics* 3.16 (2018).

[26] Shatadal Mishra, Dangli Yang, Carly Thalman, Panagiotis Polygerinos, and Wenlong Zhang. "Design and control of a hexacopter with soft grasper for autonomous object detection and grasping". In: *Dynamic Systems and Control Conference*. Vol. 51913. American Society of Mechanical Engineers. 2018, V003T36A003.

[27] Shatadal Mishra, Karishma Patnaik, YiZhuang Garrard, Zachary Chase, Michael Ploughe, and Wenlong Zhang. "Ground Trajectory Control of an Unmanned Aerial-Ground Vehicle using Thrust Vectoring for Precise Grasping". In: *2020*

IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). 2020, pp. 1270–1275. DOI: 10.1109/AIM43001.2020.9158943.

[28] Shatadal Mishra, Danish Faraaz Syed, Michael Ploughe, and Wenlong Zhang. "Autonomous Vision-guided Object Collection from Water Surfaces with a Customized Multirotor". In: *IEEE/ASME Transactions on Mechatronics* (2021).

[29] N. Kanopoulos, N. Vasanthavada, and R.L. Baker. "Design of an image edge detection filter using the Sobel operator". In: *IEEE Journal of Solid-State Circuits* 23.2 (1988), pp. 358–367. DOI: 10.1109/4.996.

[30] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[31] Xin Wang. "Laplacian operator-based edge detectors". In: *IEEE transactions on pattern analysis and machine intelligence* 29.5 (2007), pp. 886–890.

[32] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[33] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[34] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. "BRISK: Binary robust invariant scalable keypoints". In: *2011 International conference on computer vision*. Ieee. 2011, pp. 2548–2555.

[35] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[36] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. "KAZE features". In: *European conference on computer vision*. Springer. 2012, pp. 214–227.

[37] Francine Catté, Pierre-Louis Lions, Jean-Michel Morel, and Tomeu Coll. "Image selective smoothing and edge detection by nonlinear diffusion". In: *SIAM Journal on Numerical analysis* 29.1 (1992), pp. 182–193.

[38] Pietro Perona and Jitendra Malik. "Scale-space and edge detection using anisotropic diffusion". In: *IEEE Transactions on pattern analysis and machine intelligence* 12.7 (1990), pp. 629–639.

[39] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001.* Vol. 1. Ieee. 2001, pp. I–I.

[40] Richard O Duda and Peter E Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[41] Wang Zhiqiang and Liu Jun. "A review of object detection based on convolutional neural network". In: *2017 36th Chinese Control Conference (CCC).* IEEE. 2017, pp. 11104–11109.

[42] Helmut Grabner, Michael Grabner, and Horst Bischof. "Real-time tracking via on-line boosting." In: *Bmvc.* Vol. 1. 5. Citeseer. 2006, p. 6.

[43] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. "Visual tracking with online multiple instance learning". In: *2009 IEEE Conference on computer vision and Pattern Recognition.* IEEE. 2009, pp. 983–990.

[44] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. "High-speed tracking with kernelized correlation filters". In: *IEEE transactions on pattern analysis and machine intelligence* 37.3 (2014), pp. 583–596.

[45] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Tracking-learning-detection". In: *IEEE transactions on pattern analysis and machine intelligence* 34.7 (2011), pp. 1409–1422.

[46] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Forward-backward error: Automatic detection of tracking failures". In: *2010 20th international conference on pattern recognition.* IEEE. 2010, pp. 2756–2759.

[47] Alan Lukezic, Tomas Vojir, Luka ˇCehovin Zajc, Jiri Matas, and Matej Kristan. "Discriminative correlation filter with channel and spatial reliability". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 6309–6318.

[48] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE computer*

society conference on computer vision and pattern recognition. IEEE. 2010, pp. 2544–2550.

[49] Shmuel Winograd. "On computing the discrete Fourier transform". In: *Mathematics of computation* 32.141 (1978), pp. 175–199.

[50] Vladimir Vovk. "Kernel ridge regression". In: *Empirical inference*. Springer, 2013, pp. 105–116.

[51] Georg Nebehay and Roman Pflugfelder. "Clustering of static-adaptive correspondences for deformable object tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2784–2791.

[52] Rui Xu and Donald Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678.

[53] Deepak Geetha Viswanathan. "Features from accelerated segment test (fast)". In: *Proceedings of the 10th workshop on Image Analysis for Multimedia Interactive Services, London, UK*. 2009, pp. 6–8.

[54] Q. Yang, J. Tang, and N. Ahuja. "Efficient and Robust Specular Highlight Removal". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.6 (2015), pp. 1304–1311. DOI: 10.1109/TPAMI.2014.2360402.

[55] K. Yoon, Y. Choi, and I. S. Kweon. "Fast Separation of Reflection Components using a Specularity-Invariant Image Representation". In: *2006 International Conference on Image Processing*. 2006, pp. 973–976. DOI: 10.1109/ICIP.2006.312650.

[56] Hui-Liang Shen and Qing-Yuan Cai. "Simple and efficient method for specularity removal in an image". In: *Applied optics* 48.14 (2009), pp. 2711–2719.

[57] AM Raid, WM Khedr, MA El-Dosuky, and Mona Aoud. "Image restoration based on morphological operations". In: *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT)* 4.3 (2014), pp. 9–21.

[58] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[59] 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*. Springer. 2016, pp. 21–37.

[60] 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*. Springer. 2014, pp. 740–755.

[61] D Tzutalin. *LabelImg. Git code*. 2015.

[62] Jae Kyu Suhr. "Kanade-lucas-tomasi (klt) feature tracker". In: *Computer Vision (EEE6503)* (2009), pp. 9–18.

[63] Tiziano Tommasini, Andrea Fusiello, Emanuele Trucco, and Vito Roberto. "Making good features track better". In: *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*. IEEE. 1998, pp. 178–183.

[64] Inwook Shim, Joon-Young Lee, and In So Kweon. "Auto-adjusting camera exposure for outdoor robotics using gradient information". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1011–1017.

[65] Chris Stauffer and W Eric 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)*. Vol. 2. IEEE. 1999, pp. 246–252.

[66] Rui Yao, Yilun Wu, Wei Yang, Xiaolin Lin, Shidan Chen, and Su Zhang. "Specular reflection detection on gastroscopic images". In: *2010 4th International Conference on Bioinformatics and Biomedical Engineering*. IEEE. 2010, pp. 1–4.

[67] Achuta Kadambi, Ayush Bhandari, and Ramesh Raskar. "3d depth cameras in vision: Benefits and limitations of the hardware". In: *Computer vision and machine learning with RGB-D sensors*. Springer, 2014, pp. 3–26.

[68] Chen Guo-Hua, Wang Jun-Yi, and Zhang Ai-Jun. "Transparent object detection and location based on RGB-D camera". In: *Journal of Physics: Conference Series*. Vol. 1183. 1. IOP Publishing. 2019, p. 012011.

[69] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. "Towards real-time multi-object tracking". In: *Computer Vision–ECCV 2020: 16th*

European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16. Springer. 2020, pp. 107–122.