Maritime RobotX Challenge 2022:
Object/Buoy Detection with Yolov5 and OpenCV Python

Sitha Sinoun

Introduction

While OpenCV can detect objects using in-built functions to perform features such as Colour Recognition, edge detection, distance estimation and shape recognition, OpenCV also has a function to load a machine learning model. Yolov5 is intended to enable computer vision to be used for anybody and not just experts in the field of machine learning. Ultralytics Yolov5 is a reliable object detection engine that enables OpenCV to call and load a custom trained machine learning model. As a vision system to detect 4 different coloured buoys were needed, Yolov5 and OpenCV used a trained dataset to identify those different coloured buoys. The final image would place a bounding a box to identify the different buoys in the form of classes and assign a confidence value to ascertain what percentage the object detected represented a buoy.

Preparing and Training the Dataset

As different buoys and other water objects are used in subsequent Maritime Robotx challenges, it is always important to obtain the correct images objects/buoys used for the current edition as these are to be labelled and annotated. Images of the correct buoys should be used and placed in the appropriate environment to mimic the correct conditions. Overall 150 pictures of buoys and a WAM-V boat were used to train the Yolov5 model. The following pictures are the buoys used in 2022 challenge.

Using pictures of objects that are not going to be at the challenge may render a Yolov5 model useless as it may not detect for example a different shaped buoy from the particular ones the model was trained to detect.

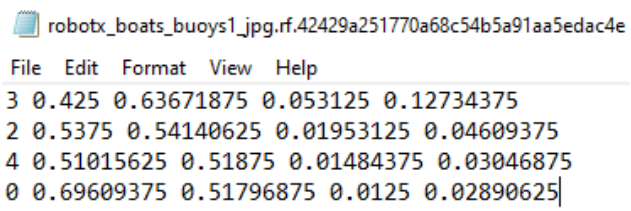
Custom training a dataset was performed using roboflow found at www.roboflow.com and a cloud based platform using a google colab notebook, this can be found here:

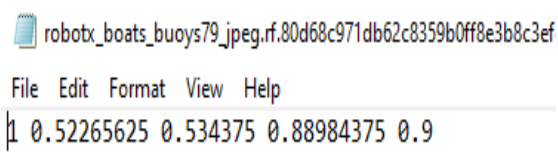
https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb

Labelling and annotating a dataset can be performed using many different resources/sites including roboflow. If annotations have already been done then simply dragging and dropping the dataset folder of buoys along with annotations means the dataset is ready to be trained by roboflow. If no annotations have been done or certain images are missing annotations, then a user will have to draw a bounding box on the desired object/buoy and assign it to a class. In the Yolov5 model created, 5 classes were made to represent the 4 different coloured buoys and were assign the names, black_buoy, red_buoy, green_buoy, white_buoy and a boat object with the name boat.

Generating a version on roboflow will split the data into three groups: test, training and validation. 70 percent of the images will be used by the Yolov5 training process to actually learn how to detect objects. The validation set uses 20 percent of the dataset which is used during training to get a sense of how well the model is doing on images that are not being used in training. Because the validation set is heavily used in model creation, it is important to hold back about 10 percent of a completely separate set of images for the test set. Choosing to export the data from roboflow into a Yolov5 PyTorch format will prep the dataset for further training in Google Colab Notebook. Colab will have all images and labels in Yolov5 format to train with correct annotations as described below:

The buoy image above and its annotation have 4 lines with each one referring to one specific buoy in the image. Let's check the lines:





The first integer number (3) is the object class id. For this dataset, the class id 0 refers to the class “red_buoy” and the class id 2 refers to the “white_buoy” class and so forth. The following float numbers are the x,y,w,h bounding box coordinates. The second picture would have a class id of 1 as that number was assigned to boat with its own bounding box coordinates.

Since Roboflow is hosting the dataset, they will give an API key to use from the version created on roboflow and this is to be used on the google colab notebook. Once again found here: https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb

```
from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("robotxsitha").project("robotx-boat-and-buoys")
dataset = project.version(3).download("yolov5")
```

Running the colab notebook and pasting the above roboflow API in the correct section will eventually lead to training a model with inference and trained weights. At the end a trained custom Yolov5 model with objects of buoys is complete and all is needed now is to export the models weight for future use. The weights can be found at runs/train/exp/weights/best.pt on the notebook and this is important because best.pt be imported into OpenCV.

The Program

The program is intended as a vision system to detect 4 coloured buoys and a boat, loading the best.pt file onto OpenCV will allow OpenCV to access the trained Yolov5 machine learning model. Using either a capture in the form of a video or live webcam stream, the results put forward should be any objects of classes that the model is trained to detect, i.e buoys.

```
import cv2
import numpy as np

# capturing the stream through a video (or webcam)
cap = cv2.VideoCapture('resources/buoy_test2.mkv') # change to 0 or 1 for webcam
# loads yolov5 dnn model of robotx data
net = cv2.dnn.readNetFromONNX('resources/best.onnx')
# loads all classes robot.onnx model can detect
file = open("resources/robotx_classes.txt", "r")
# places all the labelled classes in a python list
classes = file.read().split('\n')
print(classes)
```

Loading the Yolov5 consists of one line:

net = cv2.dnnreadNetFromONNX(‘resources/best.onnx’)

The file best.pt needs to be converted into best.onnx file format as OpenCV python can only read the yolov5 model in that format. Conversion can be done by using scripts found online and in github repositories. The converted files allow libraries that do not recognise the pytorch (pt) model to be natively imported.

```
cap = cv2.VideoCapture('resources/buoy_test2.mkv')
```

Here, a video of buoys is used for object detection, to detect objects from a video stream, place a 0 or 1 in the brackets instead.

```
# while loop to access the frame(s) of the video or webcam
while True:

    # reading the video stream to capture frames
    ret, frame = cap.read()
    # conditional statement to ascertain whether frames are being gathered
    if frame is None:
        break

    # TESTING ONLY: if frame is too big then, resize
    # frame = cv2.resize(img, (1280, 800))

    # feed our model with the images from the video or webcam
    # create a blob to preprocess images for deep learning classification and detect objects
    # scale factor normalises image pixels, mean subtraction set to zero, swapRB=true swaps BGR to RGB, no cropping
    blob = cv2.dnn.blobFromImage(frame, scalefactor=1/255, size=(640, 640), mean=[0, 0, 0], swapRB=True, crop=False)
    # provides the blob for the yolov5 robot model
    net.setInput(blob)
    # run the reference and gives Numpy ndarray as output which can be used to plot box on the given input image.
    detections = net.forward()[0]
    # FOR TESTING ONLY:  predicts 25200 bounding boxes and 85 column entries per bounding box
    print(detections.shape)
```

```
blob = cv2.dnn.blobFromImage(frame, scalefactor=1/255, size=(640, 640), mean=[0, 0, 0], swapRB=True, crop=False)
```

The images on the video in the program can't be used straight away; it needs to be converted into a blob. A blob is used to extract features from the image and to resize them. Choosing 640 x 640 gives high accuracy but slower speeds.

```
detections = net.forward()[0]
```

The detections variable displays the result of the detection and is an array that contains all relevant information on objects detected, their position and the confidence value.

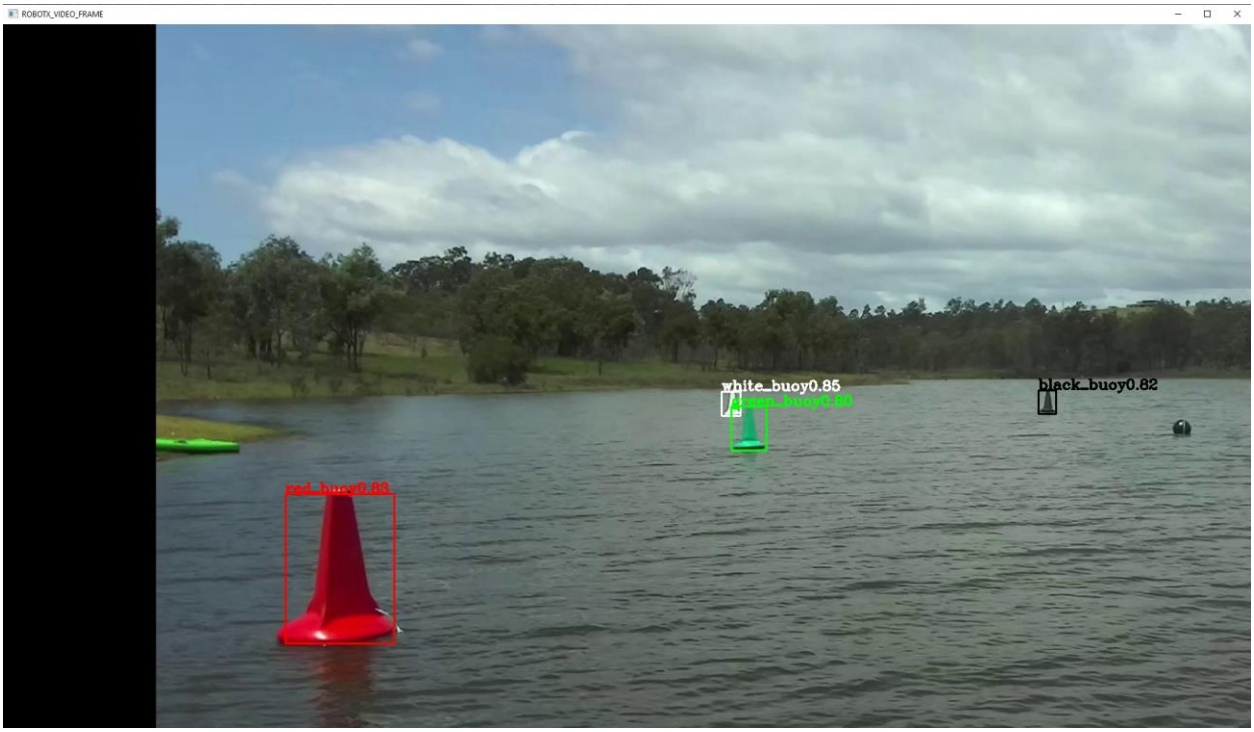
```python
# first 2 positions for e.g, cx.cy are the centre of the bounding box, w, h, confidence value, 80 class_scores
# find class_ids, confidences score, bounding_boxes
classes_ids = []
confidences = []
bounding_boxes = []
rows = detections.shape[0]

# find width and length
frame_width, frame_height = frame.shape[1], frame.shape[0]
x_scale = frame_width/640
y_scale = frame_height/640

# showing information on the screen
for i in range(rows):
    row = detections[i]
    confidence = row[4]
    if confidence > 0.5:

        # object detected
        classes_score = row[5:]
        index = np.argmax(classes_score)
        if classes_score[index] > 0.5:
            classes_ids.append(index)
            confidences.append(confidence)
            cx, cy, w, h = row[:4]

            # rectangle coordinates
            x1 = int((cx-w/2) * x_scale)
            y1 = int((cy-h/2) * y_scale)
            width = int(w * x_scale)
            height = int(h * y_scale)
            box = np.array([x1, y1, width, height])
            bounding_boxes.append(box)
```

Once detection is done, it is necessary to show results to the screen as output. In the above step, YOLOv5 performed the object detection returning all the detections found in an output 2D array. This array has 25,200 positions where each position is a 10-length 1D array. Each 1D array contains the data of one object detection. The 4 first positions of this array are the x,y,w,h coordinates of the bounding box rectangle. The fifth position is the confidence level of that detection. The 6th up to 8th elements are the scores of each class. Looping trough the detections array enables a calculation of the confidence value and an input for the confidence threshold. Not every detection out of the 25,200 are actual detections, so assigning a threshold confidence of 0.5, enables the program to detect the correct objects, otherwise it will be skipped. The threshold goes from 0 to 1. The closer to 1 means the greater the accuracy of the detection, while closer to 0 means less accuracy a greater number of the objects detected.

```
# removes redundant bounding boxes
indices = cv2.dnn.NMSBoxes(bounding_boxes, confidences, 0.5, 0.5)

for i in indices:

    x1, y1, w, h = bounding_boxes[i]
    label = classes[classes_ids[i]]
    conf = confidences[i]
    text = label + "{:.2f}".format(conf)

    # if object detected is a boat, bounding box is blue and text is pink
    if label == "boat":
        cv2.rectangle(frame, (x1, y1), (x1+w, y1+h), (255, 0, 0), 2)
        cv2.putText(frame, text, (x1, y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7, (255, 0, 255), 2)

    # if object detected is a red buoy, bounding box is red and text is red
    elif label == "red_buoy":
        cv2.rectangle(frame, (x1, y1), (x1+w, y1+h), (0, 0, 255), 2)
        cv2.putText(frame, text, (x1, y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 0, 255), 2)

    # if object detected is a black buoy, bounding box is black and text is black
    elif label == "black_buoy":
        cv2.rectangle(frame, (x1, y1), (x1+w, y1+h), (0, 0, 0), 2)
        cv2.putText(frame, text, (x1, y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 0, 0), 2)

    # if object detected is a green buoy, bounding box is green and text is green
    elif label == "green_buoy":
        cv2.rectangle(frame, (x1, y1), (x1+w, y1+h), (0, 255, 0), 2)
        cv2.putText(frame, text, (x1, y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 2)

    # if object detected is a white buoy, bounding box is white and text is white
    elif label == "white_buoy":
        cv2.rectangle(frame, (x1, y1), (x1+w, y1+h), (255, 255, 255), 2)
        cv2.putText(frame, text, (x1, y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7, (255, 255, 255), 2)

# displays the frame(s) or video in a GUI windows
cv2.imshow("ROBOTX_VIDEO_FRAME", frame)
# terminates program when 'q' is pressed
```

Finally, OpenCV functions are used to print the detected boxes and class labels showing the resulting predictions on an object identified along with their confidence value. Customisation was used to place a red text and red bounding box whenever a red buoy was detected, a green text with a green bounding box whenever a green buoy was detected and so forth for the rest of the classes.

Since, cap = cv2.VideoCapture('resources/buoy_test2.mkv'), a video containing four different colours was used. The finally result printed on screen is the below video capture.



Potential Improvements/Recommendations

For our project, using a Yolov5 machine learning model was discussed as an alternative to just using OpenCV alone, as OpenCV can be prone to false positives and interferences. However, a disadvantage of Using Yolov5 with OpenCV is that it needs and exceptionally powerful CPU to process high speed videos in real time. Even testing the vision system on an i7 processor CPU with four cores, processing times were extremely slow. Project requirements for the vision system required the python application to run on a single board computer such as a RaspberryPI. As it was already failing to process on an i7 at an acceptable speed, it was deemed that using Yolov5 machine learning with OpenCV would not be possible, and the solution for the vision system using only OpenCV was chosen instead.

The entire project involving many teams was focused on developing a very modular system design encompassing microcontrollers and Single Board computers in-lieu of a high-powered computer for the 2022 edition of Robotx. Since Yolov5 was tested at this time, it would be deemed possible to deploy it in future Robotx challenges if a high-powered computer was chosen instead. Furthermore, computers running on NVIDIA graphics have CUDA support which means computer vision programs can have a combination of a powerful CPU and GPU to process their codes. Luckily, not only does OpenCV have support for Yolov5 models, CUDA support can also be imported.

Conclusion

Our team was tasked with the problem solving issues of position-holding and navigation, it is to be noted that if this machine learning vision system was to be used, it would still have had to be integrated with logic, code and modules pertaining to position-holding and navigation. To implement this functional requirement, further developments would have been needed to ensure that the system can perform to a high standard in the competition.