April 2021

DV06

FS-SIM Traffic Cone Detection

By:
Lachlan Masson
Brayth Tarlinton

Executive Summary

The aim of DV06 was to detect Traffic Cones from a set of ROS data contained within a 'Rosbag,' file. This was achieved through utilising YOLOv5, an image-based object detection algorithm. After successfully writing a node that could extract and save vision data from the 'Rosbag,' images were saved to a local folder, allowing object detection to take place. The YOLOv5 (You Only Look Once), algorithm was then retrained on a custom data set of yellow and blue traffic cones to produce a new weights file used in the object detection script. Finally, traffic cone detection took place using the 'detect.py,' scripts available from the official yolov5 GitHub page. Results of the detection are very promising but require further refinement, with detection averaging 0.3 seconds, but with lots of false positives present.

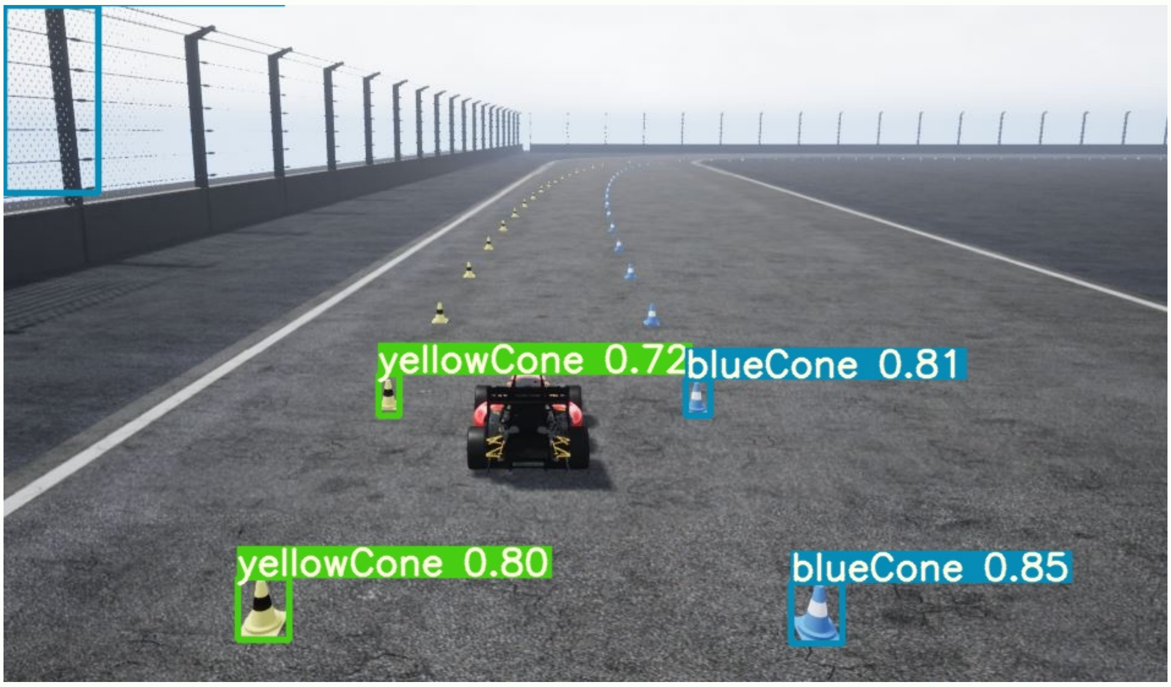
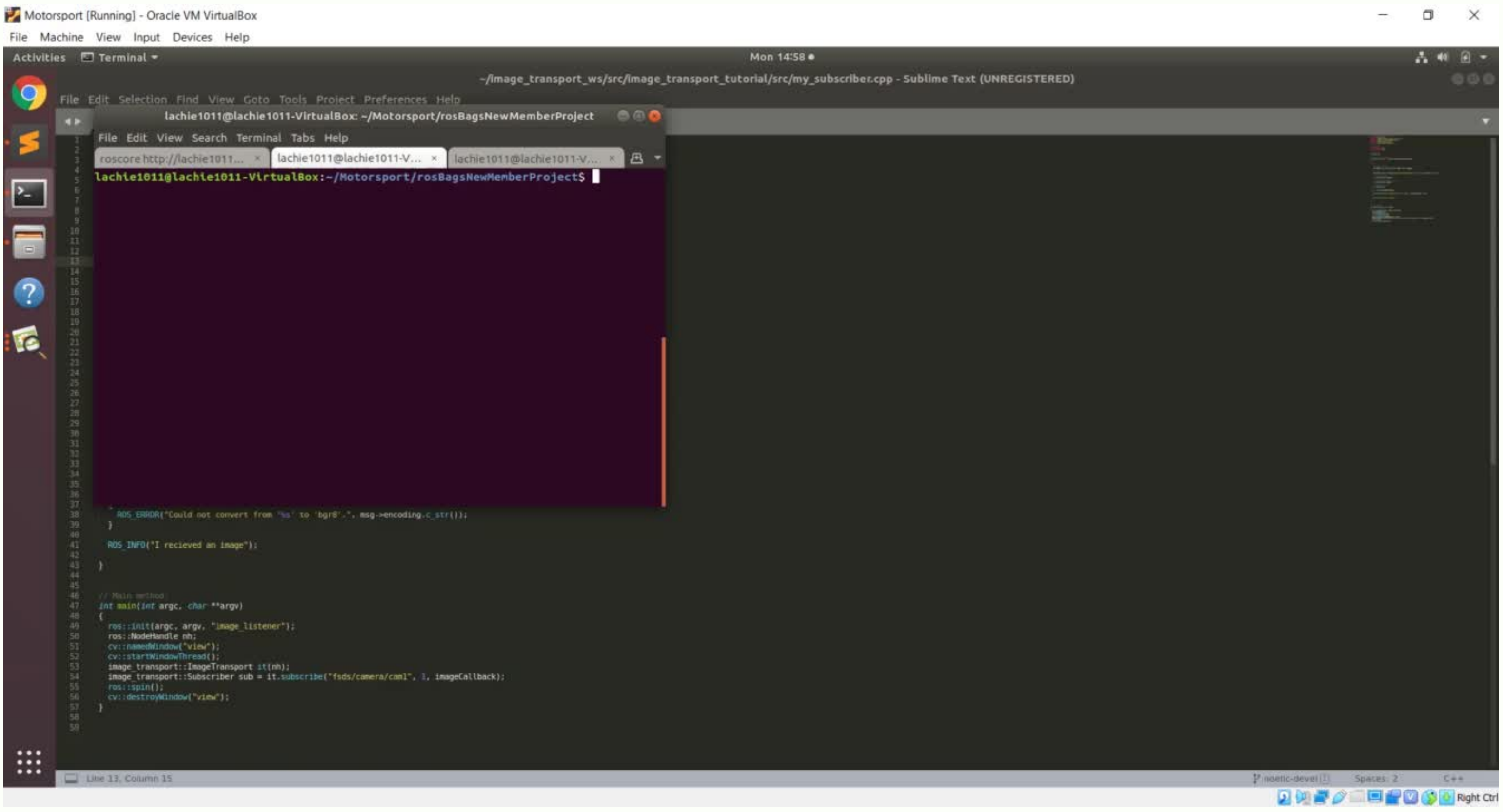


Figure 1: 'Results of detection'

Extracting 'cam1'

In order to detect traffic cones from the cameras, the extraction of image captured by the cameras has to occur. The Robot Operating System or ROS works by pushing data to an environment through 'publisher,' nodes. Currently two cameras, 'cam1,' and 'cam2,' publish data to this ROS environment, in this 'proof of concept,' only data from 'cam1,' is utilised. In order to be able to use this data or any data in this environment, subscriber nodes are written. To extract images a special kind of node an 'image_transport,' node has to be used, in order to main the integrity of the image. Using the ROS package 'image_transport,' the team was able to view the images posted by 'cam1.' These pictures were then stored in a local folder for further use.


Motorsport [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Mon 14:58
~/image_transport_ws/src/image_transport_tutorial/src/my_subscriber.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
lachie1011@lachie1011-VirtualBox: ~/Motorsport/rosBagsNewMemberProject
File Edit View Search Terminal Tabs Help
roscore http://lachie1011...
lachie1011@lachie1011-V...
lachie1011@lachie1011-V...
Line 13, Column 15

```
lachie1011@lachie1011-VirtualBox:~/Motorsport/rosBagsNewMemberProject$
```

```
      ROS_ERROR("Could not convert from '%s' to 'bgr8'.", msg->encoding.c_str());
    }

    ROS_INFO("I recieved an image");

  }


  // Main method
  int main(int argc, char **argv)
  {
    ros::init(argc, argv, "image_listener");
    ros::NodeHandle nh;
    cv::namedWindow("view");
    cv::startWindowThread();
    image_transport::ImageTransport it(nh);
    image_transport::Subscriber sub = it.subscribe("fsds/camera/cam1", 1, imageCallback);
    ros::spin();
    cv::destroyWindow("view");
  }
```

Figure 2: 'Subscriber Node Demo'

YOLOv5 and image detection

To detect objects in images, blue and yellow traffic cones in this instance, two things are required, a suitable dataset and a suitable detector has to be trained. A dataset for this 'proof of concept,' was developed by scraping mages in the 'FS_SIM binary,' using 'ROBOFLOW,' the images were then uploaded and resized to 128x128 pixels with augmentations performed to promote fidelity. These images were then transformed to YOLOv5 format within the software, and an API endpoint was given to access the images remotely.

A suitable object detector has to allow for end-end training, be fast and produce accurate enough results. YOLOv5 is the most recent installment to the 'You Only Look Once,' series, v5 allows for much easier use, being built on 'pyTorch.' The yolo series maintains its extremely fast detection number (140fps) by performing both the identification and classification stages in one look, hence 'you only look once.' It also allows for, and makes training new data sets very easy. Using the provided 'Google Colab,' training page, new weights were able to be trained in 3 minutes.

Preliminary results

Initial results, functionality over optimisation.

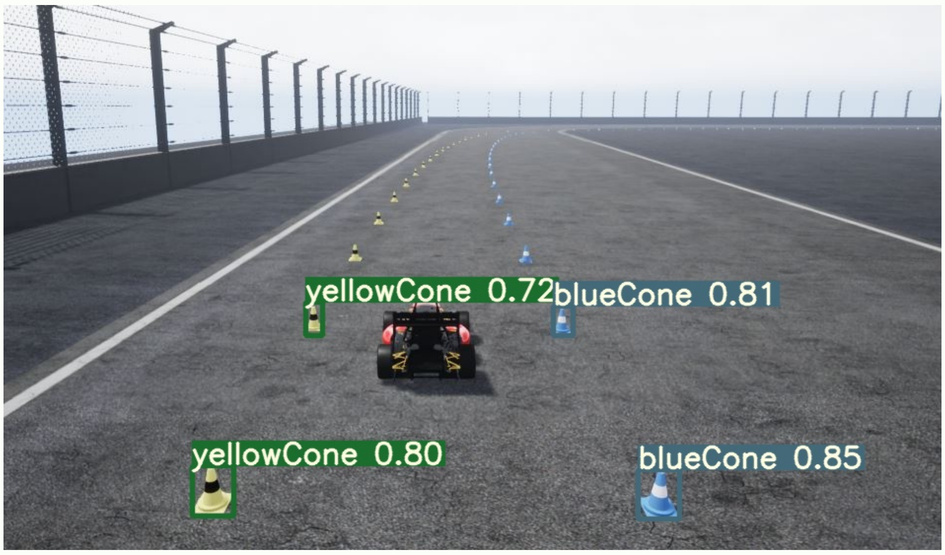
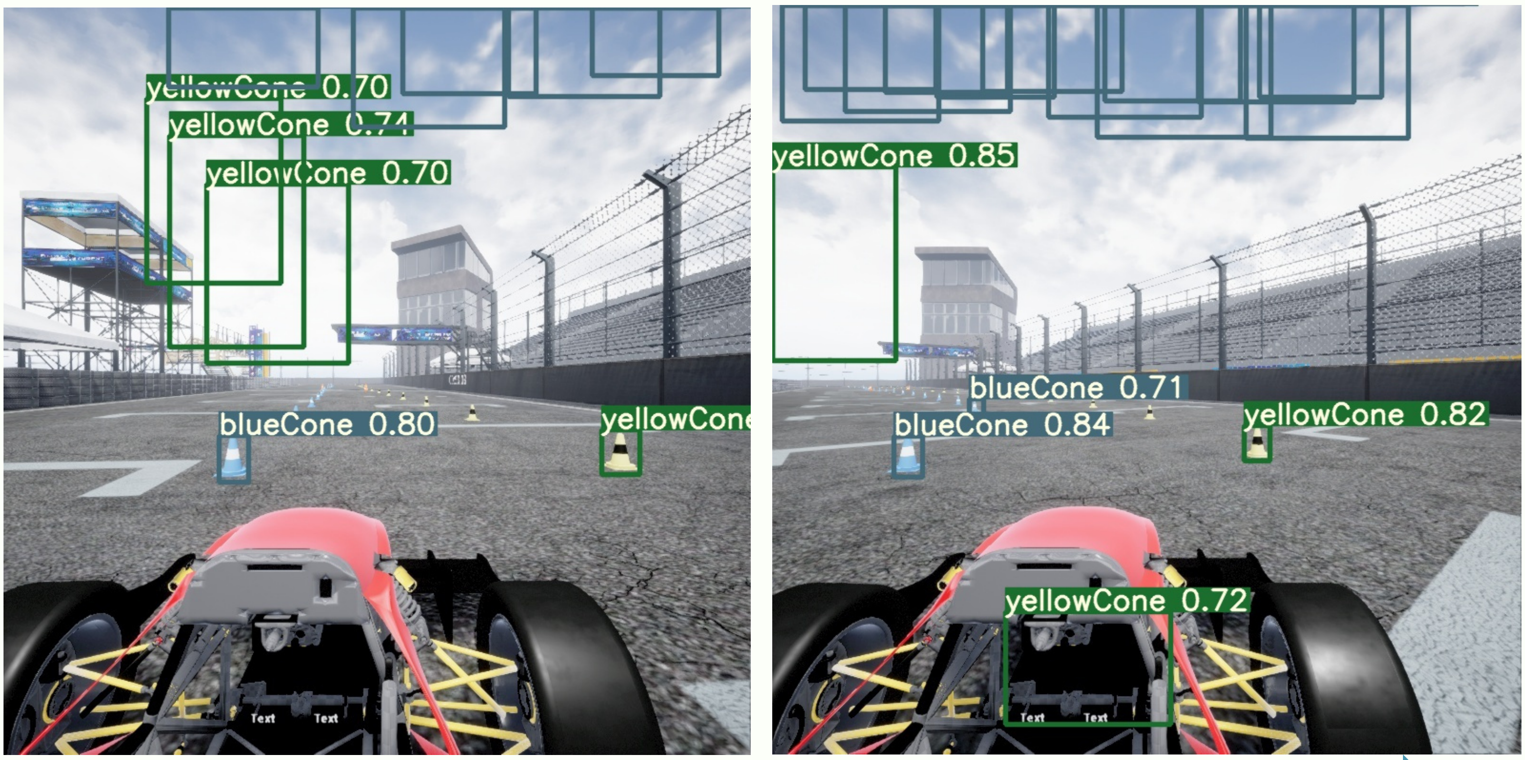


Figure 3: 'Object detection on scraped images.'



Figures 4 and 5: 'Object detection on extracted images."

Implementation now and later

What is working now and what we hope to do next:

Currently, the object detection model is working fairly well on test images taken directly from the subscriber node averaging (0.3) seconds per image. With blue and yellow traffic cones able to be detected. (Full test result images will be uploaded with the project submission). However, there are some big problems with the detection. The model is reliably saying that blocks of white or greys/blacks are a traffic cone. We believe that this is the due to the small size of the images scraped and that each of the cones contains a fair bit of black or grey area around it. This is most present in the top half and bottom third of the image. Brayth and I have come up with many ways to fix this. The three methods to further optimise this model, would be one improve training data, research techniques to get better test data. Two, have more images. Currently only 100 images before augmentation are being used, in more established models 1000 is the norm. Finally, filtering. Removing the aforementioned parts/ masking them before the object detection phase and then reforming the image. This seems very promising after mucking around with some testing.

Moving Forward:
Currently the node only saves images to a folder, which is where the object detection takes place. Moving forward, automating the process within the node is necessary to provide real-time detection. Furthermore, changing the 'detect.py' code to return number of bounding boxes and locations, so that distancing and direction algorithims can take place

End of document