Project Update - Team 03

Panth Patel, Roberto Siqueiros, Borna Mansoormoayad
Github Repository Link

Introduction

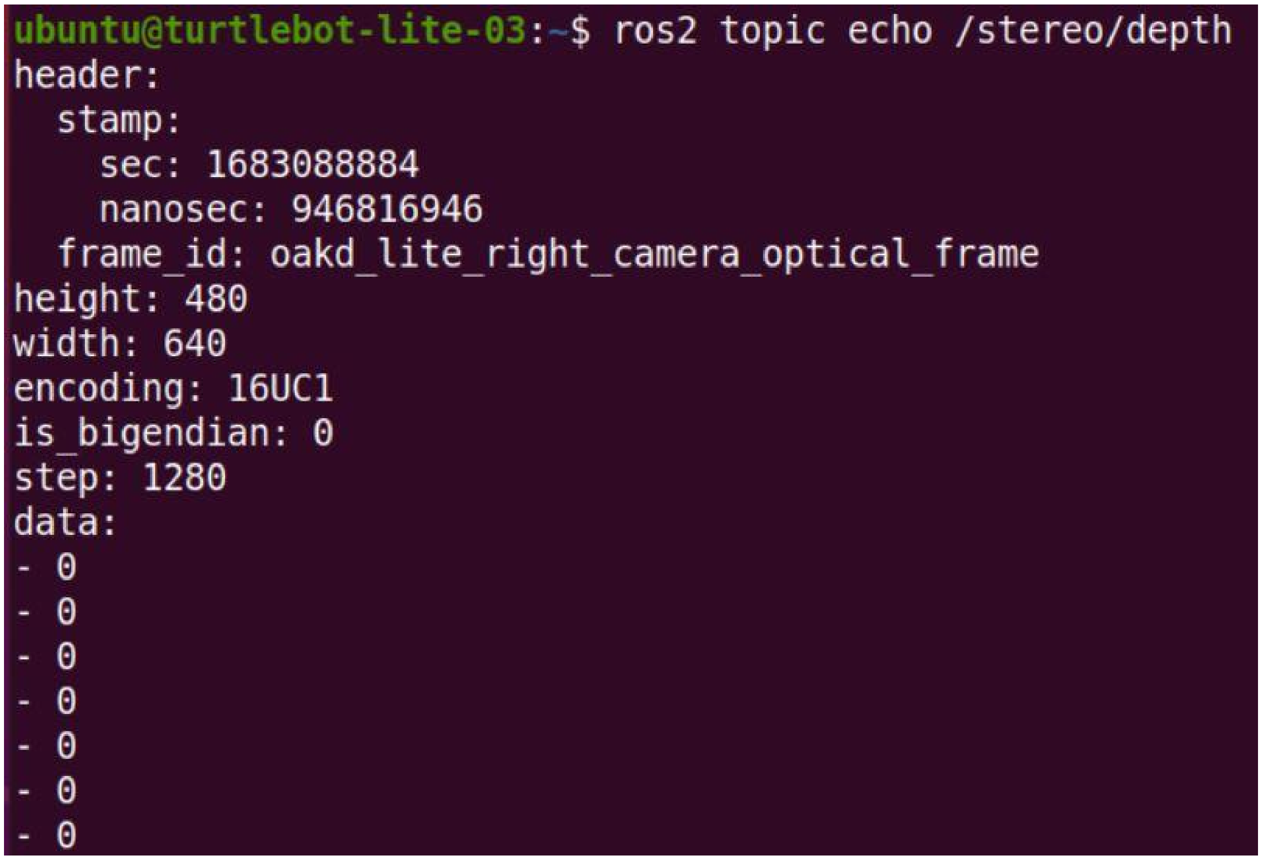
The project involves developing an object detection and tracking system using the OAK-D camera mounted on the Turtlebot platform. A custom dataset was created using the OAK-D camera and annotated using an online platform called Roboflow [2]. Roboflow is used to generate annotation files for training purposes. The data was augmented to improve the accuracy of the model. Afterwards, the object detection model was trained using the YOLOv5 Nano architecture. This lightweight neural network can run on embedded systems like Raspberry Pi. The trained model is capable of detecting multiple objects simultaneously. It provides information about the detection, such as the center point (x,y) of the bounding box, height and width, depth of the center point, class detected, and detection confidence. To incorporate the trained model into our Turtlebot platform, a ROS node was developed and called `custom_object_detection_node`, which extracts the bounding box attributes of the detected objects. This information is then published to `/object_center` topic, to which the `controller` node subscribes. The `controller` node reads the published commands to actuate the Turtlebot through the `/cmd_vel` topic. A PID controller was also integrated into the `controller` node to adjust the velocity of the Turtlebot based on the data received. In the initial objective, the depth was to be calculated using the attributes of the bounding box provided by the `custom_object_detection_node` node. To achieve this, a new ROS node was created and called `depth_extraction`, which subscribed to the `custom_object_detection_node` node. The `depth_extraction` node calculated the depth of the detected object using the stereo depth data obtained from the `/stereo/depth` topic. However, the `/stereo/depth` topic was not outputting any data, causing the `depth_extraction` node to crash. Despite this setback, the `controller` node was modified to infer depth using the bounding box, and publishing the data using a ROS topic to which the `controller` node will subscribe.

Project Goals

This project consists of creating a custom data set using the Turtlebot's OAK-D camera. The custom dataset is then used to train a model using the YOLOv5 [1] nano architecture for object detection. In this case, the model was trained to detect an orange and a book. The trained model is then incorporated into a node, called `custom_object_detection_node`, that is used to extract the bounding box attributes of the detected object. Such attributes are the center point x , center point y, and the area of the bounding box. The extracted data is then published by the custom_object_detection_node node. Next, a node called controller subscribes to the topic being published by the custom_object_detection_node node. The data obtained by the controller node is used to write commands to the `/cmd_vel` topic and actuate the Turtlebot wheels. The controller node has a PID controller that is used on velocity commands of the Turtlebot to calculate the velocity based on the data received.

Initial objective

The overall goal of the project remained the same, to detect an object and extract information from the classified object to control the Turtlebot's velocity commands. The modification in the project objective was the way data was being processed to control the Turtlebot. Initially, the object was detected and the bounding box attributes were published. With the modification in place, `depth_extraction` node was created to subscribe to `custom_object_detection_node` node. The function of the `depth_extraction` node is to calculate the depth of the detected object using the bounding box attributes provided by the `custom_dataset_node`. A `controller` node would subscribe to the topic published by `depth_extraction` node to obtain the calculated depth. Simultaneously, the `controller` node was designed to write and publish commands to the `/cmd_vel` topic based on the received data. The `depth_extraction` node crashed every time the trained object was detected. It was determined that the `extraction_depth` node malfunction was caused by the `/stereo/depth` topic not outputting usable data. Figure 1 shows the data when echoing the `/stereo/depth` topic.



```
ubuntu@turtlebot-lite-03:~$ ros2 topic echo /stereo/depth
header:
  stamp:
    sec: 1683088884
    nanosec: 946816946
  frame_id: oakd_lite_right_camera_optical_frame
height: 480
width: 640
encoding: 16UC1
is_bigendian: 0
step: 1280
data:
- 0
- 0
- 0
- 0
- 0
- 0
- 0
```

Figure 1. No Data Being Echoed via /stereo/depth Topic

Project WorkFlow

The process of the project did change because of the `/stereo_depth` topic publishing incorrect data. The figure below represents the updated project process.

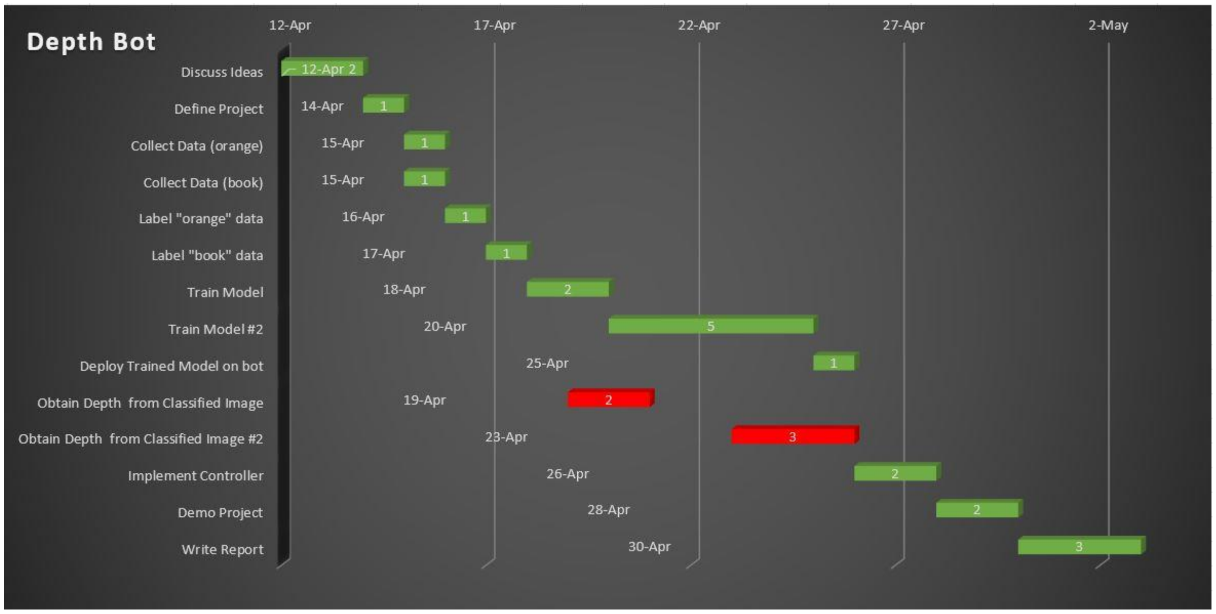


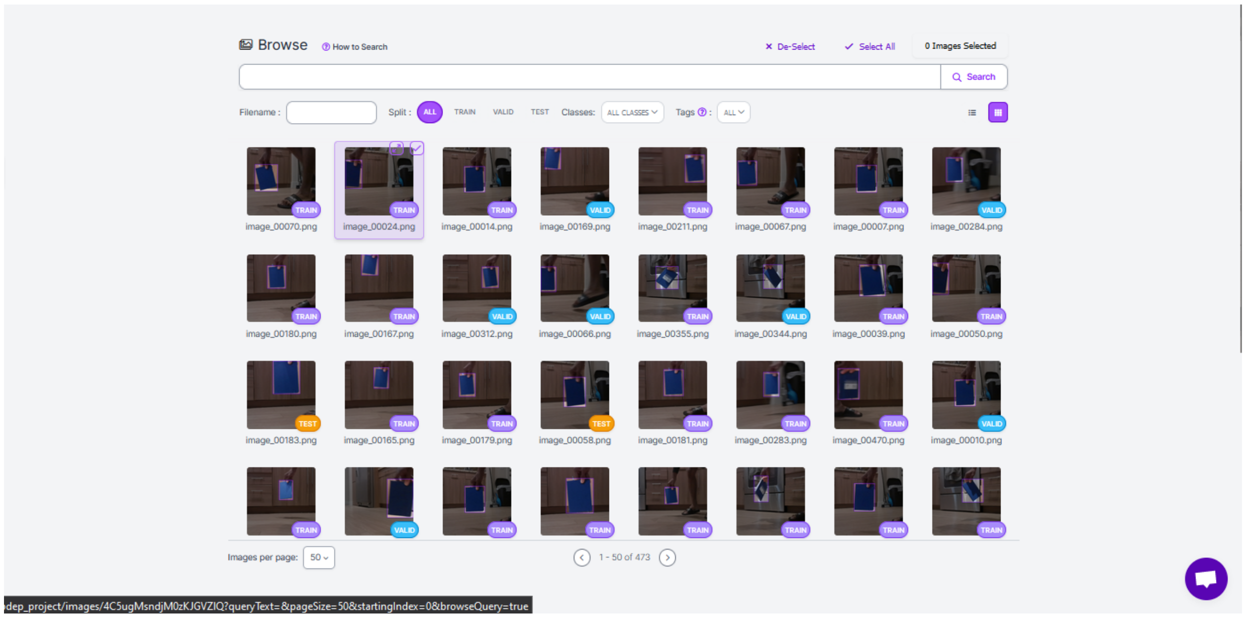
Figure 2. Gantt chart

Description of Project Process

Initially, incorporating the LIDAR sensor into the project had become a possible idea to implement, but the idea was discarded due to time constraints. Scoping the project correctly was crucial to allow a successful completion of the proposed goal. Finally, it was determined to train a deep neural network for object detection and extract data from the classified object. The data would then be used to control the turtlebot.

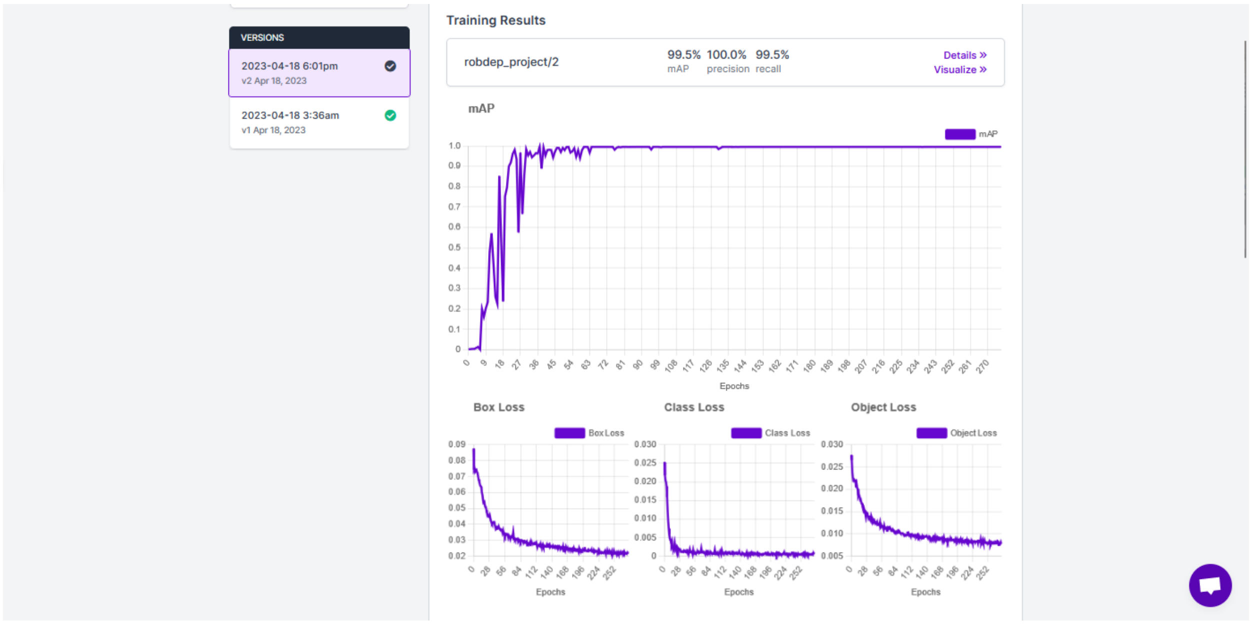
Data Collection

The first step for training the neural network is to determine the objects to be detected. For collecting data, the OAK-D camera mounted on the robot is used. The data set is created using ROS bag recordings of around 2 minutes for each object, the object is placed in front of the robot at different angles. Around 2000 images for each class is collected in the initial ROS bag, which is pruned down to 200 images per class, selecting only clear and relevant images for annotations. The 400 images are uploaded on Roboflow, an online application used for annotations. After annotating the images, augmentation techniques were applied to increase the size of the dataset for better training and to prevent overfitting. To achieve this, the images were vertically and horizontally flipped, rotated by 45 degrees, adjusted saturation, brightness, exposure levels of the images, and induced noise. In the end, 1200 images are annotated per class for training a model. The dataset is split into three parts (Figure 3), train (70%), test (20%), and validation (10%), which is standard practice for training neural networks.



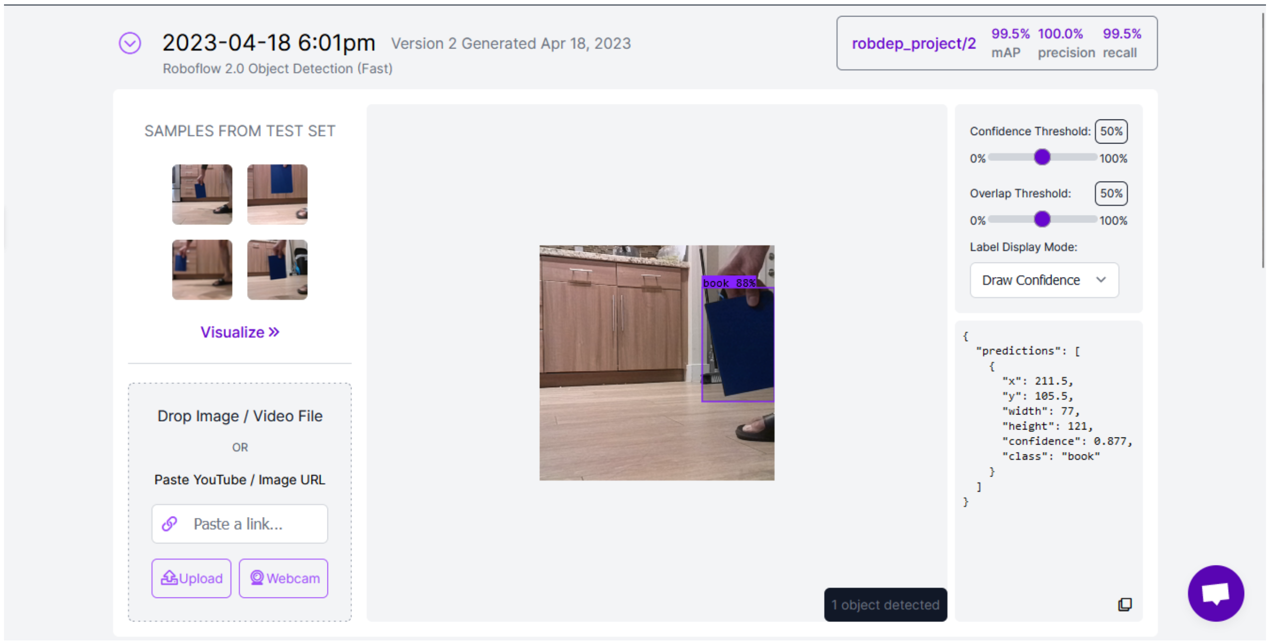
dep_project/images/4C5ugMsndjM0zKJGVZIQ?queryText=&pageSize=50&startingIndex=0&browseQuery=true

Figure 3. The dataset Split into Train, Test, and Validate [2]



robdep_project/2	99.5% mAP	100.0% precision	99.5% recall

Figure 4. Training Charts for Roboflow Model [2]


2023-04-18 6:01pm Version 2 Generated Apr 18, 2023

Roboflow 2.0 Object Detection (Fast)

robdep_project/2 99.5% mAP 100.0% precision 99.5% recall

SAMPLES FROM TEST SET

Visualize »

Drop Image / Video File

OR

Paste YouTube / Image URL

Paste a link...

Upload Webcam

book 88%

1 object detected

Confidence Threshold: 50% 0% 100%

Overlap Threshold: 50% 0% 100%

Label Display Mode: Draw Confidence

```
{
  "predictions": [
    {
      "x": 211.5,
      "y": 105.5,
      "width": 77,
      "height": 121,
      "confidence": 0.877,
      "class": "book"
    }
  ]
}
```

Figure 5. Sample Inference with Predictions [2]

Training using TensorFlow

Initially, Roboflow was used to annotate the dataset. Roboflow also enables users to train and deploy a model online using its API [2]. This process took around two days for annotating the data, training and deploying the model on the robot using API. However, the training of the model using the custom dataset was used as a trial of how good the dataset is in training it on a deployable model on the robot without using the API.

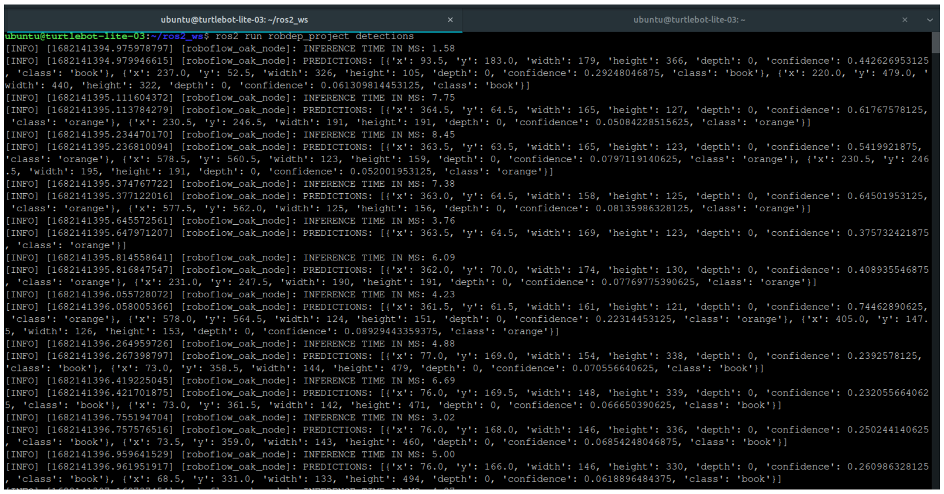
As discussed, the results of the trained model prove the dataset to be adequate. The dataset allows enough training and testing images to fine-tune and train a model. The deployment of the trained model on the robot was considered rather than deployment on the Ubuntu virtual machines and using it to infer and make the robot act accordingly. So a tensorflow lite model Efficient-det-lite-0 is selected. Initially, the Efficient-det-lite-0 results turned out poorly, and the model was taking up almost all the processing power of the robot [3]. Hence, a Tensorflow model zoo (SSD-Mobilenet-V2) was converted to a tensorflow-lite model to be deployed on the robot [4]. The results could have been better, yet, the processing power was much more, and took much less time to infer a single result. Plus, support for TensorFlow has been deprecated, so a faster, newer, and computationally less intensive solution is better.

Training using PyTorch

Next, the YOLOv5 model is trained on the dataset. The trained model is converted to an open neural network exchange (ONNX) file to be deployed directly to the camera. The Oak-D camera has shave tensor processing units that can be used for image processing and computations. Hence, the robot would have a lighter computational burden, as these processes take place in the real time domain. However, the model was not deployed on the Oak-D camera as a pipeline for extracting images after killing the ROS node was required. The incorporation of a real time pipeline for this process inside a ROS node requires network level analysis.

Finally, the model was trained using YOLOv5s [1], a smaller version of YOLOv5. YOLOv5s uses fewer parameters, hence, requires less computational power. The model was trained on Google Collab with the dataset at 150 epochs. The results of the trained model were satisfactory, exhibiting 1-5 FPS. However, 1-5 FPS is too slow for an application that needs to run in real-time, therefore, YOLOv5n was used next. YOLOv5s is the nano version of YOLOv5 with around 1.7 million parameters. The parameters are six times less than YOLOv5s and ten times less than YOLOv5. After deploying the model, the inferences were around 15 FPS and considered too slow for the real-time application. There are several ways of optimizing models, two techniques commonly used are pruning and quantization.

Pruning trims the model while it is being trained, a method which requires a deeper understanding of the field. Quantization was incorporated as well, drastically improving the model's performance by converting the entire model to work on integer-type values instead of floating-point values. Working with floating point variables requires more space and time complexity than integer-type variables, making the model faster but slightly less efficient. After optimizations, the model was deployed on the robot at 25-30 FPS with a confidence interval of 0.4-0.9. Overall, there was a significant tradeoff between training the model to have a high confidence level or allowing the model to have a low inference time.


ubuntu@turtlebot-lite-03: ~/ros2_ws
ubuntu@turtlebot-lite-03: ~

```
ubuntu@turtlebot-lite-03:~/ros2_ws$ ros2 run robdep_project detections
[INFO] [1682141394.975978797] [roboflow_oak_node]: INFERENCE TIME IN MS: 1.58
[INFO] [1682141394.979946615] [roboflow_oak_node]: PREDICTIONS: [{'x': 93.5, 'y': 183.0, 'width': 179, 'height': 366, 'depth': 0, 'confidence': 0.442626953125
, 'class': 'book'}, {'x': 237.0, 'y': 52.5, 'width': 326, 'height': 105, 'depth': 0, 'confidence': 0.29248046875, 'class': 'book'}, {'x': 220.0, 'y': 479.0, '
width': 440, 'height': 322, 'depth': 0, 'confidence': 0.061309814453125, 'class': 'book'}]
[INFO] [1682141395.111604372] [roboflow_oak_node]: INFERENCE TIME IN MS: 7.75
[INFO] [1682141395.113784279] [roboflow_oak_node]: PREDICTIONS: [{'x': 364.5, 'y': 64.5, 'width': 165, 'height': 127, 'depth': 0, 'confidence': 0.61767578125,
 'class': 'orange'}, {'x': 230.5, 'y': 246.5, 'width': 191, 'height': 191, 'depth': 0, 'confidence': 0.05084228515625, 'class': 'orange'}]
[INFO] [1682141395.234470170] [roboflow_oak_node]: INFERENCE TIME IN MS: 8.45
[INFO] [1682141395.236810094] [roboflow_oak_node]: PREDICTIONS: [{'x': 363.5, 'y': 63.5, 'width': 165, 'height': 123, 'depth': 0, 'confidence': 0.5419921875,
'class': 'orange'}, {'x': 578.5, 'y': 560.5, 'width': 123, 'height': 159, 'depth': 0, 'confidence': 0.0797119140625, 'class': 'orange'}, {'x': 230.5, 'y': 246
.5, 'width': 195, 'height': 191, 'depth': 0, 'confidence': 0.052001953125, 'class': 'orange'}]
[INFO] [1682141395.374767722] [roboflow_oak_node]: INFERENCE TIME IN MS: 7.38
[INFO] [1682141395.377122016] [roboflow_oak_node]: PREDICTIONS: [{'x': 363.0, 'y': 64.5, 'width': 158, 'height': 125, 'depth': 0, 'confidence': 0.64501953125,
 'class': 'orange'}, {'x': 577.5, 'y': 562.0, 'width': 125, 'height': 156, 'depth': 0, 'confidence': 0.08135986328125, 'class': 'orange'}]
[INFO] [1682141395.645572561] [roboflow_oak_node]: INFERENCE TIME IN MS: 3.76
[INFO] [1682141395.647971207] [roboflow_oak_node]: PREDICTIONS: [{'x': 363.5, 'y': 64.5, 'width': 169, 'height': 123, 'depth': 0, 'confidence': 0.375732421875
, 'class': 'orange'}]
[INFO] [1682141395.814558641] [roboflow_oak_node]: INFERENCE TIME IN MS: 6.09
[INFO] [1682141395.816847547] [roboflow_oak_node]: PREDICTIONS: [{'x': 362.0, 'y': 70.0, 'width': 174, 'height': 130, 'depth': 0, 'confidence': 0.408935546875
, 'class': 'orange'}, {'x': 231.0, 'y': 247.5, 'width': 190, 'height': 191, 'depth': 0, 'confidence': 0.07769775390625, 'class': 'orange'}]
[INFO] [1682141396.055728072] [roboflow_oak_node]: INFERENCE TIME IN MS: 4.23
[INFO] [1682141396.058005366] [roboflow_oak_node]: PREDICTIONS: [{'x': 361.5, 'y': 61.5, 'width': 161, 'height': 121, 'depth': 0, 'confidence': 0.74462890625,
 'class': 'orange'}, {'x': 578.0, 'y': 564.5, 'width': 124, 'height': 151, 'depth': 0, 'confidence': 0.22314453125, 'class': 'orange'}, {'x': 405.0, 'y': 147.
5, 'width': 126, 'height': 153, 'depth': 0, 'confidence': 0.08929443359375, 'class': 'orange'}]
[INFO] [1682141396.264959726] [roboflow_oak_node]: INFERENCE TIME IN MS: 4.88
[INFO] [1682141396.267398797] [roboflow_oak_node]: PREDICTIONS: [{'x': 77.0, 'y': 169.0, 'width': 154, 'height': 338, 'depth': 0, 'confidence': 0.2392578125,
'class': 'book'}, {'x': 73.0, 'y': 358.5, 'width': 144, 'height': 479, 'depth': 0, 'confidence': 0.070556640625, 'class': 'book'}]
[INFO] [1682141396.419225045] [roboflow_oak_node]: INFERENCE TIME IN MS: 6.69
[INFO] [1682141396.421701875] [roboflow_oak_node]: PREDICTIONS: [{'x': 76.0, 'y': 169.5, 'width': 148, 'height': 339, 'depth': 0, 'confidence': 0.232055664062
5, 'class': 'book'}, {'x': 73.0, 'y': 361.5, 'width': 142, 'height': 471, 'depth': 0, 'confidence': 0.066650390625, 'class': 'book'}]
[INFO] [1682141396.755194704] [roboflow_oak_node]: INFERENCE TIME IN MS: 3.02
[INFO] [1682141396.757576516] [roboflow_oak_node]: PREDICTIONS: [{'x': 76.0, 'y': 168.0, 'width': 146, 'height': 336, 'depth': 0, 'confidence': 0.250244140625
, 'class': 'book'}, {'x': 73.5, 'y': 359.0, 'width': 143, 'height': 460, 'depth': 0, 'confidence': 0.06854248046875, 'class': 'book'}]
[INFO] [1682141396.959641529] [roboflow_oak_node]: INFERENCE TIME IN MS: 5.00
[INFO] [1682141396.961951917] [roboflow_oak_node]: PREDICTIONS: [{'x': 76.0, 'y': 166.0, 'width': 146, 'height': 330, 'depth': 0, 'confidence': 0.260986328125
, 'class': 'book'}, {'x': 68.5, 'y': 331.0, 'width': 133, 'height': 494, 'depth': 0, 'confidence': 0.0618896484375, 'class': 'book'}]
```

Figure 6. Terminal Screenshot of the ROS Node Running

Validation

For an object detection model, the average confidence interval normally ranges between 0.6-0.8, which is an indication that the model is able to detect objects. As discussed above, different models were tested which returned inferences at different confidence levels. To validate if the model was satisfactory or not, an intuitive method for checking confidence levels, center point x, center point y and area of the bounding box of the object being detected at a different distance from the camera. If these inference attributes were consistent at different points where the object was being placed, it is considered the model was valid for deployment because even as a human, one is able to distinguish objects from surroundings at a certain confidence level depending on the position of the object, and one can consistently determine where the object lies.

After deploying the models on the robot, the model trained on Roboflow was the most accurate, with satisfactory inferences, and average confidence intervals between 0.6 and 0.9. The Efficient-Det-Lite-0 returned inferences with confidence interval 0.05-0.2 which is pretty low, YOLOv5 and YOLOv5s returned inferences with confidence interval of 0.7-0.9, with slow processing time and so accuracy was traded for speed by using YOLOv5n which had lesser accuracy but gave consistent results [3].

Controller Node
The `controller` node subscribes to the data published in `/object_center`. This topic contains the real time data (position and size of the boundary box) of the detected object. A PID class is defined in this node to control the velocity commands (published through `/cmd_vel`). In order for the robot to navigate in 2D space, two controller objects are defined for the angular z velocity command (yaw) and linear x velocity command (forward and backward). The input to these controllers are the data received in `/object_center`. To center the object, a setpoint of 125 is defined. This number signifies the horizontal center point of the 250 by 250 pixel image. The deduction of the x center point in the `/object_center` topic from the calibration setpoint, provides the error that is used in the PID calculations. The PID gains are defined as calibrations for this controller for future tuning. This PID controller controls the yaw command (angular velocity z) published in `/cmd_vel`. The forward and reverse velocity command is calculated by deduction of the received bounding area (in `/object_center`) from the setpoint. This error is used in the PID controller for linear velocity x command. The closer the camera is to the image, the larger the boundary area. Hence, this controller controls the robot velocity for tracking the object. An additional saturation function is defined in the PID class that uses upper and lower limit velocity thresholds to limit the speed of the robot. The saturation function takes the PID output as the input argument, and based on the speed limit calibrations, if the value of PID output surpasses the thresholds, instead it will output the threshold value. This was done so the robot moves very smoothly without perfect PID tuning and unnecessary controller responses.

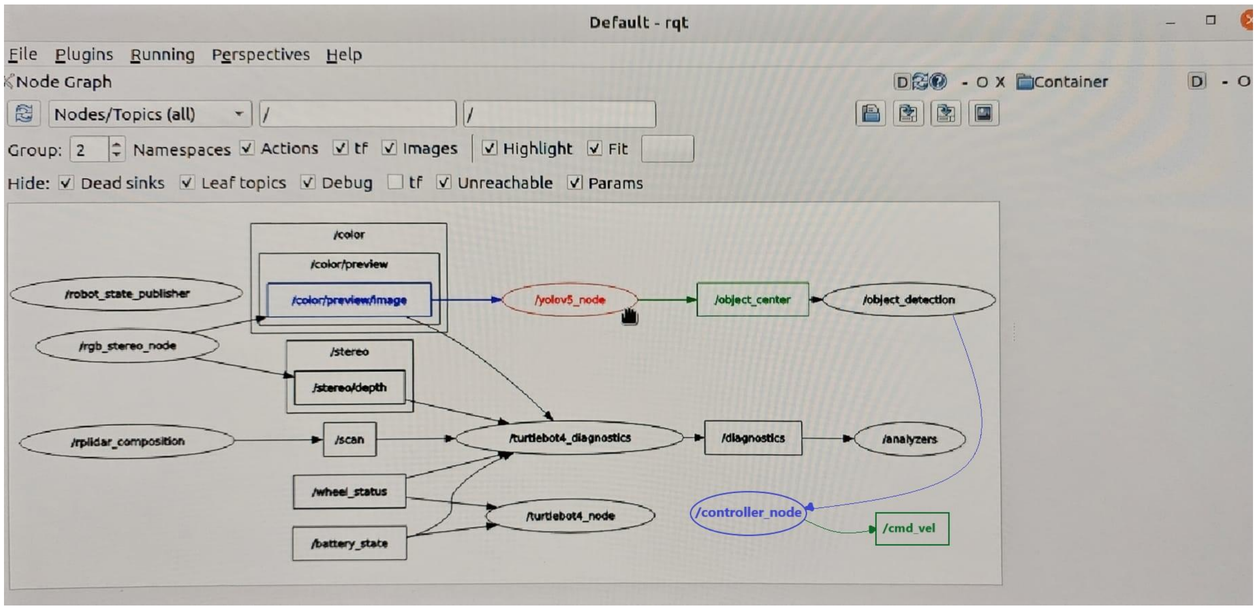
ROS Architecture

The project in total has 2 additional nodes and 1 additional topic used to transfer information between the 2 nodes. As discussed we have `custom_object_detection_node` and `controller` and /object_center topic.

The `custom_object_detection` node subscribes to `/color/preview/image` topic and runs an instance of YOLOv5n object by loading the weights that are obtained after training. The weights are already pruned and quantization of the model takes place after loading the weights.

The `custom_object_detection_node` publishes the topic `/object_center` of type `Point` from `geometry_msg` which has 3 data fields: `x`, `y` and `z`. `Point.x` is the center point x, `Point.y` is the center point y and `Point.z` is the area of the bounding box.

The controller node subscribes to the `/object_center` topic to actuate the bot using the PID controller written inside the node.



Group: 2 Namespaces [✓ Actions] [✓ tf] [✓ Images] [✓ Highlight] [✓ Fit]
Hide: [✓ Dead sinks] [✓ Leaf topics] [✓ Debug] tf [✓ Unreachable] [✓ Params]

Figure 7. Project Architecture

Trade-Offs

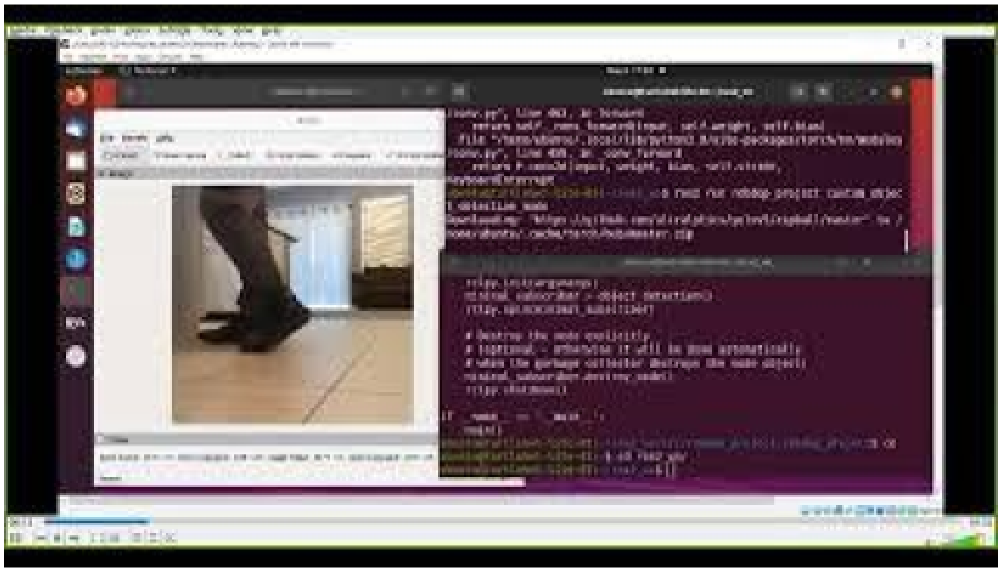
Firstly, it was crucial to balance the need for accuracy with speed when selecting the model. Initially, Roboflow was used to train a model, which yielded accurate results with fair inferences. However, the model was not deployable on the robot, which led the project to try other options. Afterwards, a TensorFlow lite model was used, which needed to be more accurate and consume more processing power. Ultimately, the YOLOv5 method was used to train the dataset and convert it to an ONNX file that could be deployed directly to the Oak-D camera. Given the time constraint, YOLOv5 was the model training method to be used due to the accuracy and the reduced computational time on the robot

Secondly, the balance model compatibility with computational resources needed to be taken into account. Initially, the TensorFlow lite model was used on the robot. However, high computational challenges arised due to the model's high processing power requirements.

Therefore, PyTorch was selected to train a YOLOv5 model that could run directly on the Oak-D camera's tensor processing units, reducing the computational burden on the robot.

Finally, there was a trade-off between the ease of deployment and incorporating the model into the ROS node. Although deploying the ONNX file directly to the Oak-D camera would have been more comfortable, it was not feasible due to the need to set up a pipeline for extracting images after killing the ROS node. Therefore, the model was incorporated into a ROS node, a more complex task. The trade-offs consisted of balancing the completion of the goals or allowing the model to have good accuracy, being compatible, and taking into account the computational resources.

Videos



```
conv.py", line 463, in forward
    return self._conv_forward(input, self.weight, self.bias)
  File "/home/ubuntu/.local/lib/python3.8/site-packages/torch/nn/modules/conv.py", line 459, in _conv_forward
    return F.conv2d(input, weight, bias, self.stride,
KeyboardInterrupt
[illegible]$ ros2 run robotdog_project custom_object_detection_node
Downloading: "https://github.com/ultralytics/yolov5/zipball/master" to /home/ubuntu/.cache/torch/hub/master.zip
```

```
    rclpy.init(args=args)
    minimal_subscriber = object_detection()
    rclpy.spin(minimal_subscriber)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
[illegible]
[illegible]$ cd robot_pkg
[illegible]
```

Citation

[1] Ultralytics, "GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite," , Tag: v7.0, Commit: 915bbf2*, GitHub*. https://github.com/ultralytics/yolov5

[2] Patel, P. (2023). *Robdep Roboflow Workspace*. Roboflow. Retrieved from https://app.roboflow.com/robdep

[3] "Retrain EfficientDet-Lite detector for the Edge TPU (TF2)" https://colab.research.google.com/github/google-coral/tutorials/blob/master/retrain_efficientdet_model_maker_tf2.ipynb

[4] Tensorflow. (2021, May 7). *TensorFlow Model Garden*. GitHub. Retrieved from https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

Appendix

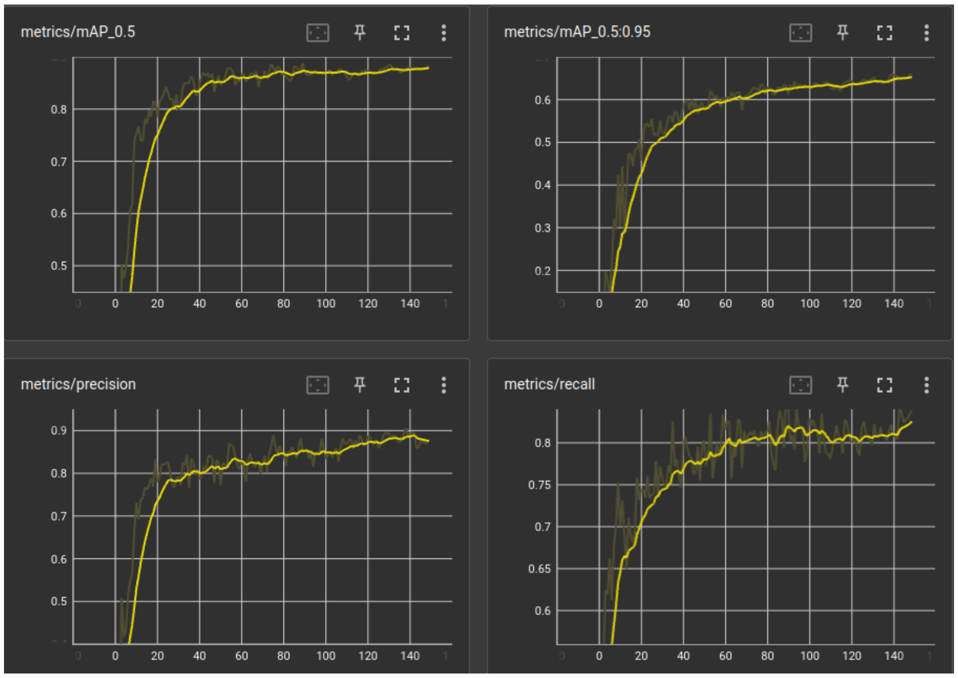


Fig 8. Training Metrics for the YOLOv5n deployed on the robot.

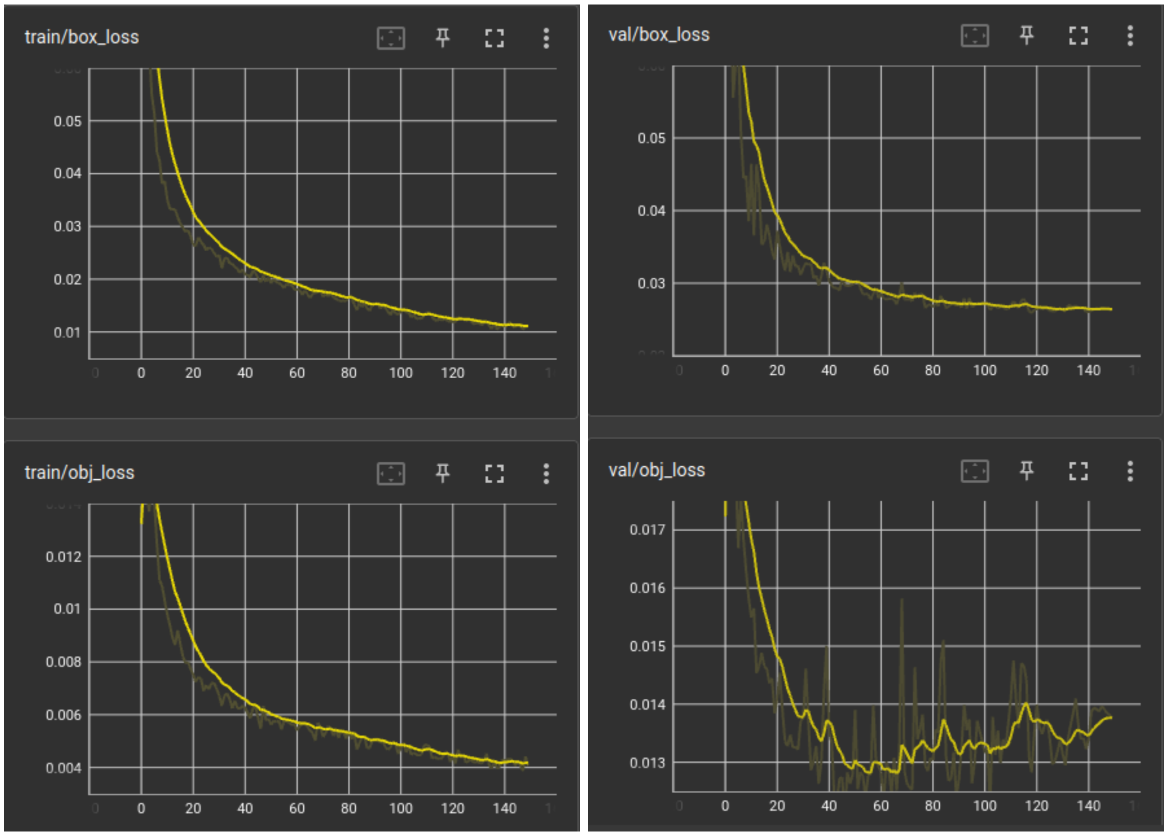


Fig 9. Training and Validation loss for the YOLOv5n deployed on the robot