ME5400A ROBOTICS PROJECT FINAL REPORT

Title: Perception Pipeline for Autonomous Treasure Hunting Robot

Author: Madkaikar Atharva Atul

Matriculation Number: A0268376M

Supervisor: Prof. Teo Chee Leong

Co-supervisor: Christina Lee Dao Wen

Co-supervisor: Sun Shuo

Department of Mechanical Engineering

National University of Singapore

Acknowledgments

I am immensely grateful for the invaluable support and assistance I received from various individuals throughout the course of this project. First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Teo Chee Leong for his invaluable guidance and support. I would also like to thank my co-supervisors Mr. Sun Shuo and Ms. Christina Lee Dao Wen for their encouragement and mentorship advice throughout the project.

I would also like to thank my department and the Advanced Robotics Centre for providing me the opportunity to work on this challenging project and providing all the necessary facilities.

Lastly, I extend my sincere thanks to my teammates, whose collaborative efforts have been instrumental in successful completion of this project.

Table of Contents

List of Figures

1. Introduction

The research and advancements in the field of autonomous vehicles propagate the development of autonomous robots. With the society moving towards a more autonomous future, these developments will speed up the transition. Moreover, development of an autonomous driving platform prototype serves as a testing ground for cutting edge technologies in a safe and a controlled environment. Further research and results obtained will help us make transition to a safer and a more reliable future for autonomous vehicles in everyday use.

1.1. Background

The task was to develop and test a Mini Autonomous Driving Platform with a focus on contributing to the core areas: Hardware Development, Perception, Localization and Navigation. To target all these core areas, a decision was made to design and implement an Autonomous Treasure Hunting robot capable of exploring an unknown environment and finding a specified treasure object hidden within the environment. On a broader scale, such a robot could be deployed in various real-life environments and scenarios like:

- Search and Rescue Operations.
- To search and retrieve for lost or buried items in locations like abandoned buildings, underground tunnels, forests, etc.
- Assist to uncover or find artifacts in archaeological sites without disturbing delicate structures or sites.

Once the challenge was finalized, it was necessary to design and build a suitable hardware from ground up, alongside the deployment of appropriate perception, localization, and navigation algorithms to achieve complete autonomy. The autonomous vehicle was capable of creating an accurate online map of the unknown environment, identifying static and dynamic obstacles, navigating through the environment, and successfully locating, recognizing, and tracking the treasure object within it.

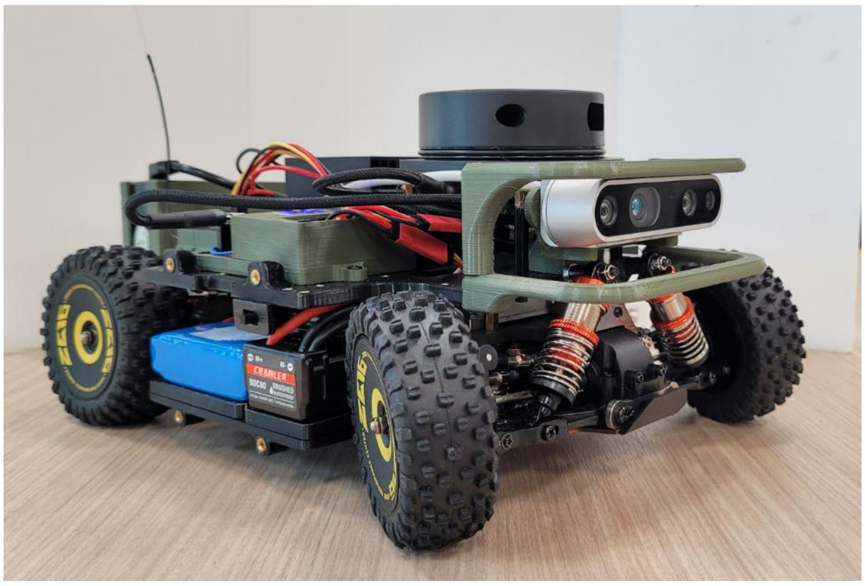


Figure 1: Autonomous Treasure Hunting Robot

This report provides comprehensive insight into the Perception pipeline developed for this project, detailing its development and an in-depth explanation about the functionality of the pipeline. Additionally, the report proposes refinements to the pipeline and introduces a new framework for potential future applications.

1.2. Objective

Perception helps the autonomous vehicles perceive and understand their surroundings and enables them to navigate safely through the environment. It enables the vehicles to adapt quickly to the dynamic changes in the environment allowing effective real-time functioning.

Considering the objective of the project, the Perception pipeline was designed to accomplish the following functionalities:

a. Dynamic object detection – Detecting static and dynamic objects to enable real-time obstacle avoidance.
b. Treasure object selection – Enabling the user to select any object as treasure.
c. Re-recognition – Locating and re-recognizing the treasure object hidden in the environment.
d. Tracking – After locating and identifying the treasure object within the environment, enable tracking to maintain consistency and enhance robustness.

Although, the Perception pipeline was developed with the project's objectives in mind, it was also designed to be used as a general purpose stack with other autonomous driving platforms and future projects.

2. Literature Review

Before developing a unified perception pipeline for the project, it was necessary to review some existing approaches and techniques that could be implemented to address the core challenges: Dynamic Obstacle Detection, Object Recognition and Object Tracking.

In addressing dynamic obstacle detection, two state of the art methods were considered and reviewed: YOLO (You Only Look Once) [1] and Fast R-CNN [2]. It was observed that Fast RCNN had superior inference accuracy in comparison to YOLO. However, it was also observed that YOLO offered significantly faster inference compared to those for Fast R-CNN [3]. Given the hardware limitations of the Jetson Orin Nano used for this project and imperative for real-time performance given the nature of the task, YOLO v8 was chosen to be implemented in the perception pipeline for object detection.

To tackle the challenges associated with object recognition and subsequent tracking, several different approaches were experimented with. One approach involved generating a probability distribution of the image frame based on the histogram extracted from the treasure object, followed by histogram backprojection and then utilizing CAMShift (Continuously Adaptive Meanshift) algorithm to locate and track the treasure object [4]. Another approach to tackle object recognition problem was to extract features from the treasure object and perform feature matching to locate the treasure object within the image frame [5]. While these approaches demonstrated effectiveness under certain environmental conditions, they also exhibited several drawbacks which will be further elaborated upon in the following section. A novel approach, combining object detection with histogram matching, was proposed and integrated into the unified pipeline. Details of this approach are presented in detail in the subsequent sections.

Initially, the object recognition approach combined with Extended Kalman Filter (EKF) was considered for object tracking to consolidate object recognition and tracking within one single block of code. However, this approach had plenty of drawbacks, as false recognitions would affect the quality of predictions, eventually compromising the tracking performance. As a result, the DeepSort object tracking framework was included into the unified pipeline to enable robust object tracking. This framework was considered for its real-time performance capabilities and feature embeddings to significantly improve the tracking performance [6].

3. Methodology

This section describes the core functionalities of the perception stack, including Dynamic Object Detection, Object Selection, Object Re-recognition and Object Tracking in detail. The section also briefly discusses the camera model and the process of de-projecting the point in the image coordinates to cartesian co-ordinates utilizing the depth information provided by the stereo camera.

3.1. Hardware

The Perception stack utilized the onboard Intel RealSense D435 stereo camera and Nvidia Jetson Orin Nano. The RealSense D435 camera was calibrated using Intel's proprietary calibration tool. The depth camera provided colour and depth frames which were utilized to generate inferences and get their location in the camera frame. The Jeston was flashed with Jetpack and the necessary libraries including CUDA, CUDNN, Torch, Torchvision, and OpenCV were built from source to enable the Jetson to run object detection and object tracking on the Nvidia GPU.

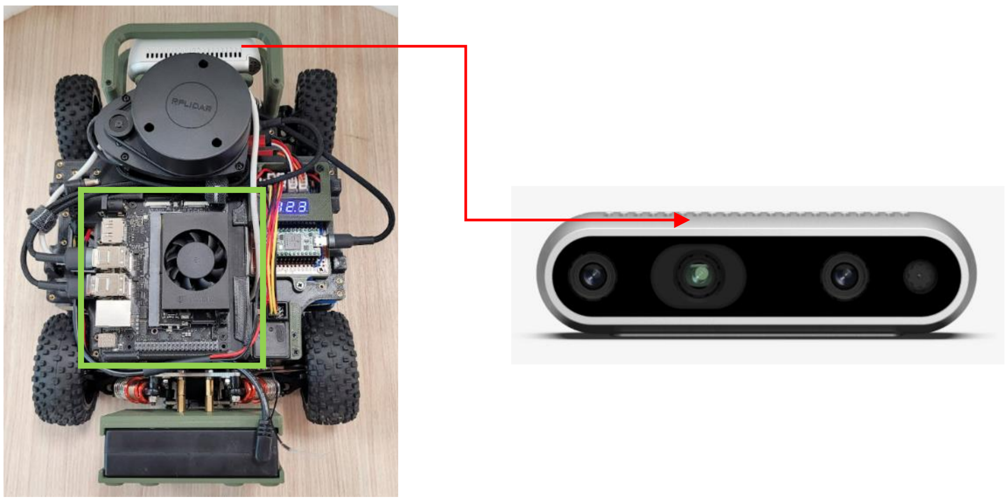


Figure 2: Onboard Hardware. Nvidia Jetson (left). RealSense D435 (right)

3.2. Object Detection

One of the primary functions of the Perception stack in an autonomous vehicle is to detect both static and dynamic obstacles enabling the vehicle to manoeuvre safely by acting promptly on the information provided by the vision sensors. To enable real-time object detection, a pretrained model of YOLO (You Only Look Once) from Ultralytics trained on the COCO 128 dataset was used. The YOLO v8 model was chosen for based on its:

- Efficiency
- Faster Inference Times
- GPU Support
- Real Time Capabilities
- Hyper Parameter Flexibility

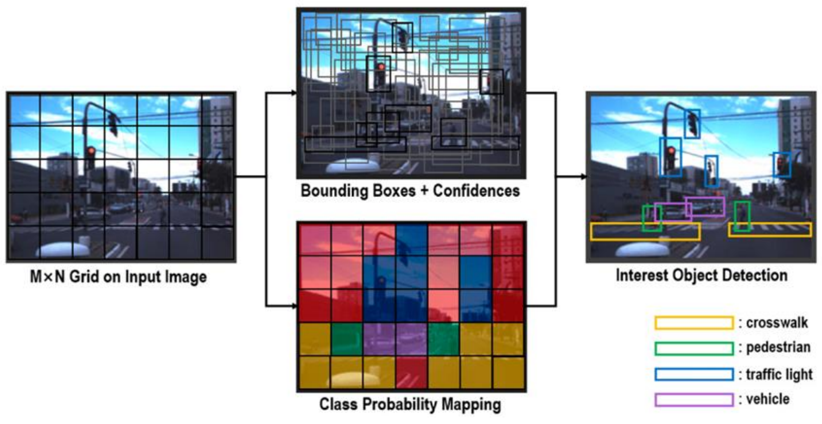
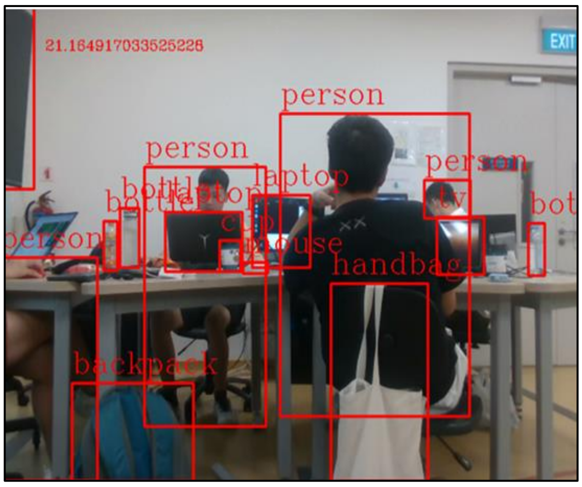
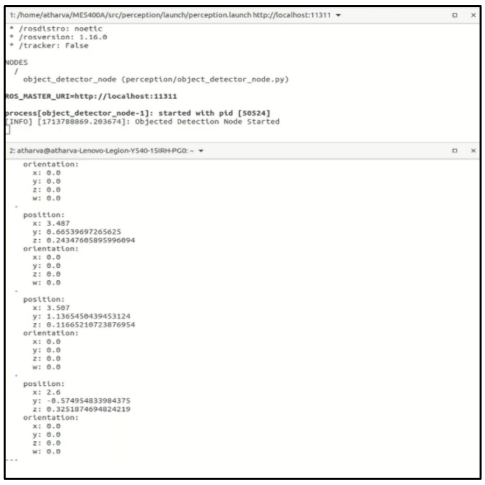

M×N Grid on Input Image
Bounding Boxes + Confidences
Class Probability Mapping
Interest Object Detection
: crosswalk
: pedestrian
: traffic light
: vehicle

Figure 3: YOLO Pipeline

The bounding boxes were extracted from the model inferences and the centre points of each bounding box were computed. The centre points were back projected from the image frame to their corresponding cartesian co-ordinates in the camera frame. A ROS publisher was established to publish these points to a ROS topic to be further utilized by the navigation stack to facilitate obstacle avoidance. The central points of the detections were published as standard ROS messages, such as ‘PoseStamped’ or ‘PoseArray’. A custom ROS message type was provided along with the perception stack which could enable publishing additional information like the size of the object alongside the class type and confidence score to further enhance the obstacle avoidance capabilities of the autonomous vehicle.




1: /home/atharva/ME5400A/src/perception/launch/perception.launch http://localhost:11311
2: atharva@atharva-Lenovo-Legion-Y540-15IRH-PG0: ~

```
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /tracker: False

NODES
  /
    object_detector_node (perception/object_detector_node.py)

ROS_MASTER_URI=http://localhost:11311

process[object_detector_node-1]: started with pid [50524]
[INFO] [1713788869.203674]: Objected Detection Node Started
```

```
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
  -
    position:
      x: 3.487
      y: 0.66539697265625
      z: 0.24347605895996094
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
  -
    position:
      x: 3.507
      y: 1.1365450439453124
      z: 0.11665210723876954
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
  -
    position:
      x: 2.6
      y: -0.574954833984375
      z: 0.3251874694824219
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
---
```

Figure 4: Object detection inferences. Bounding boxes (left). Cartesian coordinates of the centres published to ROS topic (right).

3.3. Visual Deprojection

The inferences generated by the YOLO model were two dimensional and confined to the image frame. The stereo camera provided the depth map along with the colour frame which enabled back projection of the pixel points from the image frame to three-dimensional Cartesian points in the camera frame. The camera was initialized with a suitable resolution for both the depth frames and the colour frames. The stereo frames were then rectified and aligned with the depth frame to get one to one correspondence between the pixels and the depth information. Once the colour and depth frames were aligned, the camera intrinsics were then obtained by utilizing the RealSense Python API library. The pinhole camera model was used to convert pixel co-ordinates in the image frame to 3-dimensional co-ordinates in the camera frame. This enabled sending accurate positional and size information regarding the detected objects.

Formulae:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Z \cdot K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Pinhole Camera Model:

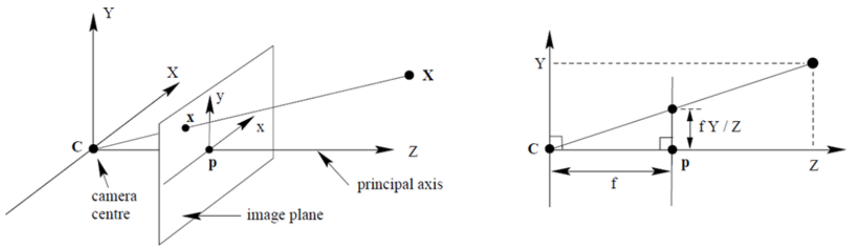


Figure 5: Pinhole Camera Model

3.4. Treasure Object Selection

To enable the selection of the treasure object, an interactive GUI (Graphical User Interface) window was included into the unified perception pipeline. The window was created using Python's 'tkinter' library and provided clickable buttons to select the object of choice as the treasure. The list of available objects was dynamically generated based on the number of detections provided by the object detector. Once the treasure object is selected, the ROI is extracted from the image frame and its histogram computed and stored in the memory. Furthermore, dynamic object selection allows for the possibility of changing the treasure object at any time while the robot is being deployed into the environment.

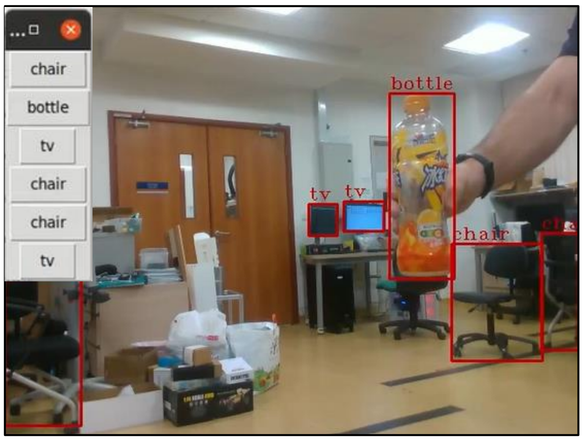


Figure 6: GUI for selecting the treasure object.

3.5. Treasure Object Re-recognition

Object Re-recognition is the ability to not only detect objects of a particular class type but also to re-identify and recall the objects that have been previously detected and encountered. Re-recognition helps verify the designated object and reduces the chances of false positives. Since the project involved finding a specific object, it was crucial for the perception stack to retain the memory of the designated 'treasure' object and accurately re-recognize it upon discovery. Therefore, implementation of a robust and an accurate method was necessary to identify the correct object. Moreover, object re-identification is an active area of research and some open-source implementations like 'Torchreid' and 'CLIP' exists. However, these implementations are used to re-identify inferences of a specific class type. For example, Torchreid is a deep learning based Python library for person re-identification while CLIP is deep learning based character re-identification framework. A generalized framework for object re-recognition has not been developed yet.

To tackle this challenge, it was necessary to come up with a custom framework tailored to our specific needs and objectives. Several distinct approaches were evaluated before finalizing one based on the accuracy, consistency, and robustness of the method. The methods are discussed in detail in the following sub-sections.

3.5.1. Histogram Matching and CAMShift

In this approach, an initial search window with the treasure object was chosen and was converted to the HSV colour space to compute the histogram. On receiving a new frame, Backprojection was performed using the stored histogram to generate a probability distribution indicating the degree of similarity between each pixel in the frame and the histogram of the treasure object. Thresholding was applied to retain pixels with matching histogram values. An update was performed using the CAMShift algorithm to locate the target object within the frame.

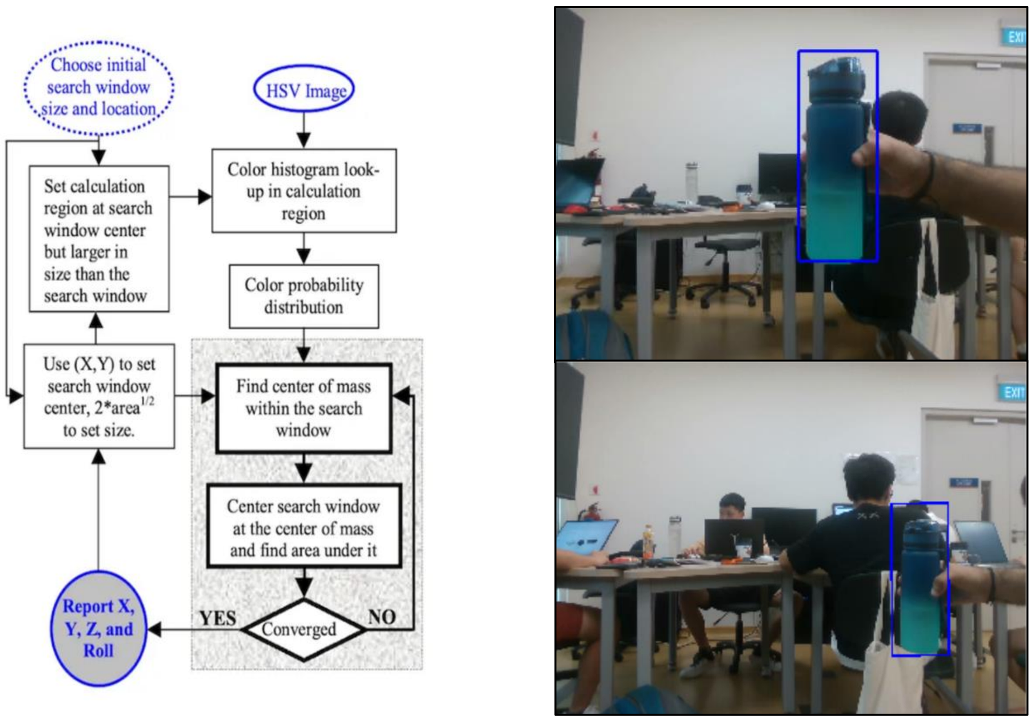

Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution
Use (X,Y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at the center of mass and find area under it
Report X, Y, Z, and Roll
YES
Converged
NO
EXIT

Figure 7: Object Recognition using CAMShift. Framework (left). Results (right).

Although, this method performed decently under certain conditions, it had plenty of the drawbacks:

- Similar histograms would result in incorrect identification leading to false positives.
- Different lighting conditions would lead to different histogram values for the same object resulting in failure to identify the object of interest.
- Re-introducing the object would sometimes result in not correct identifying or partially identifying the object.

These drawbacks can be observed from following figures (Figure 8).

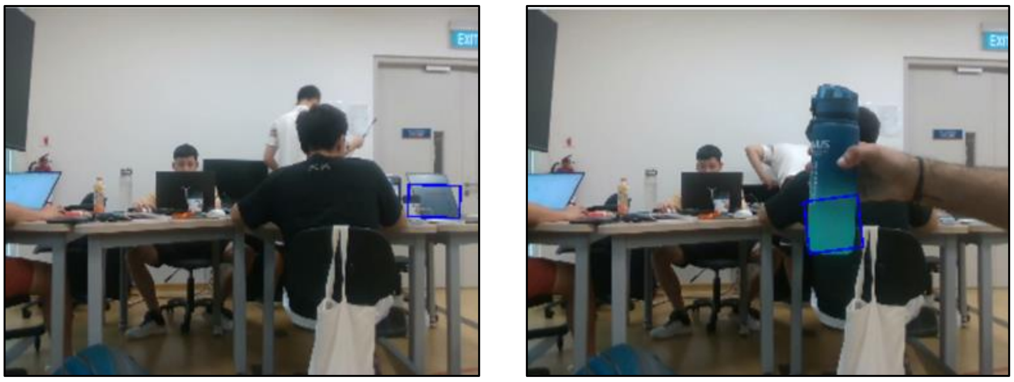


Figure 8: Drawbacks of CAMShift. False positives (left). Incorrect identification (right).

3.5.2. Feature Matching using ORB Feature Detector

In this approach, feature matching utilized to locate the object of interest within the frame. An initial ROI window containing the treasure object was selected and features were extracted followed by computing feature descriptors. Upon receiving a new frame, feature descriptors were computed and matched with those extracted from the treasure object. The location of the object was estimated based on the location of the region with maximum concentration of good matches.

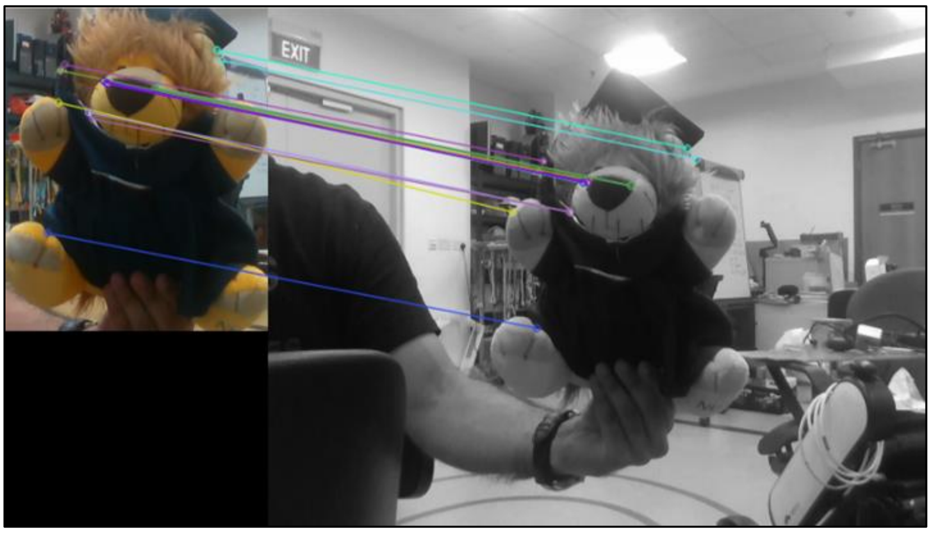


Figure 9: Object Recognition using feature matching.

Although this method performed better compared to the previous method, it still had its share of drawbacks:

- It was less effective with smooth surfaced objects.
- Occlusions, scaling and rotations could the quality of matches leading to false negatives.
- Colour information was lost as ORB feature extraction required images to be grayscale.

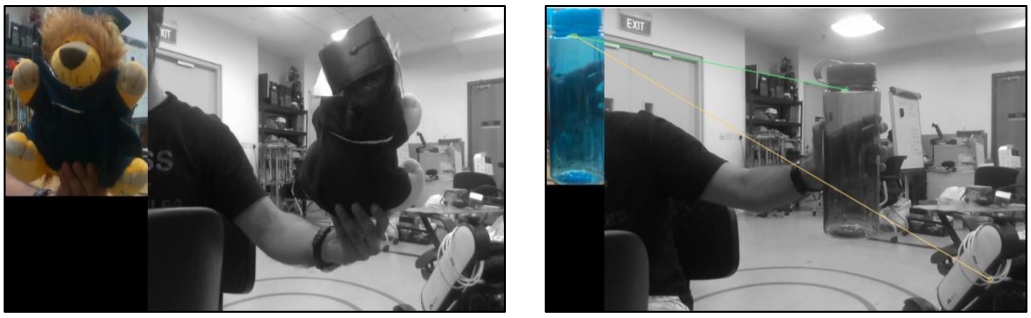


Figure 10: Drawbacks of feature matching. A complete 180' rotation results in no recognition (left). Feature extraction fails on smooth surfaces (right).

From the above figures (Figure 10), the shortcomings of this approach were apparent. Given the drawbacks, it was imperative to come up with a more robust solution to re-recognize objects.

3.5.3. Combining Histogram Matching with Object Detection

In this approach, an initial window containing the desired object was chosen followed by computation of its histogram and storage in memory. Upon receiving a new frame, object detection was performed, and the inferences were checked to determine if one or more detections belonged to the same object class as that of the treasure object. If an object of the same class was determined to be in the frame, its histogram was computed and matched with the histogram of the treasure object. If the similarity between histograms exceeded a certain threshold, the object would be confirmed as the treasure object. The outline of the approach has been illustrated in Figure 11.

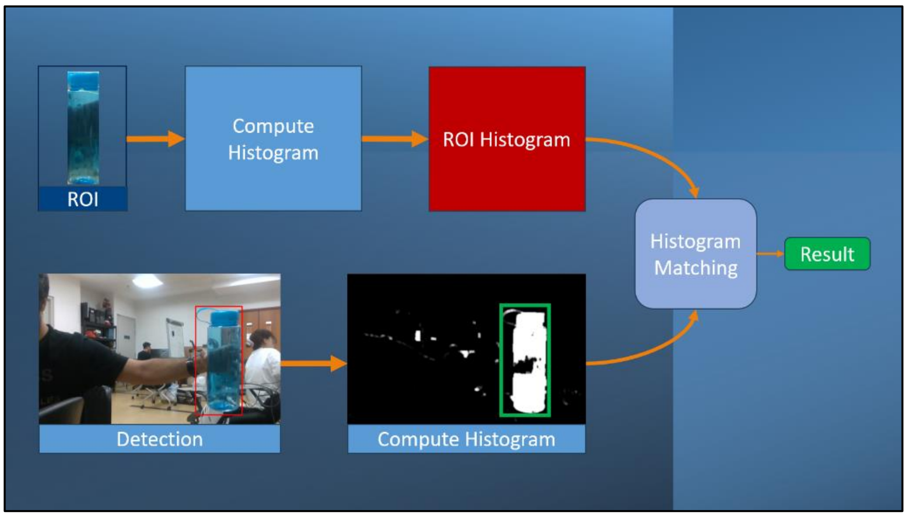


Figure 11: Object Re-recognition using Object Detection and Histogram Matching

The advantages offered several advantages including:

- Leveraged colour information for recognition.
- Since histograms are matched only for the objects of the same class, it prevented false positives ensuring that the re-recognition was more stable and less prone to errors.

- Having a base layer to check the presence of objects of the same class as that of treasure made the stack redundant.
- The stack could recognize and confirm the treasure even in the presence of multiple objects belonging to same class as that of the treasure.

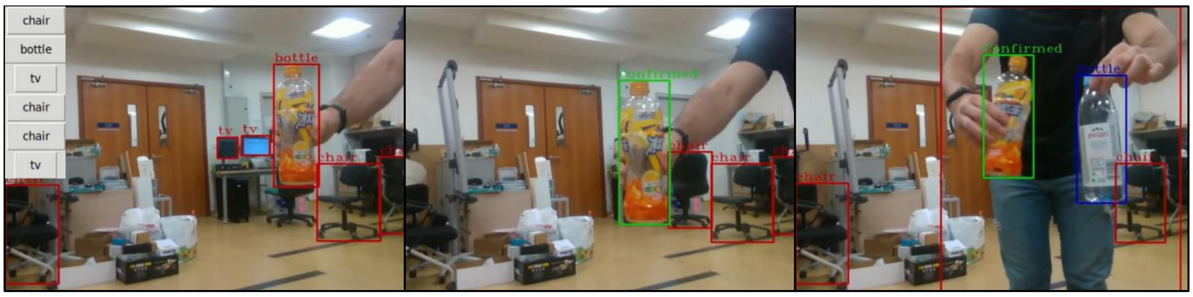


Figure 12: Re- recognition illustration. Object is selected (left). The stack recognizes and confirms the object (centre). The stack recognizes the selected object in presence of a similar object (right).

It could be observed from Figure 12 that this approach was not only able to re-recognize the selected bottle accurately, but also could differentiate between the selected bottle and a different bottle and correctly identify the selected bottle. It demonstrated that combining object detection and histogram could accurately identify and locate the target objects within the frame. As a result, this method was integrated in the final pipeline.

3.6. Object Tracking using DeepSort

In this project, object tracking was implemented to track the treasure object upon its discovery. Integrating object tracking into the pipeline addressed the imperfections and potential skips in the inferences and ensured smooth and consistent track of the object of interest.

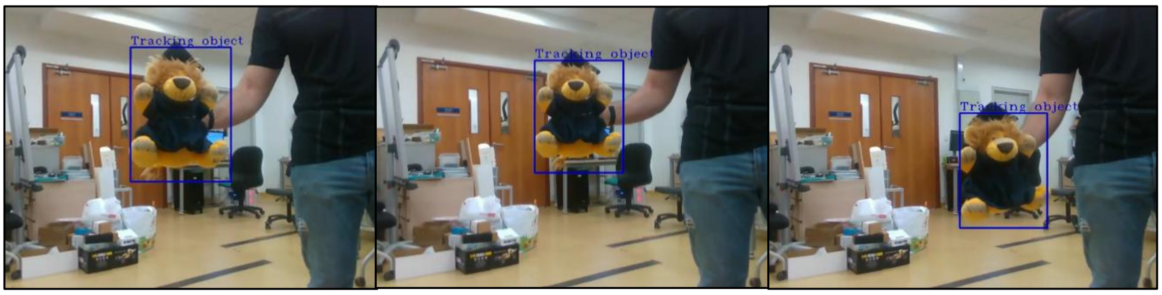


Figure 13: Object Tracking using DeepSort.

The DeepSort framework was used because of the ability of deep neural networks extract feature embeddings and appearance descriptors from the tracking window to refine the quality of predictions. The framework could be coupled with any state-of-the-art detector and achieve real-time performance. The following figure (Figure 14) shows the outline of the DeepSort Framework.

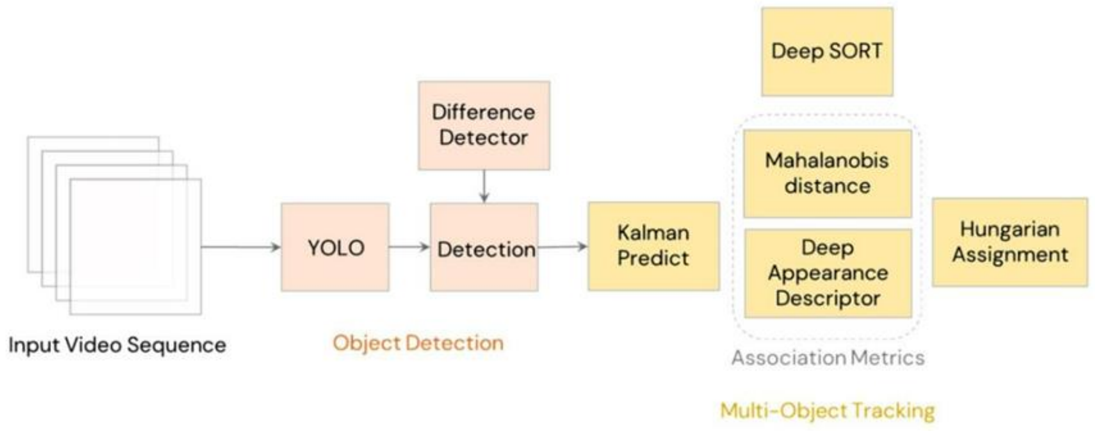


Figure 14: Framework for DeepSort.

Moreover, from Figure 15, it can be observed that although the detector failed to detect the treasure object in the second frame (bottom figure), the tracker was able to maintain its track highlighting its role in ensuring consistency.

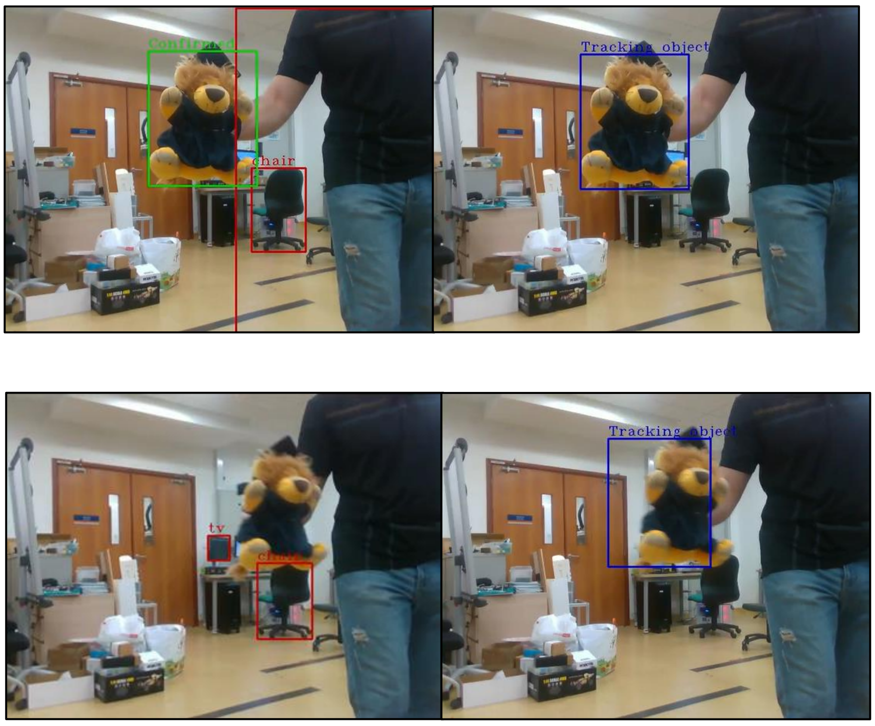


Figure 15: Tracking the target object. Detector and Tracker both locate the object (top). Detector fails but tracker still locates the object (bottom).

3.7. Unified Perception Pipeline

Once the core functionalities of the perception stack were finalized, it was necessary to integrate all the components into one unified pipeline (Figure 16). This section discusses the pipeline's operation in detail, which can be delineated into three key parts:

1. Detection and Object Selection
2. Goal Point Selection and Publishing
3. Object Confirmation

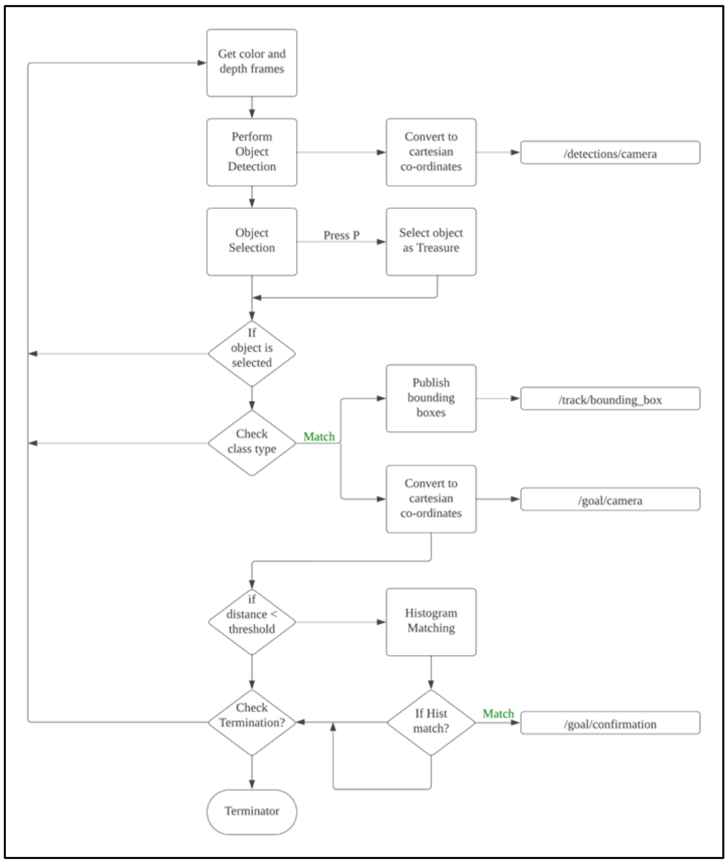


Figure 16: Complete Pipeline for the Perception Stack

3.7.1. Detection and Selection

This part of the pipeline is responsible for obstacle detection and enable object selection. Its operation unfolds as follows:

- Object detection is enabled immediately upon the initialization of the pipeline.
- The centre coordinates of the bounding boxes of the detections are transformed from the image frame to Cartesian coordinates in the camera frame by leveraging depth information and point deprojection (refer to Section 3.3).
- These 3D coordinates are then published to a ROS topic to be utilized by the Navigation stack.
- To select an object as the treasure, the user needs to press 'P' to open the object selection window and select one object from the detections as the treasure object.

- On selecting the object, its histogram is computed and stored in the memory.

3.7.2. Goal Point Selection and Publishing

Once the treasure is selected, the pipeline checks if one or more detentions in the subsequent frames correspond to the same class type as that of the treasure. If a match is found, the pipeline:

- Publishes the bounding boxes of the object of interest to a ROS topic to be utilized by the tracker.
- If multiple objects of interest are present, the pipeline selects the latest detection and converts the centre point of its bounding box to a 3D point in the camera frame.
- The 3D point is then publishing over a ROS topic as a goal point to be utilized by the navigation stack.

3.7.3. Object Confirmation

As the robot moves approaches the object of interest, it becomes imperative for the perception stack to confirm whether the object is the designated treasure object. This is achieved by:

- When the object is within a certain distance threshold of the robot (default: 0.6m), the pipeline computes the histogram of the object.
- The histogram is compared to the histogram of the treasure object.
- If the histograms match with more than 65% similarity, the object of interest is confirmed to be the treasure object and a flag is triggered to stop the exploration.
- If the histograms don't match, the object is discarded, and the robot continues to explore the environment to locate the treasure.

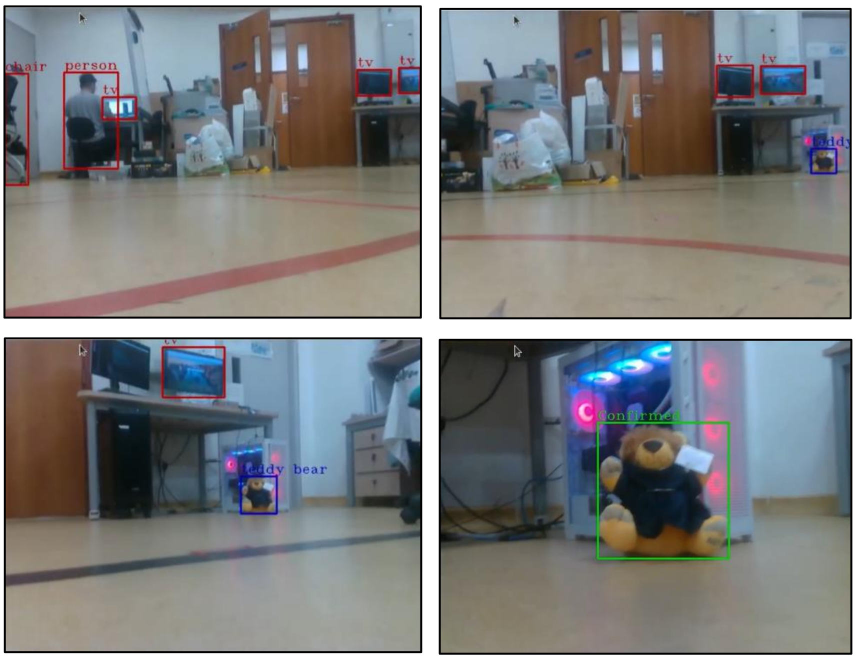


Figure 17: (Top left) Robot starts exploring. (Top right) Robot locates the treasure. (Bottom left) Robot approaches the treasure. (Bottom right) Robot confirms the treasure.

Figure 17 illustrates the operation of the pipeline. The treasure is selected and concealed within the environment. The vehicle then initiates the exploration process. The vehicle locates the object and starts approaching it (Figure 17, top right, and bottom left). Once the vehicle is close enough, it computes and performs histogram matching, thus confirming the object (Figure 17, bottom right).

3.8. ROS Packages

The pipeline includes two ROS packages: Perception package and RealSense package. These packages were crafted from scratch using the necessary Python libraries and the Python ROS API.

3.8.1. Perception Package

This package contains two executable ROS nodes:

- object_detector_node.py
- object_tracker_node.py

Launch File Parameters:

- model_name (default: 'yolov8m')
- tracker (default: True)

The object tracker node can be toggled on and off by setting the tracker parameter

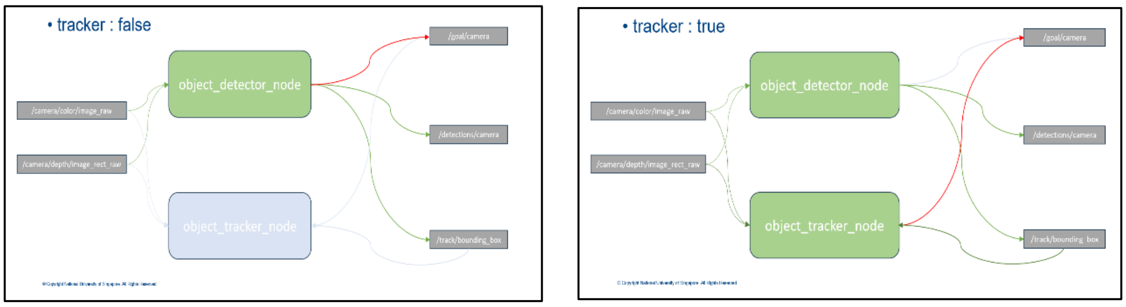


Figure 18: Outline of perception ROS package. Tracker node disabled (left). Tracker node enabled (right).

to True of False. Disabling the object tracker can help minimize GPU utilization, thus freeing up some computational resources. Toggling off the tracker node does not affect the functioning of the pipeline as illustrated in Figure 18, but can affect the consistency at which the treasure object is located within the frame.

3.8.2. RealSense Package

This package is a ROS wrapper for the RealSense D4000 series camera to publish colour frames, depth frame and camera intrinsics over ROS topics. This package includes one executable ROS node.

- Realsense_ros_node.py

Launch File Parameters:

- depth_width (default: 640)
- depth_height (default: 480)
- color_width (default: 640)
- color_height (default: 480)

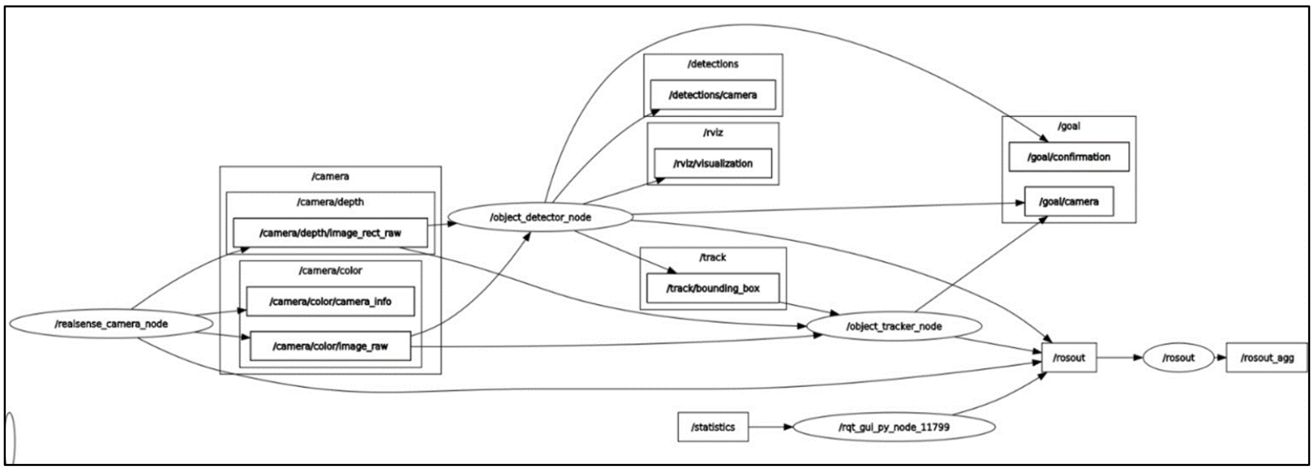


Figure 19: Perception pipeline with ROS nodes and topics.

4. **Inference**
   The Perception pipeline enables the autonomous robot to:
   - Detect static and dynamic obstacles in real time.
   - Select any one of the detections as a treasure object. The treasure object can be changed at any time during the exploration process.
   - Locate and correctly re-recognize the treasure object upon its discovery in the environment.
   - Track the treasure object in real time to maintain consistency. Also capable of tracking multiple objects of the same class type in real time.

5. **Limitations**
   The perception stack exhibits strong performance for most of the time. Although, the performance can be affected by:
   - Variations in the lighting conditions, distance and other factors which can affect the histogram values, thus affecting the re-recognition performance.
   - The pipeline at times can be susceptible to occlusions, rotations, and scaling.
   - Since re-recognition depends on object detection, failure to detect an object of interest can affect the recognition performance.
   - The pipeline may face challenges re-recognizing objects having similar characteristics like size and colour.

6. **Future Scope**
   - Optimize and refine the pipeline to improve performance and make it more robust.
   - Work on improvements to address the issues arising because of occlusions, different lighting conditions and orientations.
   - Development of a custom YOLO model to minimize the hardware resource utilization and faster inference times.
   - Establish a separate C++ pipeline to facilitate faster execution times.
   - To work on an improved object re-recognition stack which is independent of the inferences generated by the object detector.
   - Work on a lightweight tracking algorithm.

7. References

[1] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. “You Only Look Once: Unified, Real-Time Object Detection.” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[2] Girshick, R. “Fast R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” IEEE International Conference on Computer Vision (ICCV), 2015.

[3] Mahendrakar, T., Ekbald, A., Fischer, N., White, R., Wilde, M., Kish, B., and Silver, I. “Performance Study of YOLOv5 and Faster R-CNN for Autonomous Navigation around Non-Cooperative Targets.” IEEE Aerospace Conference (AERO), 2022.

[4] O. -D. Nouar, G. Ali and C. Raphael, "Improved Object Tracking with Camshift Algorithm," 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, Toulouse, France, 2006.

[5] Bohyung Han and L. Davis, "Object tracking by adaptive feature extraction," 2004 International Conference on Image Processing, 2004. ICIP '04., Singapore, 2004.

[6] A. Pujara and M. Bhamare, "DeepSORT: Real Time & Multi-Object Detection and Tracking with YOLO and TensorFlow," 2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), Trichy, India, 2022.