Project Aqua – Technical Challenges Submission
Name: Masnoon Tahmid - 230011254

Part II – Technical Challenges

Problem 1: ROV Control Mission

Objective

The objective of this task was to implement the control logic for an underwater ROV by connecting pilot inputs to hardware drivers. The ROV uses differential steering with two thrusters (left and right) and a ballast tank system for depth control.

Part A: Button Controls

I implemented the following functions inside student-task.js:

- moveForward()
- moveBackward()
- turnLeft()
- turnRight()
- ascend()
- descend()

For forward movement, both left and right motors are set to a positive speed value (e.g., 500). For backward movement, both motors are set to a negative speed value.

Turning was implemented using differential speeds:

- To turn left, the left motor speed is reduced and the right motor speed is increased.
- To turn right, the opposite is done.

For depth control:

- injectorPump(speed) is used in descend() to add water and increase weight.
- ejectorPump(speed) is used in ascend() to remove water and decrease weight.

The system automatically stops the motors when the button is released, so only speed values were set inside the functions.

Part B: Joystick Control

The function handleJoystickControl() was implemented to mix joystick inputs 60 times per second.

Inputs used:

- joystickLeft.y → Forward/Backward
- joystickLeft.x → Turning (Yaw)

Motor mixing logic:

leftSpeed = forward + turn
rightSpeed = forward - turn

This allows:

- Straight motion when turn = 0
- Rotation when forward = 0
- Combined movement when both inputs are active

The values were scaled properly to stay within motor limits.

Problem 2: ROV Telemetry Monitoring System

Objective

The goal was to design a backend telemetry system and a frontend dashboard to simulate real-time ROV sensor monitoring.

Backend Implementation

A Node.js server was built using the core http module.

Features Implemented:

• On startup, the server loads 500.json sensor data
• POST /api/telemetry

- Accepts depth, pressure, temperature, direction, timestamp
- Validates values
- Rejects invalid data with 400 status codes

• GET /api/telemetry/latest

- Returns most recent telemetry entry

• GET /api/telemetry/history?limit=N

- Returns last N records

Data Storage

- Maximum of 100 entries stored
- FIFO logic used (oldest removed when limit exceeded)

Frontend Dashboard

Built using:

- HTML
- CSS
- Vanilla JavaScript
- Chart.js

Features:

- Polls /latest every 5 seconds
- Displays:
    - Depth
    - Pressure
    - Temperature
    - Direction

- Shows system status based on pressure:

NORMAL → < 1.8 bar
WARNING → 1.8–2.0 bar
CRITICAL → > 2.0 bar

A visual alert is shown for CRITICAL status.

Graph

Depth vs Time is plotted dynamically using Chart.js.

Telemetry Simulation

Instead of random data generation, the backend streams entries from 500.json every 5 seconds to simulate live ROV operation.

This approach makes the system predictable and easier to test.

Problem 3: Underwater Waste Detection using YOLO

Objective

The objective of this task was to train a YOLO-based object detection model to identify underwater waste:

- bottle
- polythene
- styrofoam

Data Curation

The dataset provided contained underwater images of waste materials.

Steps followed:

1. Images uploaded to Roboflow

   Roboflow Project URL: https://public.roboflow.com/project/your-project-name

2. Annotated into 3 classes:

    - bottle
    - polythene
    - styrofoam
3. Dataset split into:
    - 70% Training
    - 15% Validation
    - 15% Testing
4. Augmentation applied:
    - Horizontal flip
    - Brightness adjustment
    - Minor rotations

Dataset exported in YOLO format.

Total images after augmentation: ~123

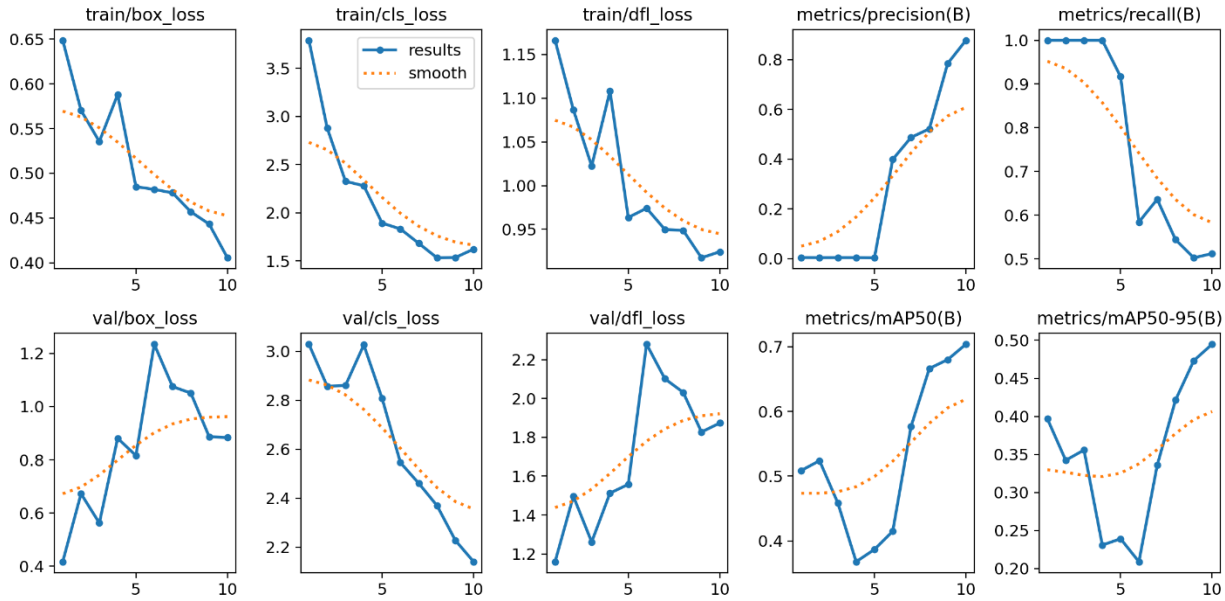
Training Configuration

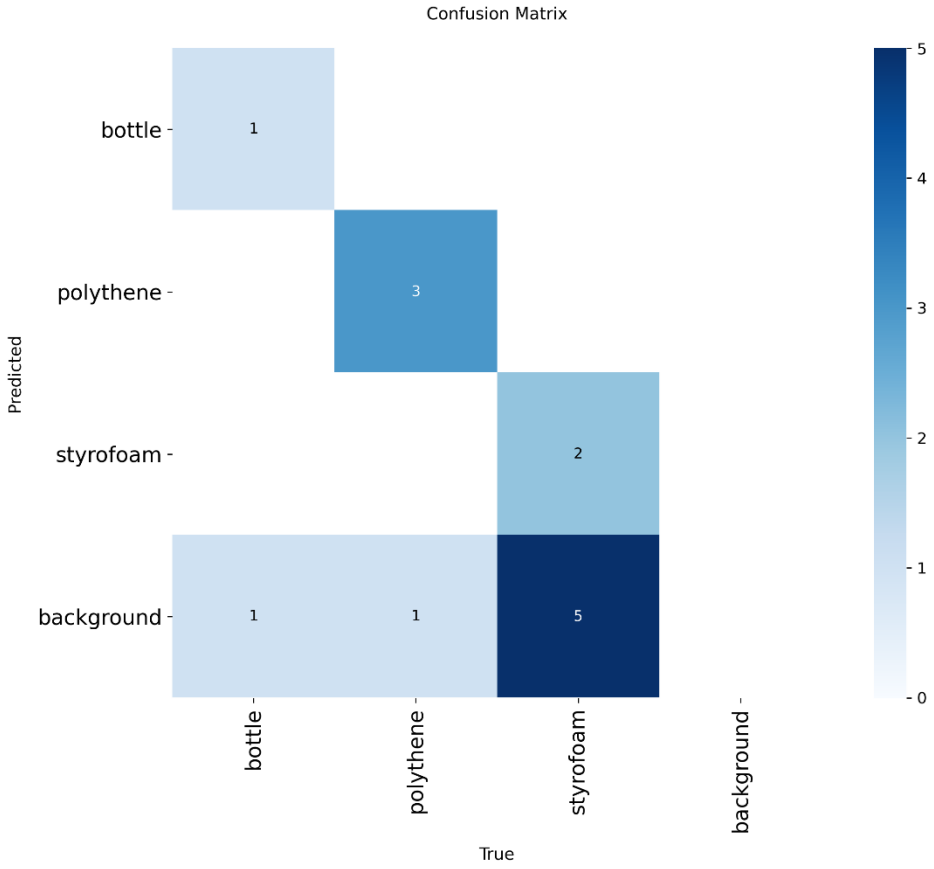
Model used: YOLOv8n (Ultralytics)
Epochs: 10
Batch size: 8
Image size: 640
GPU: RTX 3060
Framework: Python 3.10 + Ultralytics

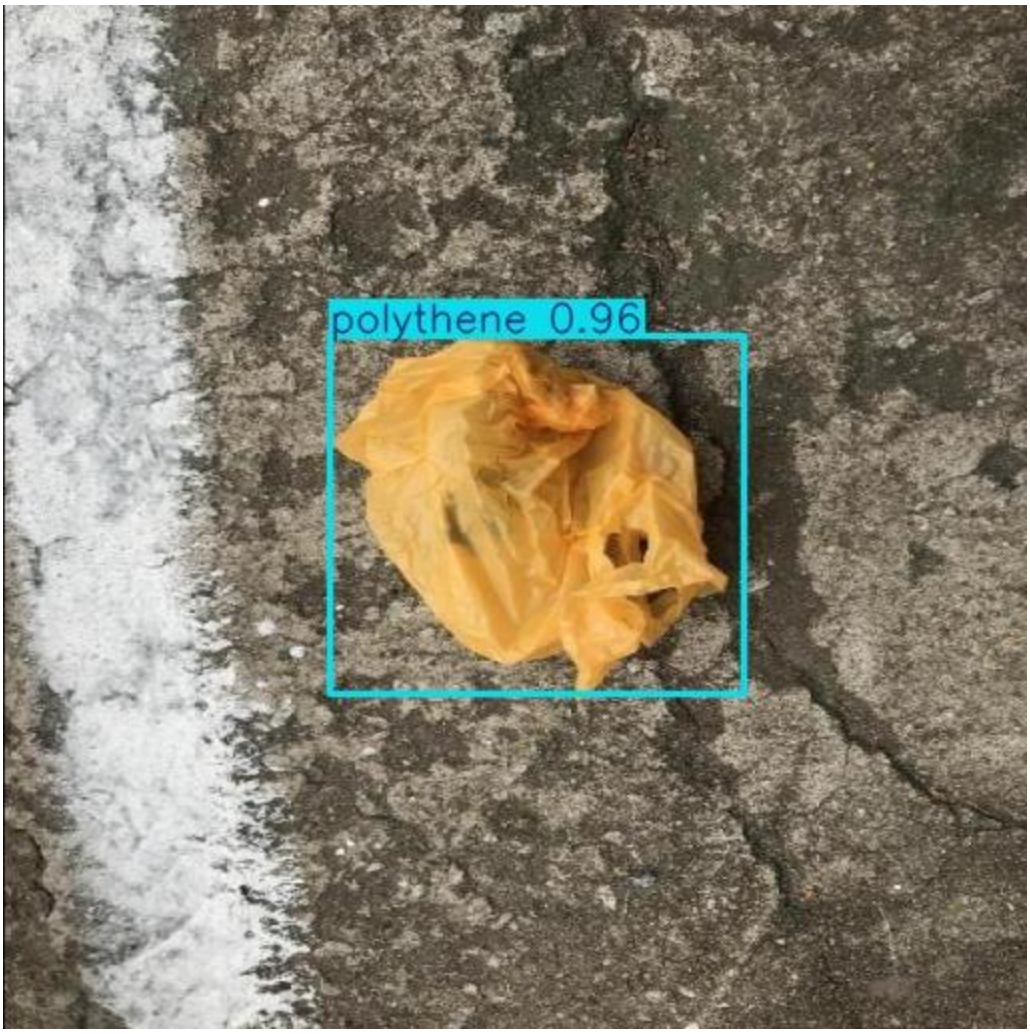
Training was executed using:

model.train()
model.val()

Results


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)


Confusion Matrix
bottle
polythene
styrofoam
background
Predicted
True
1
3
2
1
1
5
5
4
3
2
1
0


polythene 0.96


polythene 0.67

Overall performance:

- mAP@0.5 = 0.706
- mAP@0.5–0.95 = 0.496

Class-wise performance:

Bottle:

- mAP@0.5 = 0.578

Polythene:

- mAP@0.5 = 0.745

Styrofoam:

- mAP@0.5 = 0.794

The model performed reasonably well considering the small dataset size.

The following graphs were generated:

- mAP@0.5 vs epoch
- mAP@0.5–0.95 vs epoch
- Box loss vs epoch
- Class loss vs epoch
- Normalized confusion matrix

Annotated predictions were saved and tested on unseen images.

Discussion

The model shows good detection performance for polythene and styrofoam.

Bottle detection was slightly lower, possibly due to fewer samples or variations in shape and lighting.

Because the dataset size is small, there is potential overfitting. Increasing dataset size and stronger augmentation would likely improve performance.

Conclusion

All three tasks were successfully implemented.

The ROV control logic allows smooth directional and depth movement.

The telemetry system provides real-time monitoring and visualization of underwater data.

The YOLO model successfully detects underwater waste objects with acceptable performance for a limited dataset.

With more data and longer training, the system can be further improved for real-world deployment.