(EE-333)

DIGITAL IMAGE PROCESSING

DE 43 – Mechatronics

Syndicate – A

Project Report

ROBOTIC VISION SYSTEM

Group Members:

- Mohamed Nalim #398504
- Muhammad Waseem Irfan #394774

Submitted to: Dr Tahir Nawaz

ROBOTIC VISION SYSTEM FOR MODULAR SELF-RECONFIGURABLE ROBOTS

TABLE OF CONTENTS:

1. Introduction

Robotic vision is a critical component in modern robotics, enabling machines to perceive and interpret their surroundings. This project aims to develop a robotic vision system for modular self-reconfigurable robots that can work individually and collectively. These robots need to differentiate between various objects, obstacles, and other robots in their environment. This project, specifically designed for the digital image processing course, focuses on developing an image process using YOLO (You Only Look Once) for object detection and segmentation and some other techniques.

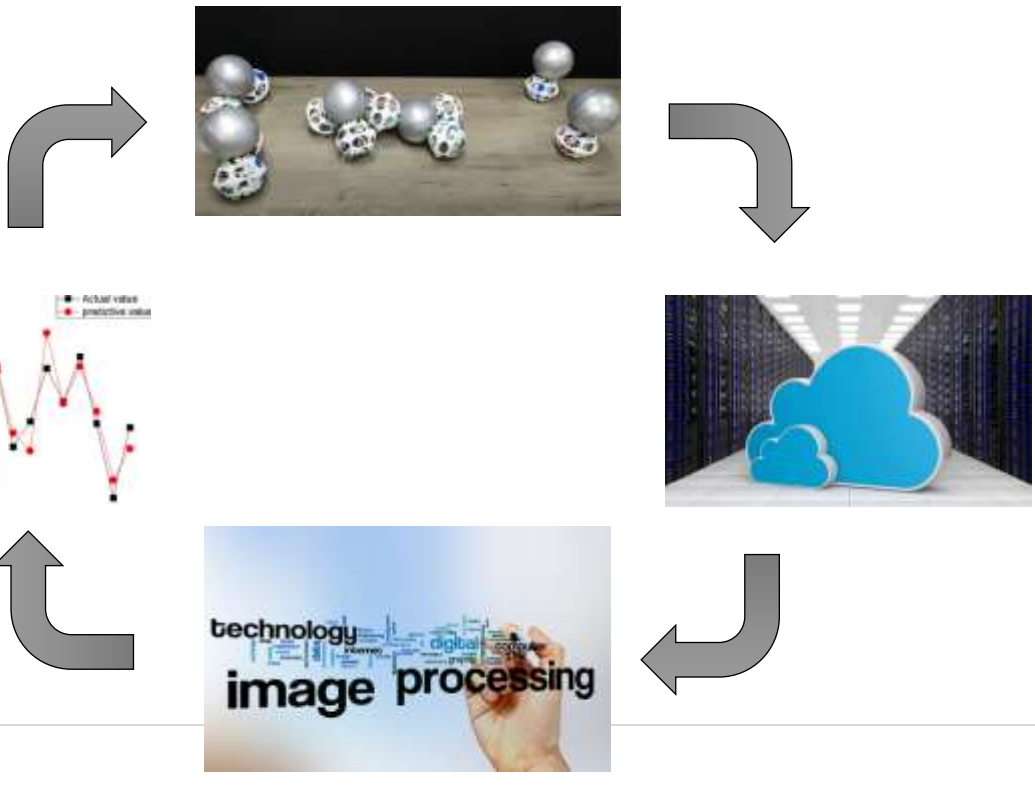
2. Objectives

The objective of this project is to leverage YOLOv8 for effective image segmentation using a custom dataset. By utilizing the data set and annotating objects with the CVAT tool, the project aims to train a robust model on Google Colab, integrating Google Drive for seamless data management. The training process involves meticulous data handling, model training, and validation using performance metrics such as training and validation loss. The ultimate goal is to achieve accurate image segmentation, evaluate model performance, and explore future enhancements in computer vision tasks.

3. System Architecture

The system architecture consists of several key components:

- Robots with Integrated Cameras: Capture live video or images.
- Cloud Server: Receives data from the robots, processes it, and returns predictions.
- Image Processing Pipeline: Utilizes YOLO for object detection and segmentation.
- Prediction Results: The processed data is used to make predictions, which are then sent back to the robots.



4. Dataset Preparation

Sample data set

Dataset Description

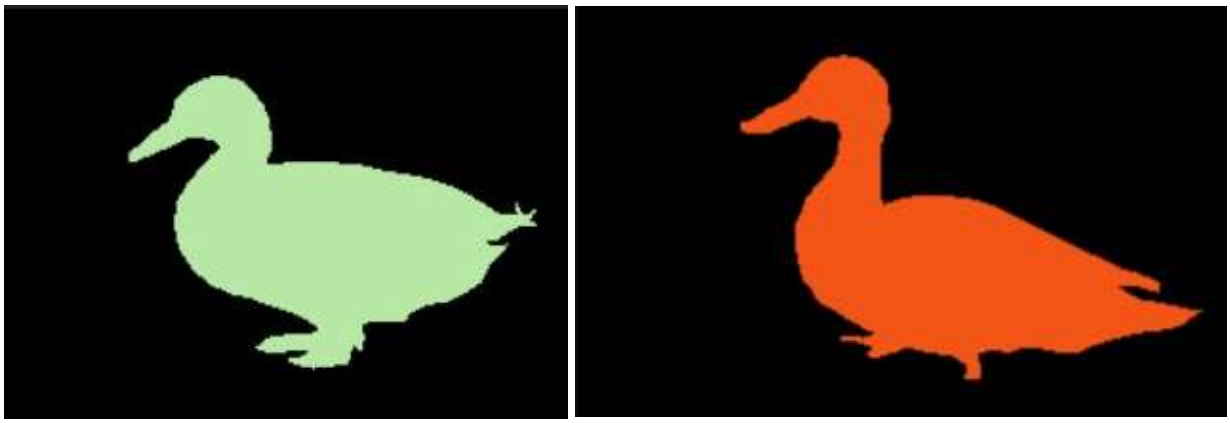
For this project, we initially used images of ducks to develop and test our image processing pipeline. This allowed us to focus on the image processing and detection tasks without needing specific data from the robots. We download the pictures of ducks from google and used that as our data set.

Preprocessing

The preprocessing steps involved cleaning the dataset by removing any corrupted or irrelevant images. This ensured that the data used for training was of high quality. Here we improved the quality of the images/data.

5. Data Annotation

Manual annotation



Binary masking after annotation

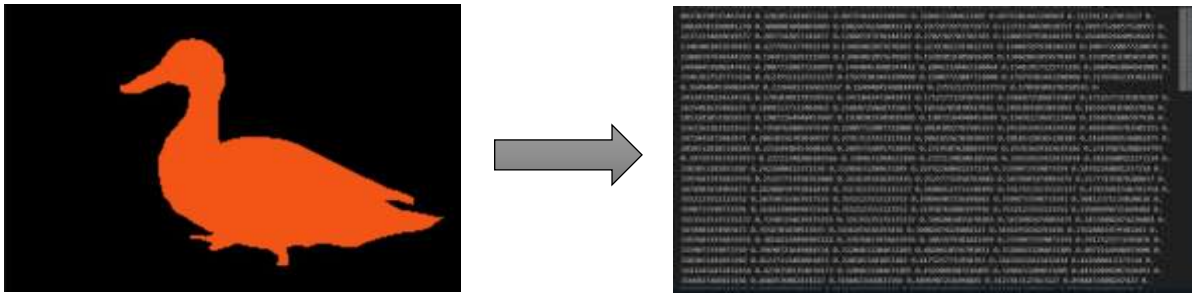
Tool Used

We used CVAT (Computer Vision Annotation Tool) for annotating the images. CVAT is a versatile tool that allows for detailed image annotation, making it suitable for our needs.

Annotation Process

The annotation process involved segmenting the objects in each image and labeling them appropriately. This provided the necessary ground truth data for training our YOLO model. We did all these steps manually so we can follow the same process for robot detection in future updates.

6. Conversion to YOLO Format



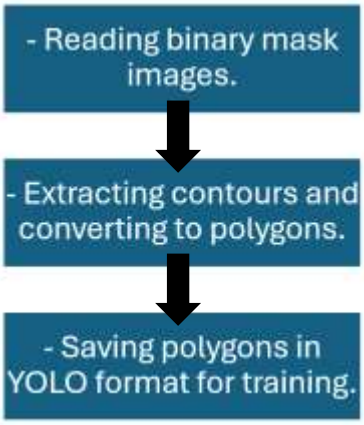
Conversion of binary image to YOLO format

Format Details

YOLO requires data to be in a specific format, where each annotation includes the class label and normalized bounding box coordinates. Our datas are not in this format, so we implement some conversions here.

Conversion Process

We developed custom scripts to convert the annotated data into YOLO format. This involved reading the binary mask images, extracting contours, converting them into polygons, and saving these polygons in the required format.



Conversion code architecture

```python
import os
import cv2

input_dir = './image-segmentation-yolov8-main/mask'
output_dir = './image-segmentation-yolov8-main/data/labels/train'

for j in os.listdir(input_dir):
    image_path = os.path.join(input_dir, j)
    # load the binary mask and get its contours
    mask = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    _, mask = cv2.threshold(mask, thresh=1, maxval=255, cv2.THRESH_BINARY)

    H, W = mask.shape
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # convert the contours to polygons
    polygons = []
    for cnt in contours:
        if cv2.contourArea(cnt) > 200:
            polygon = []
            for point in cnt:
                x, y = point[0]
                polygon.append(x / W)
                polygon.append(y / H)
            polygons.append(polygon)

    # print the polygons
    with open('{}.txt'.format(os.path.join(output_dir, j)[:-4]), 'w') as f:
        for polygon in polygons:
            for p_, p in enumerate(polygon):
                if p_ == len(polygon) - 1:
                    f.write('{}\n'.format(p))
                elif p_ == 0:
                    f.write('0 {} '.format(p))
                else:
                    f.write('{} '.format(p))

        f.close()
```

Code snippet of conversion

7. Model Training

Framework

We used YOLOv8 for training our model. YOLOv8 is the latest version, offering improved accuracy and speed.

Environment

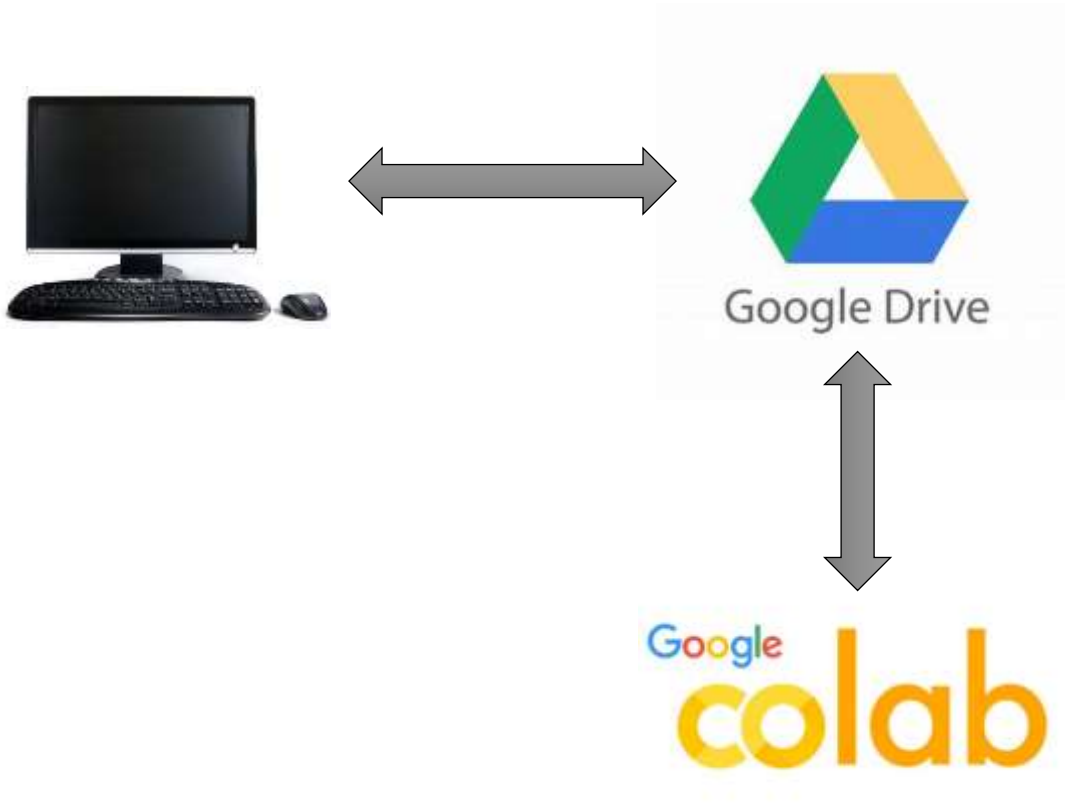
The training was conducted on Google Colab, which provides powerful GPU support necessary for training deep learning models efficiently.

Training Steps

1. Data Upload: The annotated dataset was uploaded to Google Drive.

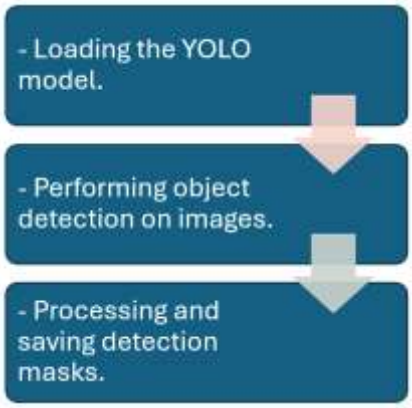
2. Mounting Drive: Google Drive was mounted in Colab to access the data.

3. Model Training: The YOLO model was trained with specified parameters such as epochs, batch size, and learning rate.



Work flow

8. Prediction and Evaluation



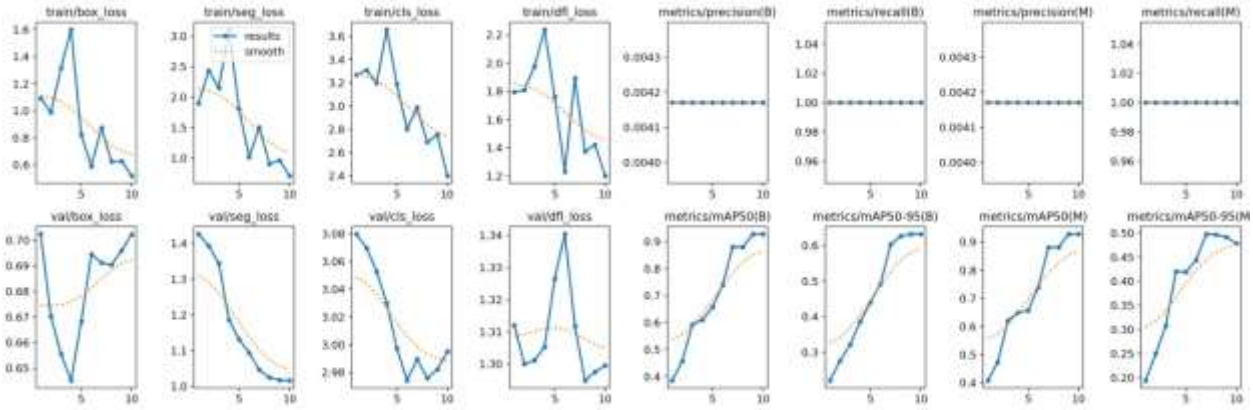
Prediction code architecture

```python
from ultralytics import YOLO

import cv2

model_path = './image-segmentation-yolov8-main/train4/weights/last.pt'
image_path = 'image-segmentation-yolov8-main/data/Images/val/duck7.jpg'
img = cv2.imread(image_path)
H, W, _ = img.shape
# Load the YOLO model
print(f"Loading YOLO model from: {model_path}")
model = YOLO(model_path)

# Perform object detection
print(f"Performing detection on image: {image_path}")
results = model(img)
# Check and log the number of results
print(f"Number of detection results: {len(results)}")
# Process and save masks
mask_detected = False
model = YOLO(model_path)

results = model(img)
# print(results)
for result in results:
    if result.masks is not None:  # Check if masks were detected
        mask_detected = True
        for mask in enumerate(result.masks.data):
            mask = mask.numpy() * 255  # Convert mask to a 2D numpy array and scale to [0, 255]
            mask = cv2.resize(mask, dsize: (W, H))  # Resize the mask to match the image dimensions
            cv2.imwrite( filename: './image-segmentation-yolov8-main/output.png', mask)
            # cv2.imshow("window",mask)
            # cv2.waitKey(0)
            # cv2.destroyAllWindows()
```

Code snippet of conversion

Performance Metrics

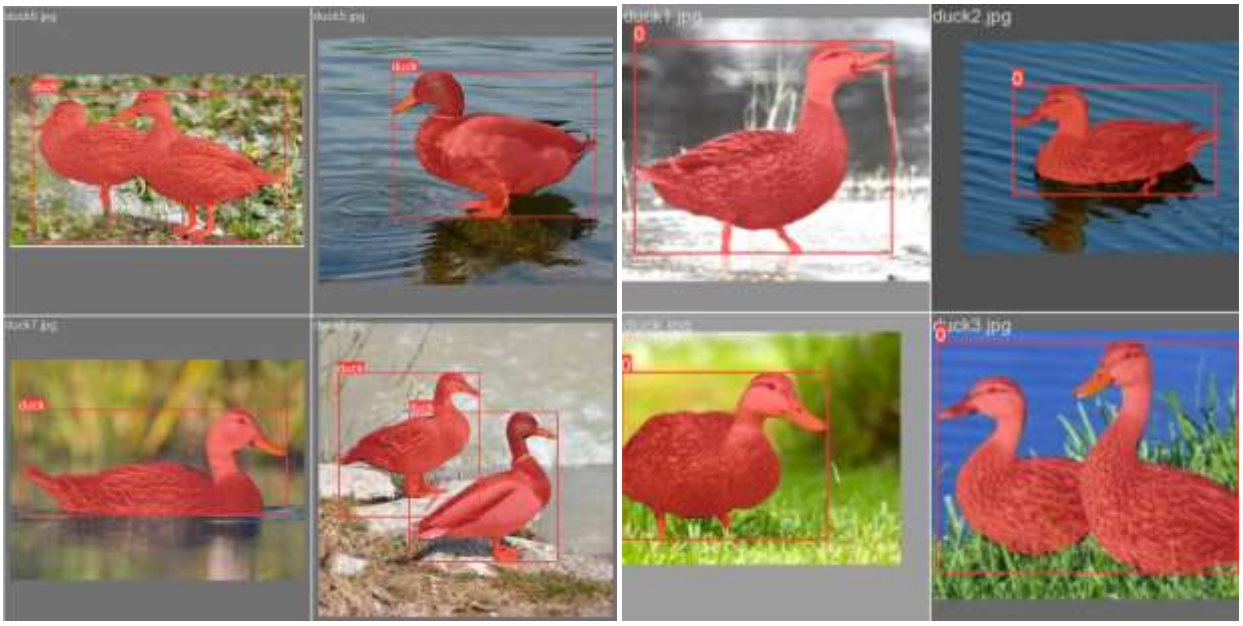
We evaluated the performance of our model using metrics such as precision, recall, and loss. These metrics helped us understand the accuracy and reliability of our predictions.



Output graphs

9. Results

The results demonstrated the effectiveness of our model in real-time scenarios. We showcased key results with images highlighting the segmented objects and predictions made by the model.



System detecting ducks

10. Discussion

Interpretation of Results

The results of our project highlight the effectiveness and practicality of using YOLO for object detection and segmentation in a robotic vision system. Our model was able to accurately detect and segment objects in the images, demonstrating the potential for real-time application in a robotic environment. The precision and recall metrics indicate that the model performs well in identifying and localizing objects, which is critical for the intended application in modular self-reconfigurable robots.

Challenges and Limitations

1. Data Quality and Quantity:
   One of the significant challenges was the quality and quantity of the dataset. While using duck images was a useful proxy for initial development, the lack of robot-specific data means that the model's performance on actual robot images remains to be fully tested.
2. Annotation Effort:
   Annotating the dataset was a time-consuming process. High-quality annotations are critical for training effective models, and this step requires considerable manual effort and precision.
3. Computational Resources:
   Training deep learning models like YOLO requires substantial computational resources. While Google Colab provided a useful platform, the limitations in processing power and memory occasionally hindered our progress.

11. Future Work

1. Collecting Robot-specific Data:
   For the final year project, we plan to collect and annotate images directly from the modular self-reconfigurable robots. This will enable us to train models that are fine-tuned for the specific challenges and environments these robots will encounter.
2. Advanced Models:
   Exploring and integrating more advanced models, such as YOLOv8 or transformer-based models, could further enhance detection accuracy and speed.
3. Real-time Implementation:
   Improving the real-time processing capabilities by optimizing the communication and processing pipeline will be a focus. This includes reducing latency in data transmission and processing.
4. Collaborative Robotics:
   Investigating how the vision system can support collaborative tasks among multiple robots, including obstacle avoidance, object manipulation, and coordinated movement.

12. Conclusion

In conclusion, this project successfully demonstrated the effectiveness of using YOLO for object detection and segmentation in a robotic vision system. Despite challenges like limited dataset quality and the effort required for annotation, the model showed high accuracy and real-time processing capabilities. Moving forward, collecting robot-specific data, and optimizing real-time processing will be crucial. This project

lays a solid foundation for integrating advanced image processing techniques into autonomous robotic systems, with potential applications in various fields.