Institute of Technology of Cambodia

Department of Electrical and Energy (GEE)

Object Detection with YOLO Model

Group: I4-GEE-EA-A2

Lecturer: Mr. IT CHIVORN

Student: LANG BANDITHVIPHO **ID:** e20200104

Academic Year: 2023 - 2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Project Background

Object detection is a well-known task in computer vision, involving the identification and localization of objects within images or video frames. Object detection has been in use since 2001, even before the advent of deep learning, and has evolved significantly with the development of deep learning techniques. Today, object detection is particularly popular and is widely used in applications such as object counting, self-driving cars, and various industrial and factory processes. We also apply the object detect on Robocon 2024 competion to help helps autonomous robots perform tasks effectively by identifying and responding to objects in their environment.

1.2. Objective

The primary objective of this project is to participate in the ABU Robocon Competition by performing object detection (system) to identify blue and red balls. It will allow a robot to complete challenges autonomously under the rules of the competition.

1.3. Scope of work

In this project we will work on the several steps:

- Data collection
- Annotate the dataset and label on it
- Setup training workspace
- Training the dataset with the selected model
- Hardware Implementation
- Result and Discussion

1.4. Outline of Report

There are somes flows which contains in this report:

- **Introduction**: Provide a background of Object detection Especially, the objective and scope of work in this report.
- **Research Methodologies**: Telling the procedure of the Object Detection setup with yolo Model selected.
- **Hardware Implementation**: This part indicate the hardware process setup.
- **Result and Discussion**: Show the result that get from the dataset training and discuss on the results.
- **Conclusion**: Conclude the results and achievement after integrate with hardware.

2. RESEARCH METHODOLOGY

2.1. What is object detection?

Object detection is a computer vision method used to find and identify objects in images or videos. It involves drawing bounding boxes around the detected objects to show their location and track their movement. For example, in **Figure 2.1**, the model detects and highlights balls using red boxes for blue balls and orange boxes for red balls. This technique is crucial for applications like surveillance, self-driving cars, and retail analytics, helping to monitor, identify, and analyze objects efficiently.

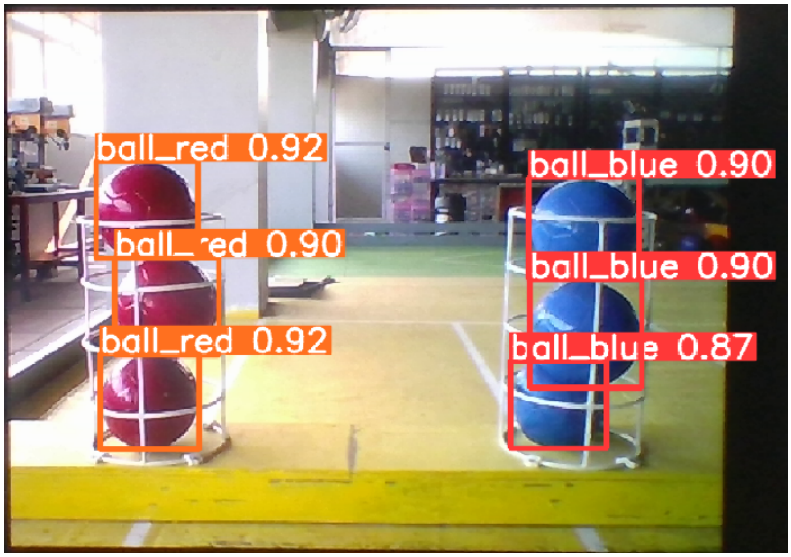

Figure 2.1. Object Detection

2.2. Overview of yolo model

We use the YOLO model to detect the color of the ball in object detection tasks. YOLO first introduced by Joseph Redmon and colleagues in 2015, YOLO (You Only Look Once) is a state-of-the-art deep learning model designed for real-time object detection in computer vision applications. Since its inception, YOLO has evolved through ten versions until present, with the latest, YOLOv5 to YOLOv10, developed by the Ultralytics team.

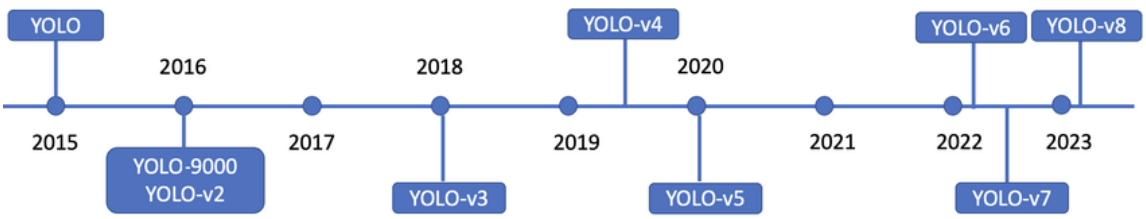

Figure 2.2. Timeline of YOLO model

YOLO is an object detection method that has good performance. YOLO uses a pre-trained Convolutional Neural Network (CNN) to process images in layers and output the bounding boxes and classes of detected objects. CNNs can perform a magnificent object recognition process from an image. CCN processes the input image and extracts increasingly complex features such as angle and texture. This technology is usually used for various applications such as security surveillance, product quality analysis, facial recognition, and others.

Figure 2.3. YOLOv8 Developed by Ultralytics Teams

As in figure, this study uses YOLOv8 to improve the detection accuracy of object detection. It evaluates the performance of the YOLOv8 method in detecting objects under different weather conditions.

2.3. YOLO Processing

In short, the YOLO model is trained on labeled datasets, optimizing the model parameters to minimize the difference between predicted bounding boxes and ground truth bounding boxes. With the help of bounding box coordinates and the class probability we not not only have the detected object, we also have the answer to object localization.

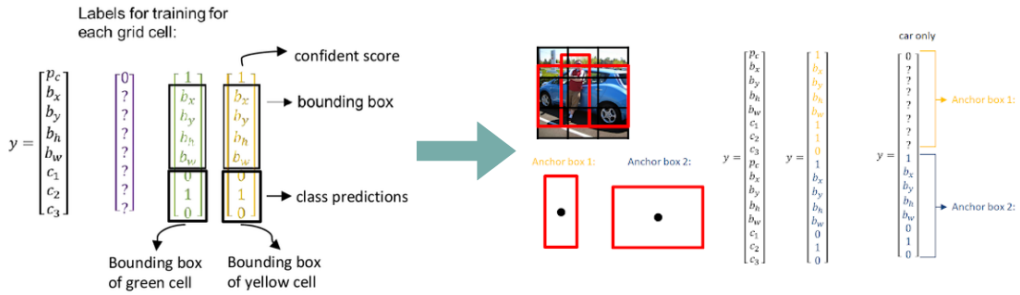

Figure 2.4. YOLO process algorithm

Now let's get into a bit more detail, and break down what the **Figure 2.4** just described. The YOLO algorithm takes an image as input and is passed to a deep Convolutional Neural Network,

which generates an output in the form of a vector that appears similar to this: $x, y, w, h, P_c, c_1, c_2, c_3$. For convenience, let us denote this vector by $\mathbf{n}$.

- Pc is the probability of the class which shows if an object is present or not.
- bx, by, bw, bh specifies the coordinates of the bounding box from the center point of the object.
- c1, c2, c3 represent the classes present in the image. For example, c1=1 if it is a dog and the rest will be 0. Similarly, if c2 represents a human, c2 will be equal to 1 and the rest of the classes will be 0. If there is no object present in the image, the vector will be [0, 0, 0, ..., 0]. In this case, the Pc will be 0 and the rest of the elements in the vector will not matter.
- This is fed to the neural network. Here we have provided one example, but in the real world a huge number of images are provided as the training set. These images are converted into vectors for each corresponding image. Since this is a supervised problem, the X_train, Y_train will be the images and the vectors corresponding to the image and the network will again output a vector

This approach works for a single object in an image, but if there are multiple objects in a single image. It will be difficult to determine the dimension output of the neural network.

2.4. Setup YOLO Model

To get data for object detection, follow these stages:

- Data preparation
- YOLOv8 model selection
- Training the Dataset

2.4.1. Data preparation

This stage involves collecting and organizing the data we'll use to train the model. we need images that contain the objects we want to detect, and each image should be labeled with the locations and classes of these objects. This can involve tasks like downloading images from the internet, taking images from camera frame, or using existing datasets. Then, we need to annotate the images, which means drawing bounding boxes around the objects and assigning labels to them.

Noted that: This annotate step is crucial because the quality and accuracy of your annotations directly impact your model's performance.

Figure 2.5. Example Dataset in Differences Condition

Additionally, in **Figure 2.5** this dataset focuses on scenarios found in Robocon 2024 competition game-fields, featuring blue and red balls. It covers various challenging conditions encountered during competitions situation, including the area for blue ball and area for red ball. The dataset includes images categorized into different game-field scenarios, as depicted in **Figure 2.5**, with classes for blue ball and red ball.

Annotation:

After preparing our dataset for each condition needed, the next stage involves annotating the images. We chose Roboflow for this task because it's a widely recognized and user-friendly framework, especially effective for labeling images in YOLO dataset training.

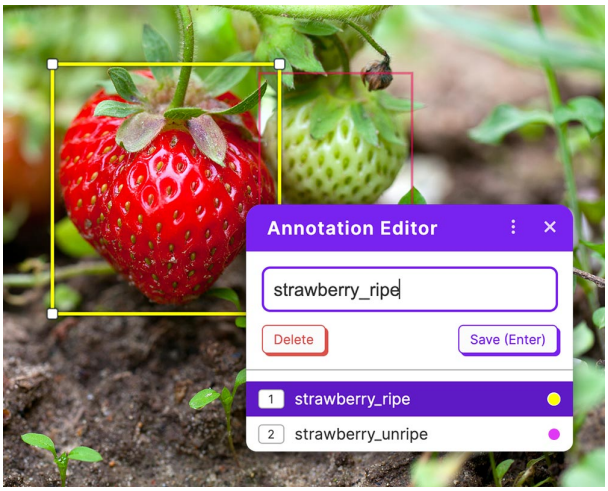

Figure 2.6. Example of images for annotation and label

As shown in the figure above, the square box around the object represents the bounding box concept in object detection. We need to label the object with its name inside this box.

2.4.2. YOLOv8 Model Selection

Once more, the Ultralytics team has conducted benchmarking of YOLOv8 using the COCO dataset, demonstrating significant advancements over previous YOLO versions across all five model sizes. YOLOv8 represents the latest iteration in the YOLO (You Only Look Once) series of object detection models, known for their speed and accuracy in real-time applications. The COCO (Common Objects in Context) dataset is widely used for evaluating object detection algorithms, providing a diverse set of images with annotated objects in various categories.

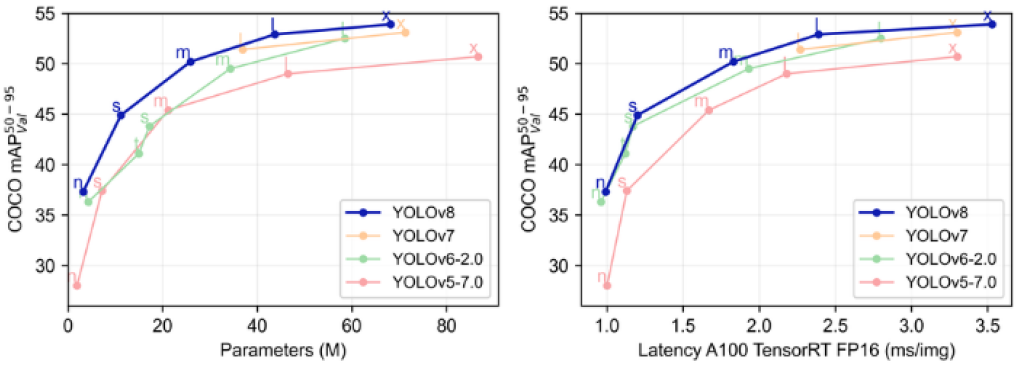

Figure 2.7. Benchmark Results Across YOLO (mAP:Performance, In fps: Speed of the inference, FLOPs and params: Compute or the model size)

In this **Figure 2.7**, the benchmarking results highlight improvements in detection accuracy, speed, and model efficiency, making YOLOv8 a promising choice for applications requiring robust and fast object detection capabilities. These advancements are crucial in fields such as autonomous driving, Health care, and robotics, where real-time detection of objects in complex environments is essential for decision-making and safety. Ultralytics' findings underscore YOLOv8's capabilities in pushing the boundaries of object detection technology, paving the way for enhanced performance and reliability in demanding scenarios.

Comparison Object detection model size for YOLOv8:

Object detection is the task that involves identifying the location and class of objects in an image or video stream. In **Figure 2.8**, we compare object detection performance across five different model sizes of YOLOv8. The YOLOv8m model obtained a mean Average Precision (mAP) of 50.2 % on the COCO dataset.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 2.8. The comparison of object detection across five different model sizes

The COCO dataset provides a diverse set of images with annotations for multiple object classes, making it an ideal standard for assessing model performance. Meanwhile, the YOLOv8x, the largest model among the set, achieved an impressive 53.9 % mAP, despite having more than twice the number of parameters compared to the YOLOv8m model. This indicates that larger models, while more computationally intensive, can yield higher accuracy due to their increased capacity to learn complex features and patterns from the data.

2.4.3. Training Procedure

To train a dataset with large images effectively and achieve high accuracy and efficiency, a powerful computer is essential. Handling large datasets requires substantial computational resources to ensure the training process is both efficient and precise. In our scenario, we used the YOLOv8m.pt model and trained it on an ASUS ROG Strix G15. The detailed specifications of the ASUS ROG Strix G15 can be seen in the figure below.

CPU	
CPU Series	AMD Ryzen 7
CPU Cores	8
CPU Threads	16
Display	
Display Resolution	Full-HD (1920 x 1080 Pixel)
Display Refresh Rate	144 Hz
Display Panel Type	IPS
Display Surface Type	non-glare
Graphics Card	
GPU Model	NVIDIA GeForce RTX 3050
GPU VRAM Capacity	4 GB

Figure 2.9. Asus ROG Strix G15 Specification

- Hardware: A GPU with sufficient VRAM will speed up the training process.
- Batch Size: Start with a moderate batch size (16 or 32), adjust based hardware capabilities and VRAM availability.
- Epochs: Train for a sufficient number of epochs (50-100) but monitor for overfitting.
- Speed: Refers to the time taken for inference, which is the process of making predictions using a trained model.
- CPU: Indicates that the inference is being performed on the Central Processing Unit, as opposed to a GPU (Graphics Processing Unit).

$\Longrightarrow$ To start training the dataset we can follow the command line below (CLI):

```
$ yolo train data=(.yaml) model=yolov8m.pt epochs=100 imgsz=640 ...
    device=0 batch=8
```

Table 2.1. Parameters for Training Procedure

Model	yolov8m.pt
Dataset size	24000
epoches	75
batches	8
Duration	2 days

In **Table 2.1**, the hyper-parameters used in CLI to train our dataset with the YOLOv8m.pt model are presented. These hyper-parameters can be customized depending on the specifications of the computer used to handle the training algorithm in the YOLO model. Factors such as GPU type, memory capacity, and processor speed can influence the selection of appropriate hyper-parameters to optimize training efficiency and model performance.

Note: We used a small batch size of only 8 because our computer's GPU VRAM can handle only 4GB. This limitation requires careful selection of hyper-parameters to ensure efficient training without exceeding the memory capacity of the GPU.

Results after Training the Dataset:

After the training is finished, the terminal will display detailed usage information, indicating the data and framework used during the training process.

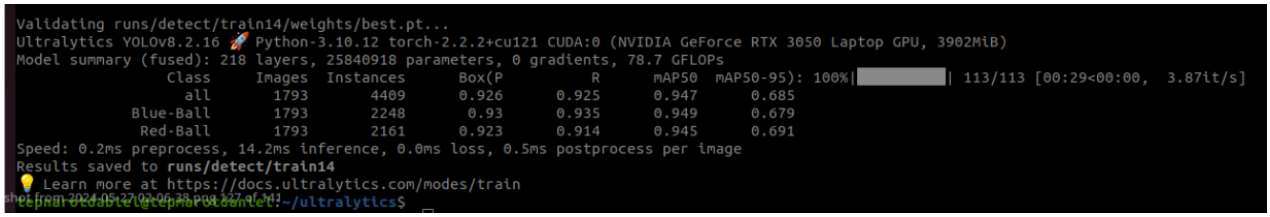

Figure 2.10. Detailed usage information in training process

As showed in **Figure 2.10**. We used Ultralytics packages YOLOv8.2.16, python3.10.12 torch-2.2.2+cu121 CUDA:0 (Nvidia Geforce RTX 3050, VRAM = 3902MB)

Table 2.2. Training Results

Class	Precision	Recall	mAP	mAP 50 -90
All	0.926	0.925	0.947	0.685
Blue-Ball	0.93	0.935	0.949	0.679
Red-Ball	0.923	0.914	0.945	0.691

In this **Table 2.2** describe the model performance across all metrics with high precision, recall, and mAP values. This indicates that the model is generally effective at detecting both blue and red balls.

- Precision: Indicating the model's ability to identify objects accurately.
  - The precision for blue balls (0.93) is slightly higher than for red balls (0.923). This means the model has a slightly lower false positive rate for blue balls.
  - For red ball is also slightly lower false positive rate with all class 0.926.
- Recall: Representing the model's ability to detect all objects.
  - The recall for both of ball (0.925) are indicated the model is better at identifying.
- Mean Average Precision (mAP): Perform measure that combines accuracy, precision, and recall to evaluate the model's object detection capabilities across multiple frames.
  - The mAP values are very close for both blue balls (0.949) and red balls (0.945), suggesting that the model has a balanced performance for both classes in terms of average precision across different recall levels.
- mAP50-95: Check accuracy at IoU levels from 0.5 to 0.95 (High mean better). Interestingly, the mAP 50-90 for red balls (0.691) is higher than for blue balls (0.679). This could indicate that the model performs slightly better at higher IoU thresholds for red balls compared to blue balls.

3. HARDWARE IMPLEMENTATION

After the dataset training is finished, the terminal will display detailed usage information, indicating the data and framework used during the training process. In this project, we implemented the trained model on the Jetson AGX Xavier using the RealSense D435i camera. The Jetson AGX Xavier, known for its powerful AI capabilities, is ideal for deploying deep learning models in real-time applications.

NVIDIA JETSON AGX XAVIER
TECHNICAL SPECIFICATIONS

GPU	512-Core Volta GPU with Tensor Cores
DL Accelerator	(2x) NVDLA Engines
CPU	8-Core ARM v8.2 64-Bit CPU, 8 MB L2 + 4 MB L3
Memory	16 GB 256-Bit LPDDR4x | 137 GB/s
Display	HDMI 2.0, eDP1.4, DP HBR3
Storage	32 GB eMMC 5.1
Vision Accelerator	7-Way VLIW Vision Processor
Encoder/Decoder	(2x) 4Kp60 | HEVC/(2x) 8Kp30 | 12-Bit Support
CSI	(16x) CSI-2 Lanes
PCIE/SLVS/USB/UFS	(8x) PCIe Gen4 / (8x) SLVS-EC (3x) USB 3.1 Single Lane UFS
Other	UART, SPI, CAN, I²C, I²S, DMIC, GPIOs
Connectivity	Gigabit Ethernet
Power	10 W–30 W
Size	87 mm x 100 mm
Mechanical	699 pin Molex Mirror Mex Connector Integrated Thermal Transfer Plate

Figure 3.1. Jetson AGX Xavier Specification

As in **Figure 3.1** describe the technical specifications of Jetson AGX Xavier which is the powerful AI computing device developed by NVIDIA, specifically designed for high performance AI computing with high GPU and CPU 8-core built-in. It's small and tiny easy to setup, especially for robotic and AI application.

Setup Jetson through Jetpack:

In this part will provide the step for Jetson setting up through Jetpack, since Jetson AGX Xavier can be support with ubuntu version 20.04, so we need to setup with computer that handle on Ubuntu 20.04 to flash image jetson through jetpack:

- Download the NVIDIA JetPack SDK from the NVIDIA website.
- We setup with Jetpack 5.1.2 on Ubuntu operating system 20.04.

- Jetpack 5.1.2 buit-in Libraries:
  - CUDA: 11.4.315
  - TensorRT: 8.5.2.2
  - CuDNN: 8.6.0.166
  - OpenCV: 4.5.4
- Connect the Jetson AGX Xavier to your host computer via a USB cable.
- Open the NVIDIA SDK Manager on your host computer and follow the instructions.

Setup Realsense Camera:

In this section, we will use the RealSense D435i camera to perform object detection with the YOLOv8 model. To enable the RealSense camera to work with the Jetson AGX Xavier, we need to follow these steps:

- Install RealSense SDK and test it with command:

  ```
  $ realsense-viewer
  ```

- Install YOLOv8 Model through Ultralytics github and follow the give step in the github

Autonomous Robot MR2:

For **Figure 3.2** is an overview of the hardware setup for MR2 to catch blue or red balls and deliver them to silos.

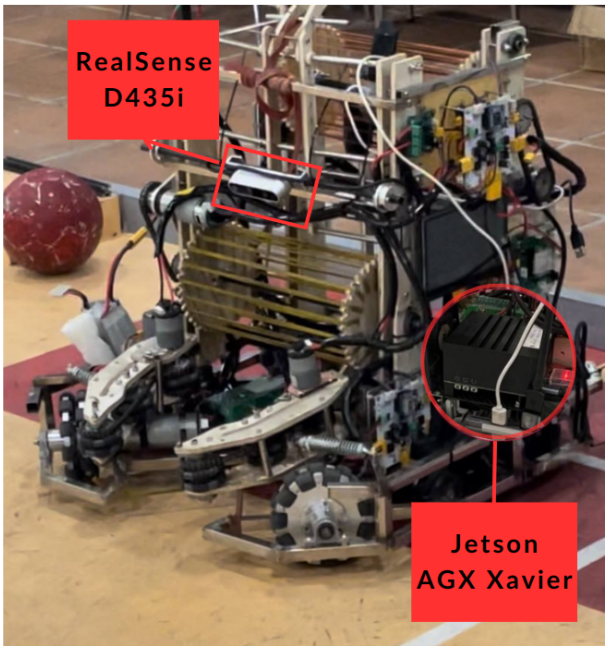

Figure 3.2. MR2 (Credit: ITC01 H101)

4. RESULT AND DISCUSSION

4.1. Results

After, setup the yolo model on the hardware we get the result from the both frames of Realsense D435i and Webcam on the Robot MR2:

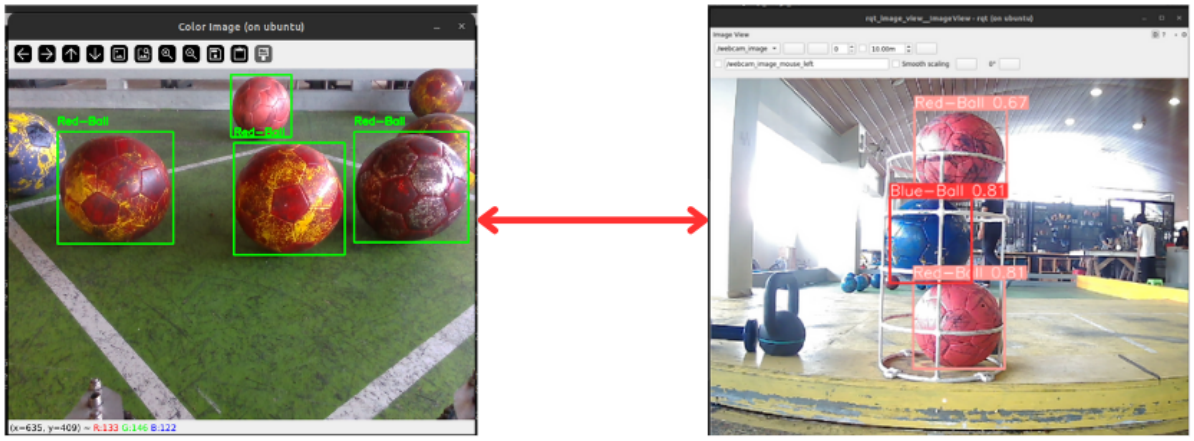

Figure 4.1. Ball Detection from realsense D435I frame and Webcam

As shown in **Figure 4.1**, the output results for both red and blue balls are clearly recognized, with an acceptable confidence interval ranging from 0.70 to 0.80. This performance is consistent even under varying conditions such as different lighting, darkness, brightness, and cluttered backgrounds.

4.2. Discussion

In this part, we will discuss three main aspects as shown in **Figure 4.2** of our system's performance: inference time, GPU performance, and CPU performance.

- Inference Time: refers to the duration it takes for the model to process an input image and produce an output. As shown in **Figure 4.2.a** the Model take 21.8ms of inference time. It means he system can provide quick and accurate detections, which is essential for the dynamic and fast-paced environment of a Robocon game.
- GPU Performance: In **Figure 4.2.b** the GPU operating with 79% on temperature $40.00°C$ during object detection, Optimizing GPU performance further enhances the system's ability to handle high-resolution images and complex scenes with multiple objects. The robust GPU performance ensures that the object detection model maintains high accuracy and responsiveness.

- CPU Performance: In **FIgure 4.2.c** with the CPU operating at 38.6% during object detection, there's enough capacity to use ROS2 for creating a controller node to manage the robot's motion. The spare processing power allows for real-time control and decision-making within ROS2, ensuring smooth and responsive robot motion based on object detection. This setup enhances task execution in dynamic environments such as Robocon competitions.

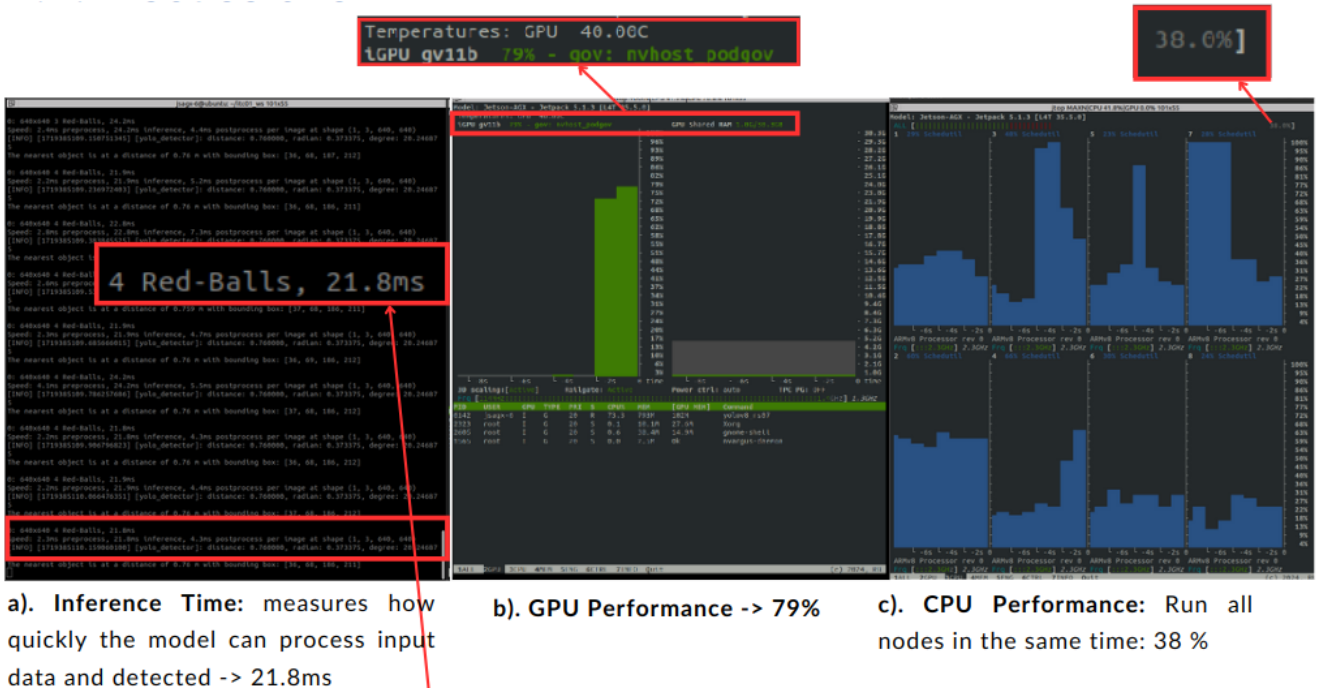

Figure 4.2. Hardware Output during YOLO model process images

5. CONCLUSION

The purpose of this project is to enable the robot to automatically catch and deliver balls to silos within a maximum of 3 minutes for a game competition. Therefore, base on 4 main points:

- The GPU utilization ranges between 79-90%, indicating that the model leverages the GPU efficiently, which is expected for a deep learning application like YOLOv8.
- The CPU usage at 38% indicates there is enough capacity for other tasks, ensuring the system can handle additional processes without overloading.
- The YOLOv8m.pt model demonstrates accuracy and efficiency with an inference time of 21.8ms.
- Optimizing inference time ensures that the system can provide quick and accurate detections, which is essential for the dynamic and fast-paced environment of a Robocon game. In our setup, we observed an average inference time that meets the real-time requirements, allowing the MR2 to respond promptly to the detected balls.

As the result, our model operates efficiently on the robot and does not cause any slowdowns, making it suitable for the competition.

REFERENCES

[1] Panja, E., Hendry, H., & Dewi, C. (2024). YOLOv8 Analysis for Vehicle Classification Under Various Image Conditions. *Scientific Journal of Informatics*, 11(1), 127–138.

[2] Xiao, X., & Feng, X. (2023). Multi-Object pedestrian tracking using improved YOLOV8 and OC-SORT. Sensors, 23(20), 8439.

[3] Terven, J., Córdova-Esparza, D., & Romero-González, J. (2023). A comprehensive review of YOLO architectures in computer vision: from YOLOV1 to YOLOV8 and YOLO-NAS. Machine Learning and Knowledge Extraction, 5(4), 1680–1716.

[4] Lou, H., Duan, X., Guo, J., Liu, H., Gu, J., Bi, L., & Chen, H. (2023). DC-YOLOV8: Small-Size Object Detection Algorithm based on Camera Sensor. Electronics, 12(10), 2323.

[5] rLei, R., Wu, Y., Ren, Y., & Gu, S. (2024). An improved YOLOV8 target detection algorithm and its application in tennis ball picking robot. In Lecture notes in computer science (pp. 186–197).

[6] Zhou, X., Koltun, V., & Krähenbühl, P. (2020). Tracking objects as points. In Lecture notes in computer science (pp. 474–490).

[7] Mondragon, I. F., Campoy, P., Olivares-Mendez, M. A., & Martinez, C. (2011). 3D object following based on visual information for Unmanned Aerial Vehicles. Yolo, 1–9