Cairo University
Faculty of Engineering

Credit Hours system
Mechanical Department

ADVANCED ROBOTICS AND ARTIFICIAL INTELLIGENCE (MDPS453)

TRASH-BOT

Mina Amir Girgis
1200835
Group: 9

Submitted to: Dr. Hossam H. Ammar

Submission date: **21/1/2025**

Table of contents

Table of figures

Table of scripts

BagToNPY.py: Used to convert rosbag files to numpy arrays.

camCalibration.py: Used to calibrate the camera.

modelTraining.py: Used for manually navigating the robot.

ROS_Bag_to_AI_Reg.py: Used for normalizing and visualizing the data-sets

AI_Reg_2_2: Used to train the ML models.

rosBagMerge.py: Used for merging rosbag files.

RosCoreNode.py: The node responsible for operating the bot using ML models.

YOLONode.py: Used to identify the closest objects to the robot.

split_dataset.py: Used to split the yolo data-set into training, validation and testing parts.

convert_coco_to_yolo.py: Used to convert COCO labelling format to YOLO format.

Dataset.yaml.py: Contains the YOLO model configuration.

main.cpp: contains the esp32 code.

I- Abstract

Up to this point in the project, we managed to operate the turtle-bot through obstacles using a machine learning model and Aruco markers. In this challenge, we aimed at improving the model accuracy and substituting the Aruco markers with yolo model. The project purpose is to quickly detect nearby trash items and reaching these items while navigating through obstacles. When the bot reaches an item, it stops in front of it and turn on a buzzer till the object is removed and the robot continues.

II- Introduction

The implementation steps for the machine learning models are the same as the last challenge, except for one thing. In the last challenge, after training the bot manually, recording rosbags and collecting data, we trained these data to reach a specific goal while avoiding obstacles. This technique showed a low accuracy and a high error rate. So, we decided to train two models. One acts as a go-to-goal code that takes the coordinates of the specific location and go to it. The second acts as a fuzzy logic that takes the coordinates of an obstacles and avoid it. The separate models showed to be more reliable and easier to train.

The selected algorithm in this report is Artificial Neural Network which is a machine learning algorithm inspired by the structure and function of the human brain. It is a powerful tool for modeling and solving complex problems, especially those involving non-linear relationships and large amounts of data. ANNs are widely used in applications such as image recognition, natural language processing, and predictive modeling. We will provide some basic information about ANN in the next section.

III- Proposed Algorithm

As mentioned before, the selected algorithm in this report is Artificial Neural Network, which is a machine learning algorithm that consists of several layers of interconnected nodes (neurons), where each node mimics a biological neuron. These layers are typically: input layer which receives the raw data, hidden layer which perform some computations, and output layer which produces the final output.

1- ANN training process

A- Initializing weights and biases

ANN initializing is done through activation functions. Activation functions introduce non-linearities into the model, enabling it to learn complex patterns. Common activation functions include:

- **ReLU (Rectified Linear Unit):** f(x) = max(0, x)
- **Sigmoid:** f(x) = 1 / (1 + e^(-x))
- **Tanh (Hyperbolic Tangent):** f(x) = (e^x - e^(-x)) / (e^x + e^(-x))

B- Forward propagation

In forward propagation, data flows from input layer to the output layer. Each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function:

$$z = \Sigma (w * x) + b$$

$$a = \text{ActivationFunction}(z)$$

- w: weights
- x: inputs
- b: bias
- a: activated output

C- Compute loss

The loss function quantifies the difference between the predicted output and the actual target. Common loss functions include:

- Mean Squared Error (MSE) for regression tasks.
- Cross-Entropy Loss for classification tasks.

D- Back propagation

ANN also applies back propagation which adjusts the weights and biases to minimize the loss function (Mean square error for regression tasks) and then computes a gradient and then uses an optimization algorithm such as Gradient Descent to update the weights:

w_new = w_old - learning_rate * gradient

E- Optimizing

Optimization algorithms improve model performance by minimizing the loss function. Common ones include:

- **Stochastic Gradient Descent (SGD):** Updates weights using small batches of data.
- **Adam**: Combines momentum and adaptive learning rates for faster convergence.

IV- Implementation

1- ESP Node

The code uploaded on the esp32 has only one purpose which is to operate the robot using the velocities sent by the ROS Core node. The code does that by converting linear and angular velocities to rpm values of the two wheels and then map these values through a previously calibrated references to output a PWM values that operates the motors accordingly. The code also publishes the encoder values after converting it to distances in the two directions.

The new addition in this challenge is that we added a buzzer and an IR sensor to the esp. the purpose of the buzzer is to indicate the existence of a trash item in front of the robot through intermittent peeps. The faster the sound, the closer the object to the robot. The purpose of the IR sensor is to detect walls in front of the robot and quickly divert its path to avoid hitting them. This process was necessary since any walls will appear as flat surfaces on the camera and won’t be detected.

2- ROS-Core Node

As explained above, the purpose of the Core node is to successfully navigate the robot through obstacles and stops in front of the closer trash item. To do that, the node subscribes to the camera node which gives the distance, angle and class of the nearest object detected by the camera. Then the code decides whether this object is an obstacle or a goal based on the class number sent by the camera. If it is an obstacle, the node uses the obstacles avoidance ML model till the robot navigate successfully through it. And once it detects a goal, it uses the ML model for go-to-goal which navigates the robot towards this goal and finally stopping in front of it.

The implementation steps for the two models are basically the same and quite similar to the steps used in the last challenge. These steps are as following:

A- Manual Trials

The first step in this challenge implementation is collecting data from the robot by manually navigating it and recording these the data published and subscribed in these trials. We do this by running two scripts: **YOLONode.py** and **modelTraining.py**.

The first script is using the mobile camera to detect the nearby objects and calculating the distance and angel to the closest one, then publishing in “trash” topic. The second script is the tele-operation script that navigates the robot using the computer keyboard. This script also subscribes to the “Encoder” topic to calculate the error between the robot position and the desired object, then publishes these data in “error” topic. The script also publishes the linear and angular velocities in “cmd_vel” topic based on the user navigation.

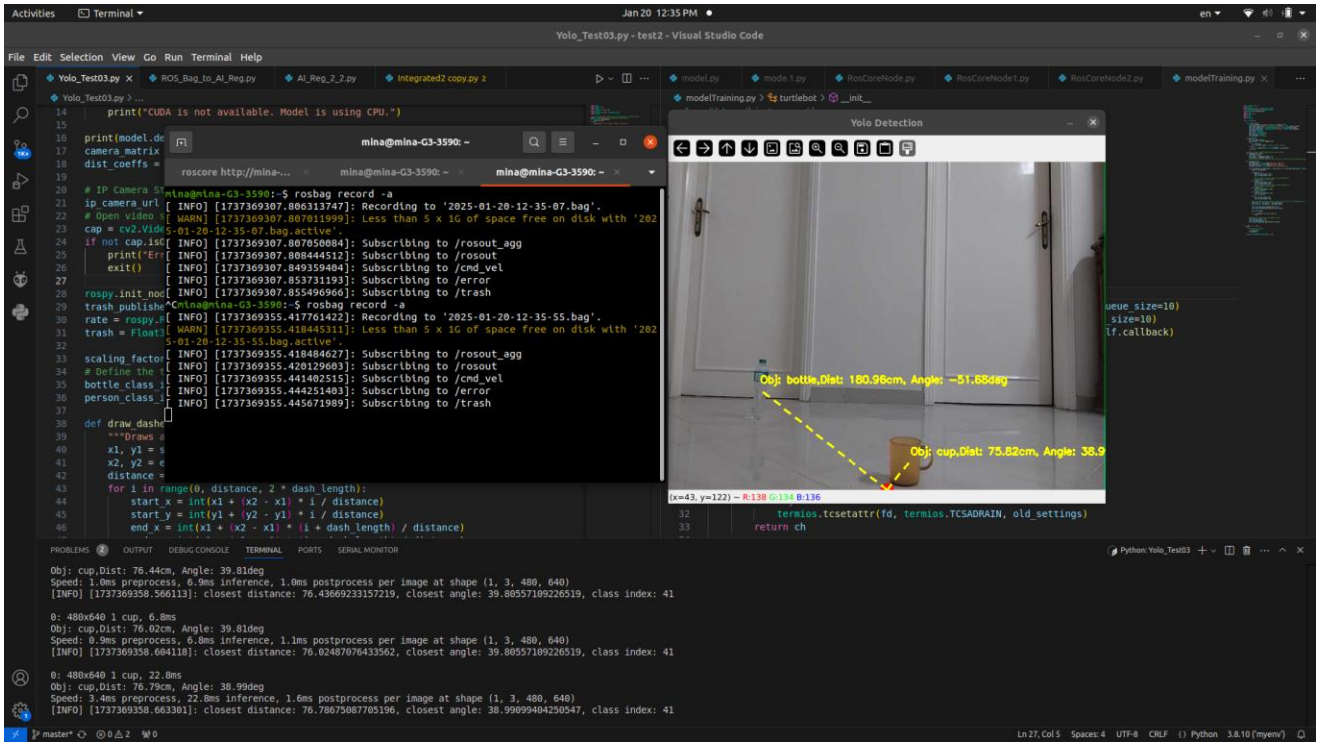


Fig (1)

Before running these two scripts and manually navigating the robot through obstacles, we started recording the published and subscribed topics through the rosbag command: “rosbag record -a”. This command records the messages of all published and subscribed topics and while no one actually subscribe to “error” topic, it is still recorded in the rosbag file.

B- Merging

After running several trials and collecting a number of data-sets, we needed to merge these recorded data to obtain one large data-set that the model will be trained on. In order to merge these files, we used the script **rosBagMerge.py**, which basically loops through all the recorded bag files while writing the data in them in a new merged bag file.

C- Converting

After collecting a unified bag file that carry all the data-sets, we needed to convert it to NumPy arrays in order to train the model and we also needed to make the number of samples (publishes or subscribes) of each topic equal. That’s not normally the case, so, we used the script **BagToNPY.py** to loop

through the desired topics in the bag file and appending them in an array, making sure their number of samples are equivalent.

D- Normalizing & Visualizing

In order to normalize and visualize the collected data-set, we used the script **ROS_Bag_to_AI_Reg.py**. Normalizing the data means that we make the wights of the topics consistent by setting a common range with a fixed minimum and maximum value (0 and 1). This step is important because the recorded topics have different data ranges. e.g., the cmd_vel.linear.x has a range of -0.8 to 0.8 m/s while the cmd_vel.angular.z has a range of 0.3 to -0.3 rad/s.

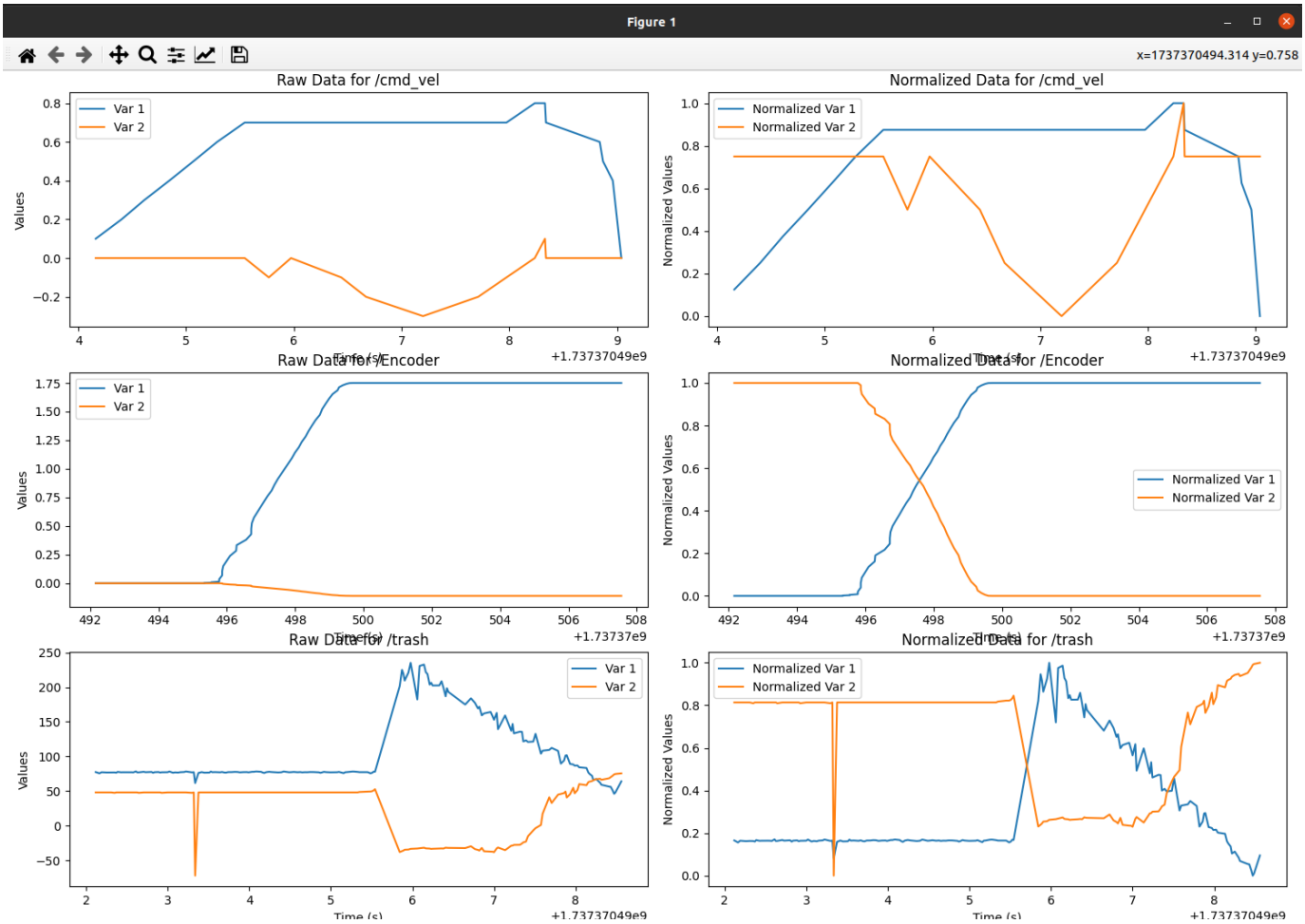


Fig (2)

E- Model Training

Finally, we uploaded the recorded data as NumPy arrays and trained them using different machine learning algorithms by using sklearn library. I used the script **AI_Reg_2_2.py** in order to train the model using Artificial Neural Network algorithm and calculated its mean square error which was about 0.0028 for the go-to-goal model and 0.0013 for the obstacle-avoidance model. Finally, I saved the trained models as pkl files.

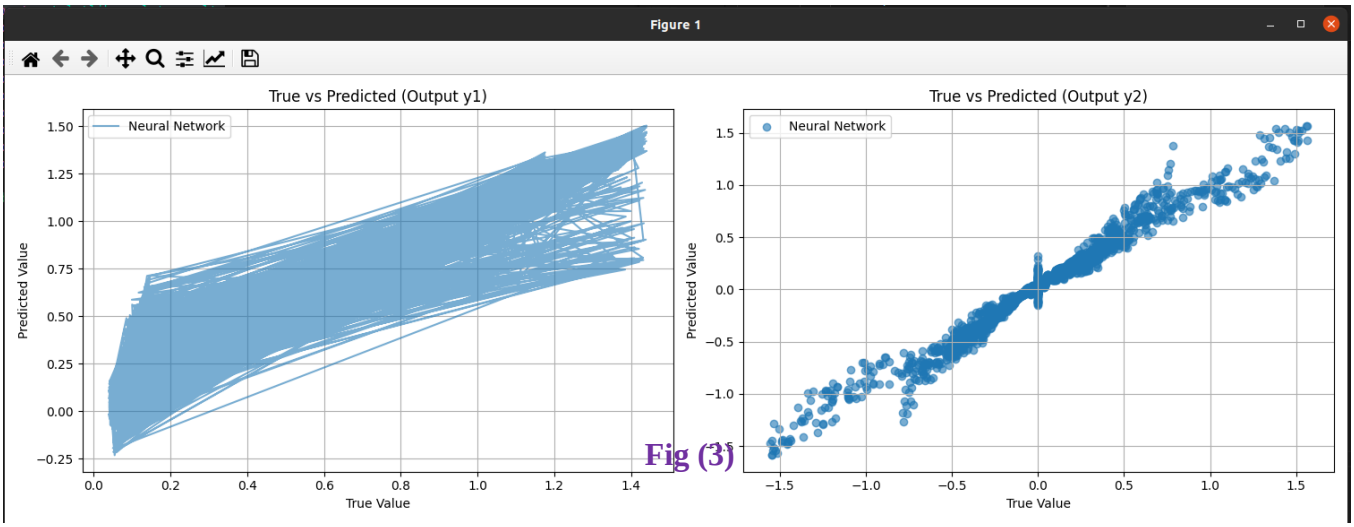


Fig (3)

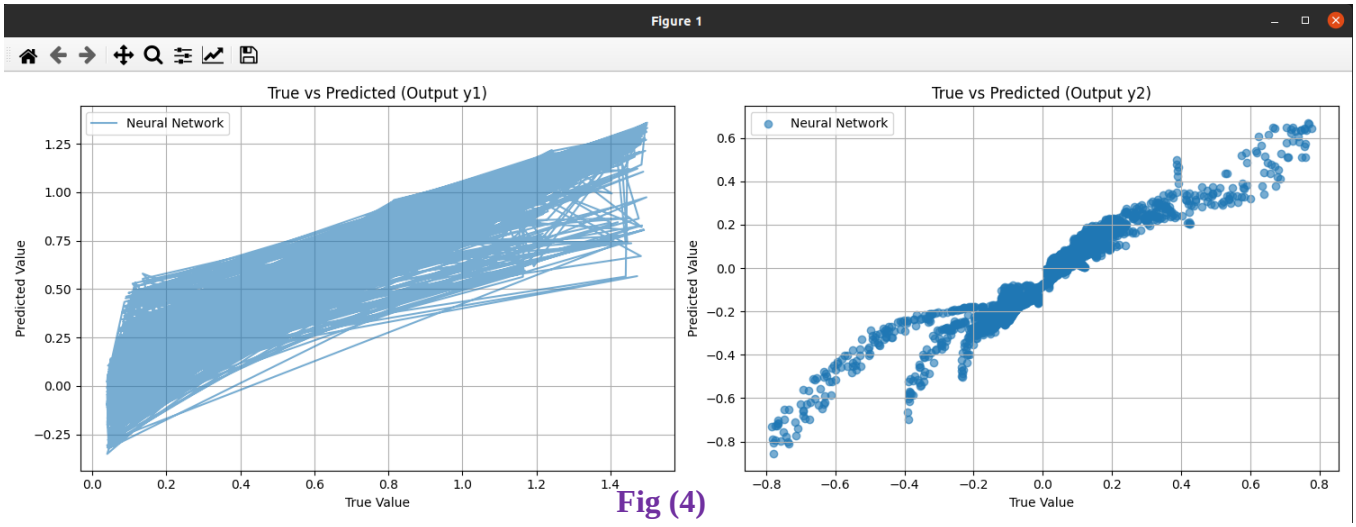


Fig (4)

F- Testing

The final step was to test out the model by fully operating the robot using the trained model. I used the **RosCoreNode.py** and **YOLONode.py** scripts to do so. I ran the model in a number of trials recording the robot behavior and

measuring its real-life accuracy and precision which we will discuss later in the results section.

3- YOLO node

A- Collecting data-set

As mentioned before, the purpose of the project to identify and classify trash items for recycling purposes. So, in order to do that, we downloaded a dataset called Trash Annotations in Context or TACO for short. This open-source data-set contains more than 2500 picture divided into 15 batch, but unfortunately, this data-set uses COCO format for labeling which is not compatible with the YOLO format.

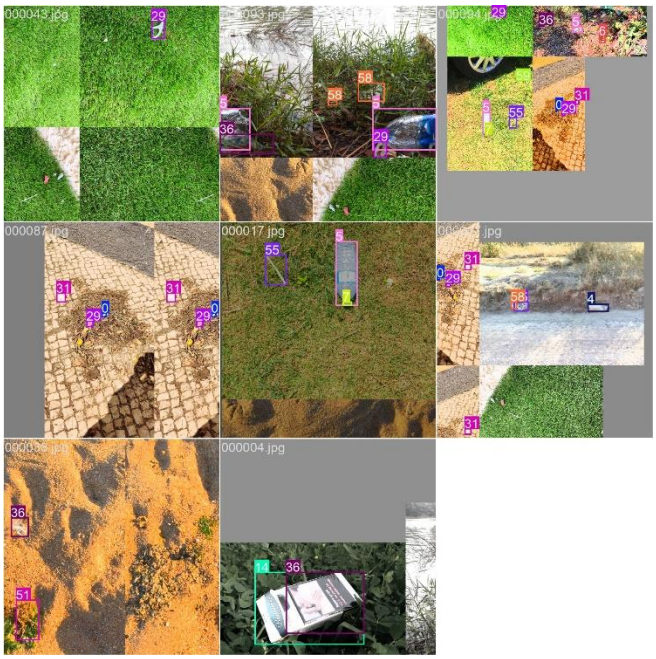


Fig (5)

Fig (6)

B- Converting from COCO to YOLO

In order to convert from COCO to YOLO format, we used the script **convert_coco_to_yolo.py**, which basically extract the images' classes, label boxes coordinates and dimensions from a .json file and separate them into .txt files for each image containing its information.

C- Dividing the data-set

After converting the data-set to a compatible format, we divided it into three categories: train, validate and test, which will be later used by YOLO to train the model.

D- Configuring the model

A .yaml file was created at the root of the directory. This file contains necessary information for the training that will be used by YOLO to create the model. This information are the paths to the training, validation and testing data-sets. Also, it contains the number of classes and their names.

E- Model Training

After everything is ready, we used the command: “yolo detect train data=dataset.yaml model=yolov5s.pt epochs=50 batch=8 imgsz=640 workers=4 cache=False overlap_mask=False” in order to set the training parameters and the path to the .yaml file. We used yolov5s for training because it provides a balance between performance and accuracy.

F- Training Results

After about an hour, the training finished creating a directory containing the training results. These results include training statistics such as confusion matrix and labels correlogram. The results also contained labeled picture examples from the training and validation batches. Finally, the directory contained two models: best.pt and last.pt. That is because yolo performs iteration while training and modify its parameters between them. We used the best model and upload it to the script Yolo_Test03.py in order to use it for object identification and classification and finally publishing the closest object distance, angle and class to “trash” topic

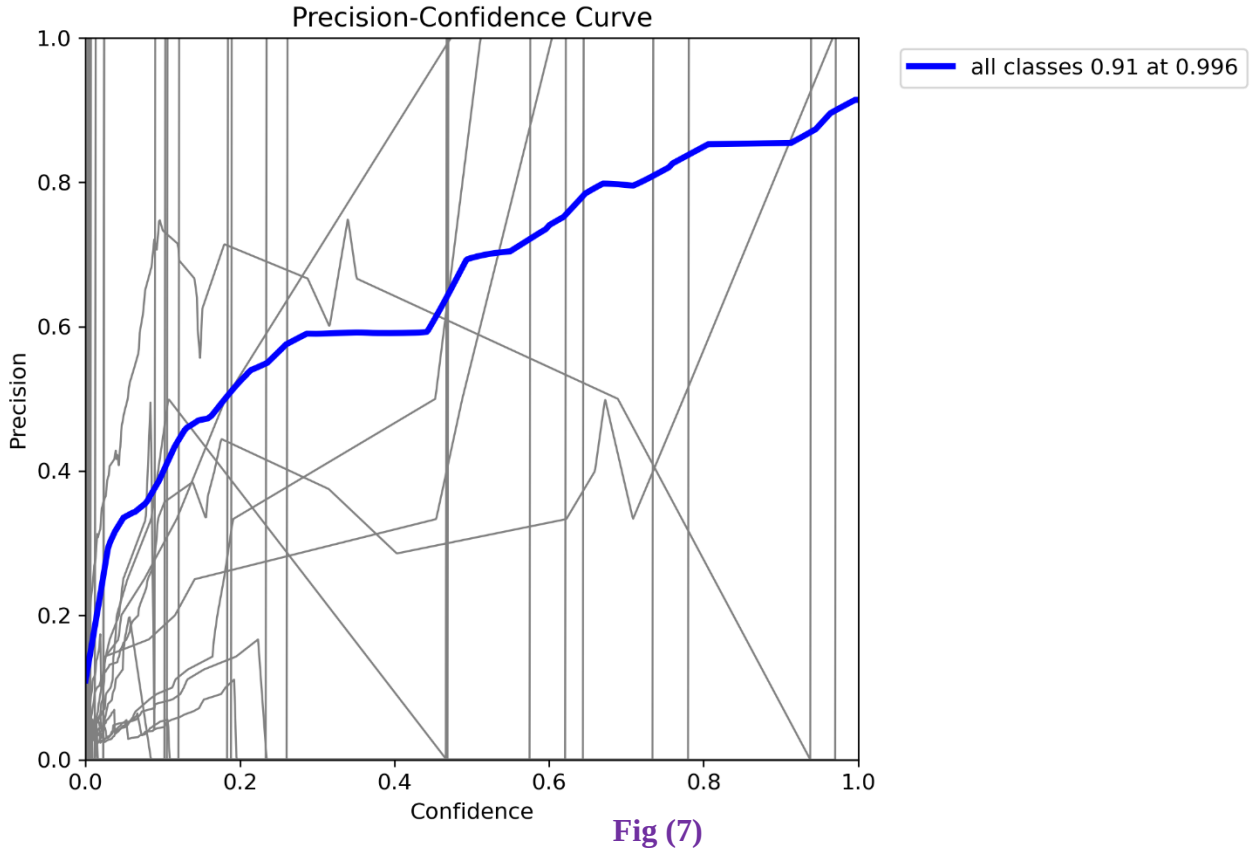


Fig (7)

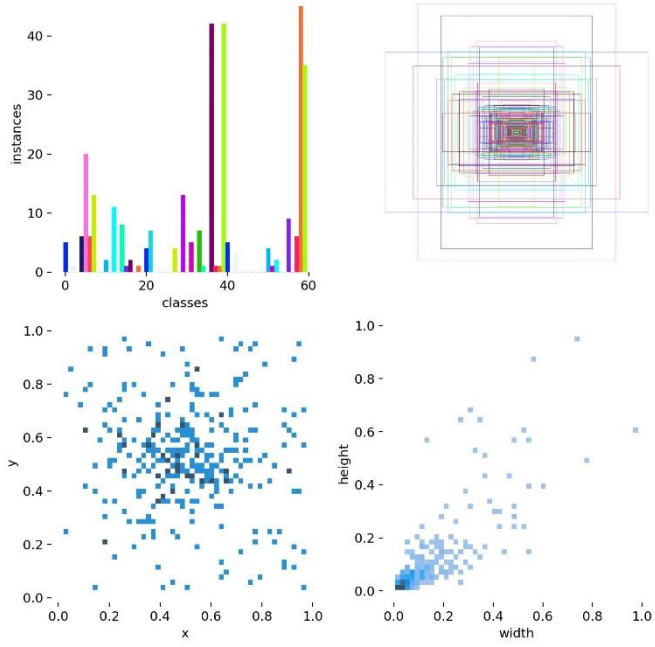


Fig (8)

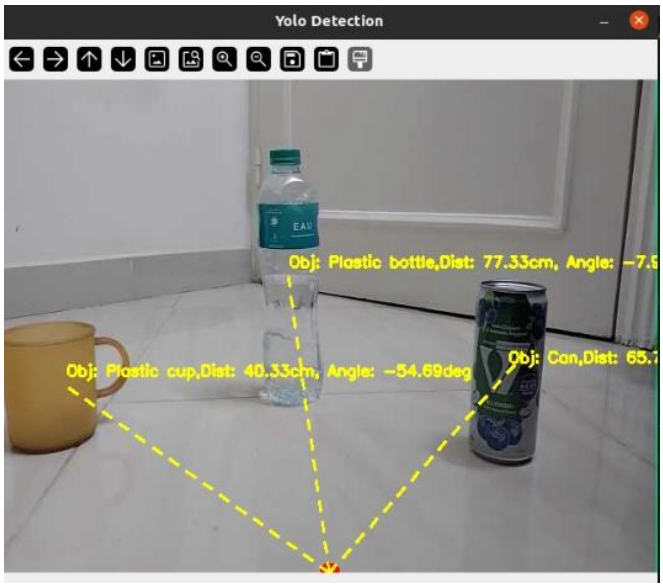


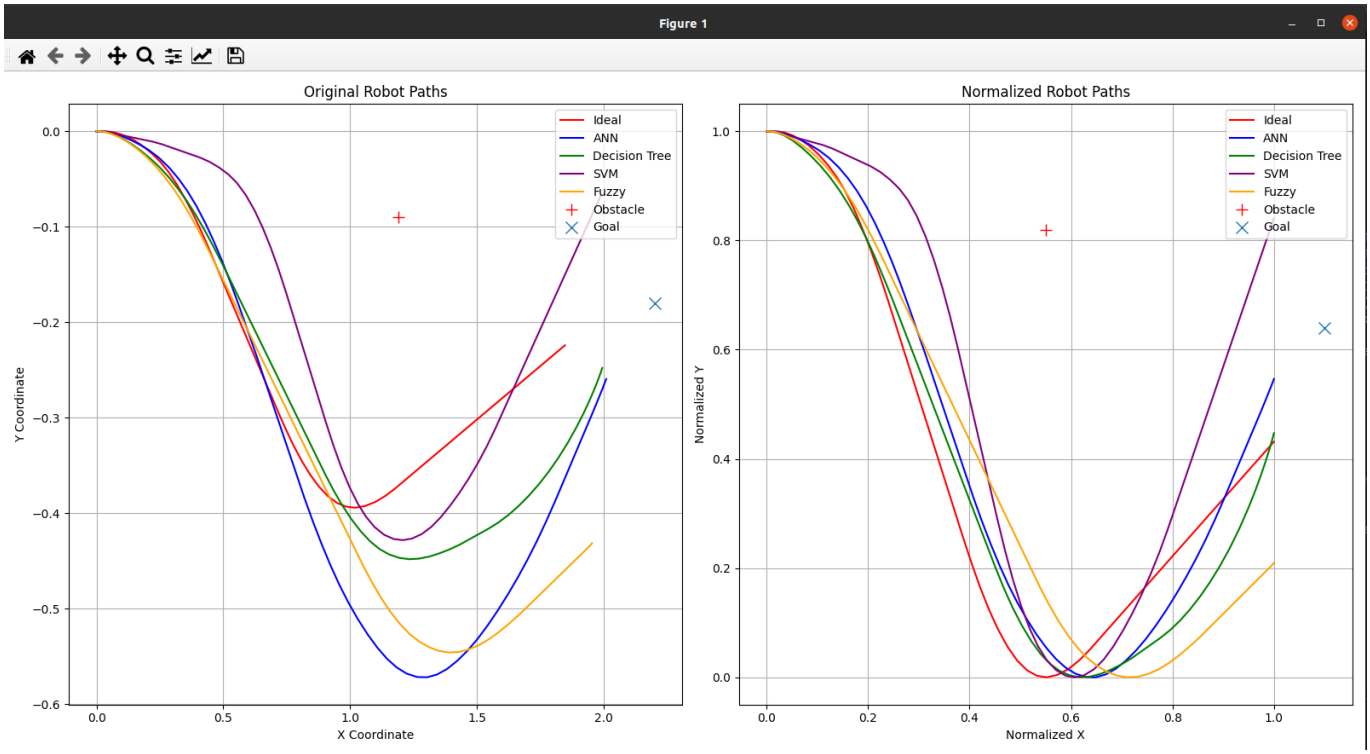
Fig (9)

V- Results

In order to evaluate the performance and accuracy of each proposed algorithm. We calculated the theoretical mean square error and then tested each model separately. The testing parameters such as obstacle and goal positions were identical in all tests, that is to ensure an unbiased result. We also tested the fuzzy logic on these parameters. Finally, we navigated carefully one more time using manual tele-operation to make a reference or control group. We compared the models and the fuzzy logic to the manual data according to the maximum turn they made and time of arrival. The actual and normalized paths are shown in the next figures.

	MSE	Maximum turn (m)	Arrival Time (s)
Fuzzy		0.36	6.2
SVM	0.00315	0.36	5.37
Decision tree	0.0005	0.265	4.84
ANN	0.00205	0.39	4.6

Fig (10)



VI- Conclusion

In contrast, there is no machine learning model without error or without the need of further training. That's because the accuracy and precision of a model not only depends on its structure and mathematical formulas but also on the size and accuracy of the collected data, and the more we train the model the more accurate it will become. In the end, we hope to create our own data-set by taking pictures of various items and train the model on them. That should increase the model accuracy because the data-set will be locally collected rather than gathered from foreign sources.