Vehicle Detection and Distance Estimation

Hassan Ramadan[1], Ahmed AbdElkader[2], Ahmed Khaled[3], Ahmed AbdElalem[4], Weam Mohamed[5], Hend Tawfik[6]

Faculty of computer and information, Zagazig University, Zagazig, 44519, Egypt
Emails: RmdanJrx@gmail.com; ahmedabdalkaderma@gmail.com; ahmmkhaled20@gmail.com; ahmed.abdelalem61@gmail.com; weaammohamed321@gmail.com; hendmohammed2809@gmail.com
Correspondence: RmdanJrx@gmail.com

Abstract

We present a scheme of how we can help drivers and provide a way to reduce car accidents. How Yolo can be improved to detect objects on thermal videos and how object detection results are used to predict the absolute distance between objects and the source camera.

1- Introduction

1.1 Objectives

Our project mainly is about object detection and distance estimation of vehicles using thermal imaging. We have seen a lot of car accidents happen a lot, especially on the roads with darkness or foggy weather.

We played the scene of the scenario. We have reached a satisfying result to solve this problem using thermal imaging and ML/DL technologies.

We have solved this problem by providing the driver with a video with all objects detected and info written and proved the accuracy of distance estimation on thermal imaging.

Figure 1. Example of End Result.

1.2 Summary of Project Timeline

- Phase A: Specifications & Setup
  - Idea, competitive analysis, and, learning - are all about the idea evaluation, analysis, initial architecture, and technologies used.
  - At that phase, we did a lot of searches and analyses to determine the project outline and take the initial steps to build the model, so it's all about choosing the optimal solution that helps in the point of the darkness and the foggy weather till evaluating the result of the competitive analysis depending on a specific metric.
- Phase B: Design
  - At this phase, we had built initial abstract architecture to build an intuition about how the stuff is going, starting from processing inputs, calculating info needed to visualizing our results.
- Phase C: Development
  - At this phase, we developed our custom object detector which detects vehicles on images or videos captured by a thermal camera, and distance estimation model which calculates the absolute distance between vehicles and the source camera using object detection results.
  - Using thermal imaging we can have a clear video with all objects obvious regardless of the weather and visibility which is an extremely high advantage compared to normal imaging.
  - The object detection model helps us detect vehicles & other objects and identify their type (e.g. car, bicycle, person, and so on) then determine the correct coordinates of these objects at each video frame.
  - Then we use object detection results and with the help of the distance estimation model to determine the distance between the source camera and objects.
  - At the end of this phase, we built a visualizer to write all this information back to the video frames so that the user can take a specific action based on the provided information.

2- Available Solutions

2.1 Detection and Distance Estimation using Normal Imaging

- There is already a model that uses normal imaging to detect objects and estimate distance but in the presence of clouds and blurred vision here is the problem.
- This solution is based mainly on DL to detect moving vehicles in video streams and apply a distance estimation method to provide this info to the user.

2.2 Arduino Ultrasonic Sensor

- Estimates the distance only. But it can't detect objects whose distance is more than 20m. And can't detect objects which move at high speed.

Summary

Solution	Pros	Cons
Detection & Estimation using Normal Imaging.	Estimates distance.	Doesn't work in unclear vision.
Arduino Ultrasonic Sensor	Works well regardless of vision. Measures distance.	Can't detect objects which distance farther than 70 feet and move at high speed.

Figure 2. Available Solutions Pros and Cons Summary.

3- Our Proposed Solution

We provided a solution by using thermal imaging to estimate the distance of objects in bad weather and darkness.

We used the distance between detected objects and the vehicle which holds the thermal camera to provide some information to the driver and to take some action which in turn will reduce car accidents.

We collected a dataset of thermal images in foggy weather or unclear vision and retrain or build a new deep learning architecture based on the collected dataset to detect objects and estimate their distance.

4- Development

Before explaining every part in detail & what we have done. Let's take a look at the high-level conceptual arch of our system.

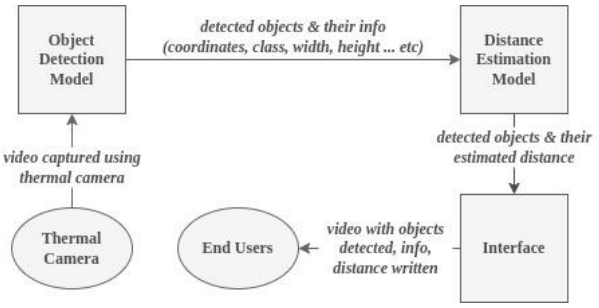

Figure 3. Abstract Architecture.

We developed a deep learning model to help us detect objects in thermal images/videos, identify their types/classes (e.g. car, person), and determine the coordinates of their locations.

Then we format detection results to match the distance estimation model and then use this formatted information afterward to estimate the object's distance.

Then we write this info back to the different video frames.

Subsequently, this process can be divided into 4 main parts:

- Object Detection
- Intermediate Results Calculation
- Distance Estimation
- Visualize Final Results

1) Object detection

1.1 What is YOLO and why?

YOLO, an acronym for 'You only look once, is an object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. It uses neural networks to provide real-time object detection so it's popular because of its speed and accuracy.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects. In addition to High accuracy, it is a predictive technique that provides accurate results with minimal background errors.

This technique provides improved detection results compared to other object detection techniques such as Fast R-CNN and Retina-Net.

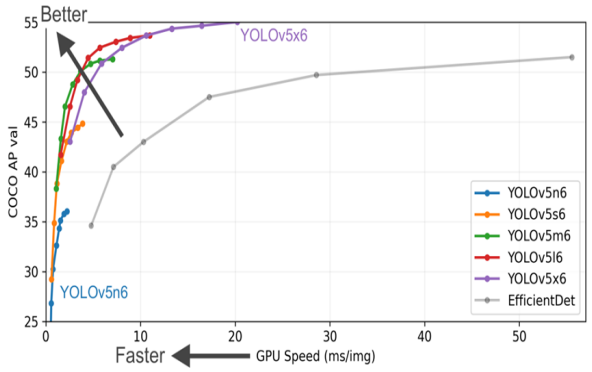

Figure 4. Different YOLOV5 Models.

It's always a trade-off of power vs performance. We trained a custom YOLO 5 object detection on our thermal dataset using pre-trained weights. First, we formatted our dataset to match the YOLO format. Then, we train and test the model using different hyper parameters to get higher accuracy.

1.2 Training YoloV5s

Download the thermal camera Dataset (FLIR Thermal Images Dataset)

The FLIR Thermal Starter Dataset provides an annotated thermal image and non-annotated RGB image set for training and validation of object detection neural networks.

The dataset was acquired via an RGB and thermal camera mounted on a vehicle. The dataset contains a total of 14,452 annotated thermal images with 10,228 images sampled from short videos and 4,224 images from a continuous 144-second video.

All videos were taken on the streets and highways in Santa Barbara, California, the USA from November to May. Videos were taken under generally clear-sky conditions both day and night.

Human annotators labeled and put bounding boxes around 3 categories of objects. The MSCOCO label vector was used for class numbering.

- Category 1: People
- Category 2: Bicycles - bicycles and motorcycles (not consistent with coco)
- Category 3: Cars - personal vehicles and some small commercial vehicles.

For more details about the dataset (**link**).

Prepare dataset to match YOlO5 Format - Convert the Annotations into the YOLO v5 Format

Export labels to YOLO format, with one *.txt file per image (if no objects in the image, no *.txt files are required).

The *.txt file specifications are

- One row per object.
- Each row is class x_center y_center width height format.
- Box coordinates must be normalized by the dimensions of the image (i.e. have values between 0 and 1).
- Class numbers are zero-indexed (start from 0).

Prepare dataset to match YOlO5 Format - Organize Directories

YOLOv5 locates labels automatically for each image by replacing the last instance of /images/ in each image path with /labels/. But first directories have to be in this format.

```
thermal_dataset
├── images
│   ├── train
│   └── val
└── labels
    ├── train
    └── val
```

Figure 5. FLIR Dataset Directory Structure.

Prepare dataset to match YOlO5 Format - Create the dataset YAML file

Here we write a model configuration file for our custom object detector. We chose the smallest, fastest base model of YOLOv5.

The following parameters have to be defined in a data config file

1) Train, Test, and Val: Paths to train, test, and validation sets.
2) NC: Number of classes in the dataset.
3) Names: Names of the classes in the dataset. The index of the classes in this list would be used as an identifier for the class names in the code.

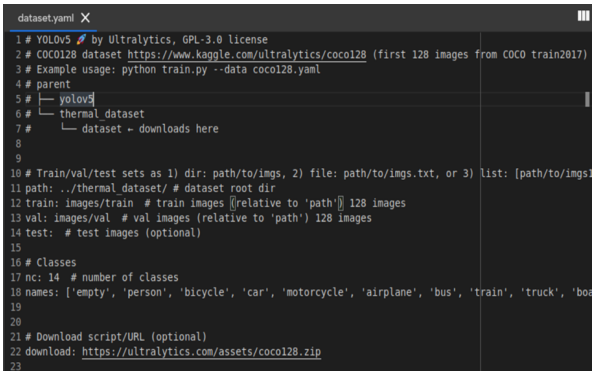

Figure 6. Dataset YAML File Content.

Train a custom YOLOv5 detection model

We chose the smallest, fastest base model of YOLOv5. Starting from pre-trained weight instead of randomly initialized for better performance. Then, export saved YOLOv5 Weights for future use.

We train the model with different epochs (24, 50, and 70) to detect three objects: car, bicycle, and person.

1.3 Evaluate Performance

Now that we have completed training, we can evaluate how well the training process performed by looking at the validation metrics. The training script will drop output logs in “runs”.

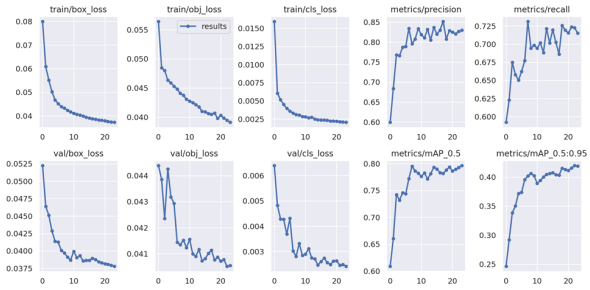

Figure 7. Results of Train, Validation Sets for 24 Epochs.

The more accurate results we get using 24 epochs but in the higher epochs size the model becomes “overfitted” and we can’t optimize it using data augmentation as the bounding boxes will change when the image flipped or rotated.

1.4 Visualize YOLOv5 training data

During training, the YOLOv5 training creates batches of training data. Here is a sample:

Figure 8. Object Detection - Sample

1.5 Export YOLOv5 Weights for Future use

Now our custom YOLOv5 object detector has been verified. We want to take the weights out for use in the coming detection tasks.

2) Intermediate Results Calculation

To estimate the distance to objects (cars, pedestrians, trucks) in the scene on the basis of detection information.

We modified the object detection model output format to output a generated sheet containing detected object coordinates.
The sheet has a row for each frame and a column with a coordinate (xmin, ymin, xmax, ymax).
We will use this sheet in the next step (distance estimation).
We also saved annotated video frames to write distance estimation results to them back later.

frame	xmin	ymin	xmax	ymax
0	231.9046875	159.375	260.04375	174.4791667
0	360.95625	136.4583333	471.571875	231.25
0	536.5828125	169.2708333	568.603125	199.4791667
0	916.9453125	179.6875	1030.471875	207.8125
0	743.259375	168.75	879.103125	221.875
0	1170.196875	190.625	1242	236.4583333
0	564.721875	141.1458333	698.625	227.6041667
0	932.4703125	189.0625	1111.978125	247.3958333
0	790.8046875	187.5	922.7671875	245.3125
1	232.875	159.8958333	260.04375	174.4791667
1	536.5828125	169.2708333	568.603125	198.9583333
1	361.9265625	136.9791667	474.4828125	232.8125
1	917.915625	179.1666667	1031.442188	207.2916667
1	744.2296875	169.7916667	875.221875	221.875
1	1170.196875	190.625	1242	236.4583333
1	564.721875	141.1458333	698.625	227.6041667
1	932.4703125	189.0625	1112.948438	247.3958333
1	790.8046875	187.5	922.7671875	245.3125
2	530.7609375	168.75	576.365625	201.0416667

Figure 9. Object Coordinates Sheet

Edit detect.py in Yolov5 to extract the coordinate form (x_center,y_center, width, height) like that:

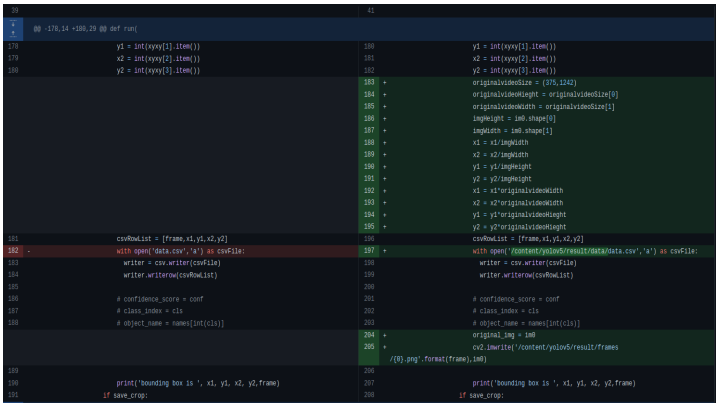

Figure 10. detect.py Modifications

3) Distance Estimation

Train a deep learning model that takes in bounding box coordinates of the detected object and estimates the distance to the object.

3.1 Training

1. Download KITTI Dataset

Kitti contains a suite of vision tasks built using an autonomous driving platform. The full benchmark contains many tasks such as stereo, optical flow, visual odometry, etc. This dataset contains the object detection dataset, including the monocular images and bounding boxes.

Figure 11. Image Sample - Training Set

2. Format Dataset to match this directory structure

KITTI Dataset

|-- test_images

|-- train_annots

`-- train_images

3. Train Model

We could train the model from scratch by basically running the train.py and passing train and test sets to it. Or continue training a pre-trained model by running training_continuer.py and passing the pre-trained model, training weights, train, and test sets.

3.2 Predicting

We use bounding box coordinates (xmin, ymin, xmax, ymax) to predict distance (z).

frame	xmin	ymin	xmax	ymax	scaled_xmin	scaled_ymin	scaled_xmax	scaled_ymax	distance
0	842	315	912	362	817.003125	164.0625	884.925	188.5416667	37
0	36	327	59	347	34.93125	170.3125	57.2484375	180.7291667	42
0	952	344	1066	396	923.7375	179.1666667	1034.353125	206.25	17
0	553	325	584	385	536.5828125	169.2708333	566.6625	200.5208333	32
0	789	335	873	413	746.1703125	174.4791667	847.0828125	215.1041667	12
0	1204	365	1280	452	1168.25625	190.1041667	1242	235.4166667	7
0	585	268	717	439	567.6328125	139.5833333	695.7140625	228.6458333	11
0	962	363	1145	476	933.440625	189.0625	1111.007813	247.9166667	6
0	810	360	954	470	785.953125	187.5	925.678125	244.7916667	6
1	842	314	911	361	817.003125	163.5416667	883.9546875	188.0208333	38
1	36	327	59	347	34.93125	170.3125	57.2484375	180.7291667	42
1	953	343	1067	396	924.7078125	178.6458333	1035.323438	206.25	17
1	552	324	585	384	535.6125	168.75	567.6328125	200	33
1	789	335	874	413	746.1703125	174.4791667	848.053125	215.1041667	12
1	1204	364	1280	452	1168.25625	189.5833333	1242	235.4166667	7
1	585	268	717	439	567.6328125	139.5833333	695.7140625	228.6458333	11
1	810	359	954	471	785.953125	186.8791667	925.678125	245.3125	6
1	962	364	1145	476	933.440625	189.5833333	1111.007813	247.9166667	6
2	779	316	831	363	755.8734375	164.5833333	806.3296875	189.0625	41
2	1041	275	1066	301	1010.095313	143.2291667	1034.353125	156.7708333	32
2	743	305	788	334	720.9421875	158.8541667	764.60625	173.9583333	42
2	1241	376	1279	421	1204.157813	196.875	1241.029688	219.2708333	17
2	548	324	593	383	531.73125	168.75	575.3953125	199.4791667	32
2	988	343	1113	413	958.86875	178.6458333	1079.957813	215.1041667	12
2	784	338	899	422	760.725	176.0416667	872.3109375	219.7916667	10

Figure 12. Distance Estimation Output Example

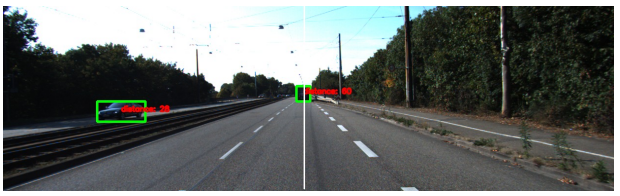

Figure 13. Distance Estimation Result

4) Visualize Final Results

In this part, we wrote a basic visualizer to do the following two functions

1. Write all the estimated data to the video frames.
2. Generate the video from its frames to visualize the results.

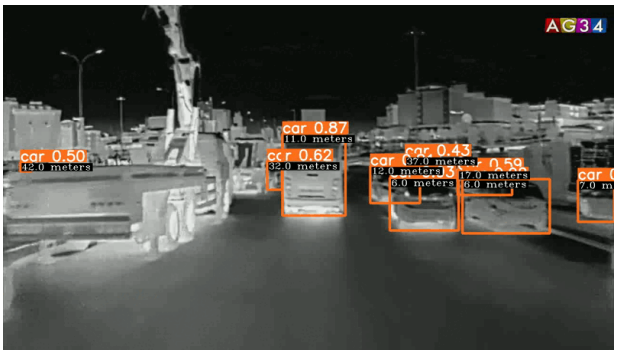

Figure 14. Visualized Final Result Example

5- Conclusion

In conclusion, we hope that we have clarified the problem faced by drivers, which is driving in bad weather, fog, lack of visibility, and many situations such as these. We hope our project to solve these problems using thermal imaging & ML/DL technologies.

We hope we can help drivers to see the road well regardless of vision problems, identify the surrounding vehicles, and determine the distance between them to enable them to make the right decisions.

We hope our project will serve people in maintaining their safety, benefit them and reduce accidents caused by bad vision.

6- References

[1] FLIR Thermal Images Dataset

[2] KITTI Dataset

[3] Owl-to-join-the-thermal-imaging-fray

[4] Arduino-Ultrasonic-Range-Finding-Sensor-for-20M-Distance-Measurement