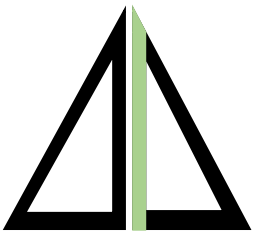
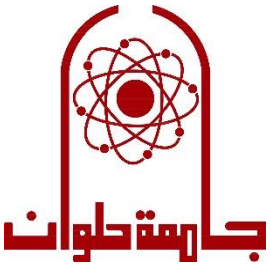


كلية الهندسة جامعه حلوان



Faculty Of Engineering

Computer Engineering Department

Helwan university

MODULAR-APPROACH SELF-DRIVING CAR SYSTEM

Supervised by

Assoc. Prof. Amr E. Mohamed

Presented by

Abdullah Nasser Mohamed

Abdulrahman Hassanin Ali

Assem Khaled Taha

Omar Shahat Ahmed

2021

Individual contributions

Name	Contributions
Abdullah Nasser Mohamed	• Introduction • Problem statement • Acknowledgement • Module 1 (Perception): ◦ Lane detection • Module 2 (Planning) part 1

Abdulrahman Hassanin	• Goals and objectives
	• Related works
	• Module 1 (Perception):
	  ◦ Depth estimation
	  ◦ Object Detection
	• System requirements
	• Future work

Assem Khaled Taha	• Module 1 (Perception):
	  ◦ Object detection
	  ◦ Traffic lights classifications
	• Module 3 (Control) part 1
	• Testing
	• Project timeline
	• Future work

Omar Shahat Ahmed	• Abstract • Tools and technologies • Module 1 (Perception): ◦ Transfer learning • Module 2 (Planning) part 2 • Module 3 (Control) part 2 • Finalize document

TABLE OF CONTENTS

LIST OF FIGURES

Acknowledgement

Would like to express my sincere gratitude to several individuals and organizations for supporting us throughout out Graduate study. First, we wish to express our sincere gratitude to our supervisor, Professor Amr El-Sayed, for his enthusiasm, patience, insightful comments, helpful information, practical advice and unceasing ideas that have helped us tremendously at all times in our research and writing of this thesis. His immense knowledge, profound experience and professional expertise in Deep Learning in general, and self-driving cars specifically has enabled us to complete this research successfully. Without his support and guidance, this project would not have been possible. We could not have imagined having a better supervisor in our study.

Abstract

The autonomous car or the driverless car can be referred to as a robotic car in simple language. This car is capable of sensing the environment, navigating and fulfilling the human transportation capabilities without any human input. It is a big step in the advancing future technology. Autonomous cars sense their surroundings with cameras, radar, lidar, GPS and navigational paths. Advanced control systems interpret sensory information to keep track of their position even though the conditions change. The advantages of autonomous cars, such as fewer traffic collisions, increased reliability, increased roadway capacity, reduced traffic congestion as well as reduction of traffic police and care insurance, are compulsive for the development of autonomous car even though we have to overcome the issues of cyber security, software reliability, liability of damage and loss of driver related jobs. Autonomous cruise control or the Lane departure warning system and the Anti-lock braking system (ABS) are the early steps. These steps though small are conclusive towards the progress in the direction of making the autonomous car. Companies such as Tesla, Google, Volvo, Mercedes-Benz and Audi are the fore runners in making the autonomous car a reality. The development and expansion of the sector in Indian conditions is also worth considering. We strongly believe that the autonomous car will be a reality soon and be a necessity of life by overcoming the current obstacles, as human life needs to be secured by safe, efficient, cost effective and comfortable means of transport.

1- Introduction

When a person gets behind the wheel in the morning, he/she is pretty confident that they will reach their destination safely. However, this is increasingly becoming untrue.

There are numerous reasons why a car accident may occur. Driver negligence, bad weather, poor road conditions, and the negligence of third parties could all be contributors, and sometimes, they are caused by a combination of several different factors.

Number of fatalities:

- Approximately 1.35 million people die in road crashes each year; on average 3,700 people lose their lives every day on the roads. That is, one person is killed every 25 seconds
- An additional 20-50 million suffer non-fatal injuries, often resulting in long-term disabilities.
- Road traffic injuries are the leading cause of death among young people aged 5-29. Young adults aged 15-44 account for more than half of all road deaths.
- Road crashes may cost countries 2-8% of their gross domestic product.
- Road crashes cost the U.S. more than $380 million in direct medical costs.
- Road crashes are the leading cause of death in the U.S. for people aged 1-54. [1]

And in our era of huge technological advancements, technologies like cars are becoming increasingly accessible to the point where almost every family owns at least a car. Thus, increasing the number of accidents exponentially.

Therefore, a gap in the market was created for self-driving-cars and self-Driving Cars have gone from “maybe Possible” to “definitely possible” to “inevitable”.

In fact, the industry market invests a huge effort to increase autonomous vehicle levels, as Research and Markets says: “The autonomous/driverless car market was valued at USD 24.10 billion in 2019”

The Society of Automotive Engineers (SAE) created “Levels of Driving Automation” standard that defines the six levels of driving automation, starting from Level 0 (No Automation) to Level 5 (Fully Autonomous). [2]

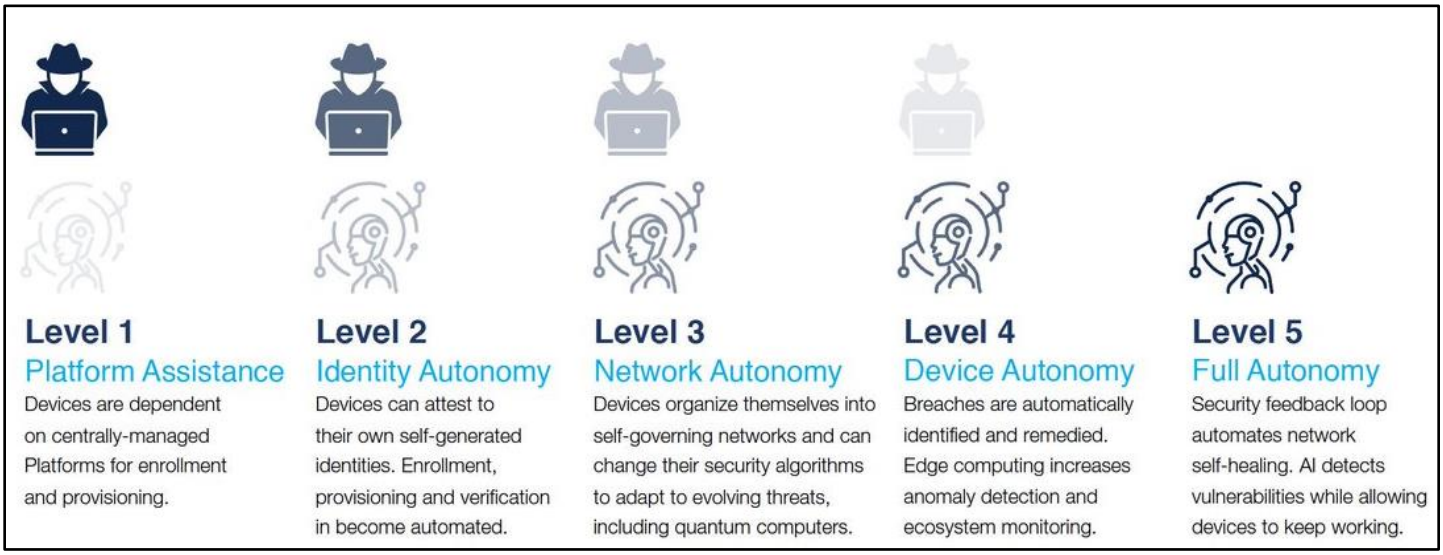


Figure 1: Levels of autonomous vehicles

2- Problem Statement

Having discussed car accidents in general, lets now check how many of them are preventable. One of the saddest facts about car accidents is that most of them are preventable. A 2016 study by the National Highway Transportation Safety Administration (NHTSA) found that human error accounts for anywhere between 94% to 96% of all auto accidents.

These human errors include:

- Speeding
- Aggressive/Reckless Driving
- Distracted Driving
- Drunk Driving
- Drowsy Driving

Vehicle passengers who grab the steering wheel or otherwise distract the driver. Vehicle owners or lessors who negligently entrust the car to someone who is known to be an unfit driver.

Drowsy driving is far more common than most people realize. A National Sleep Foundation study found that six out of every 10 Americans has driven when they were feeling sleepy, and 37% of those surveyed admit to having fallen asleep behind the wheel within the past year. [3]

Thus, we conclude that by removing humans from the process of driving, we eliminate most car accidents.

The Project idea aims to make an autonomous vehicle system that is capable of understanding the environment and navigating safely. The system should be able to sense the environment and make its own decisions without any human interference, thus achieving a level 5 autonomous vehicle.

3- Goals & Objectives

The aim is to develop a fully self-driving car system which gives the ability to the vehicle to sense the environment and understand the surrounding scene then operate to take decisions from the route to the destination without any human interaction.

Objective:

- Develop a self-driving car system to sense the environment and navigate safely.
- Develop a system based on modules that integrate with each other
- Test and evaluate the performance using Carla Simulator [4]
- Develop a perception module to produce scene understanding:
    - Detect static and dynamic objects
    - Detect and segment lanes of the road
- Develop depth estimation to determine each detected object distance
- Develop a planning module that aims to navigate the vehicle between points on the map
    - Implement a route planning algorithm to navigate the vehicle's route from the vehicle's position to the destination.
    - Implement a trajectory planning that updates the vehicle's trajectory whenever the object changes its state.
- Develop both longitudinal and lateral control algorithms to follow the planned trajectory

4- Related works

The system architecture is realized with two alternative approaches: modular or end-to-end driving. The classification of architecture is demonstrated in Figure:

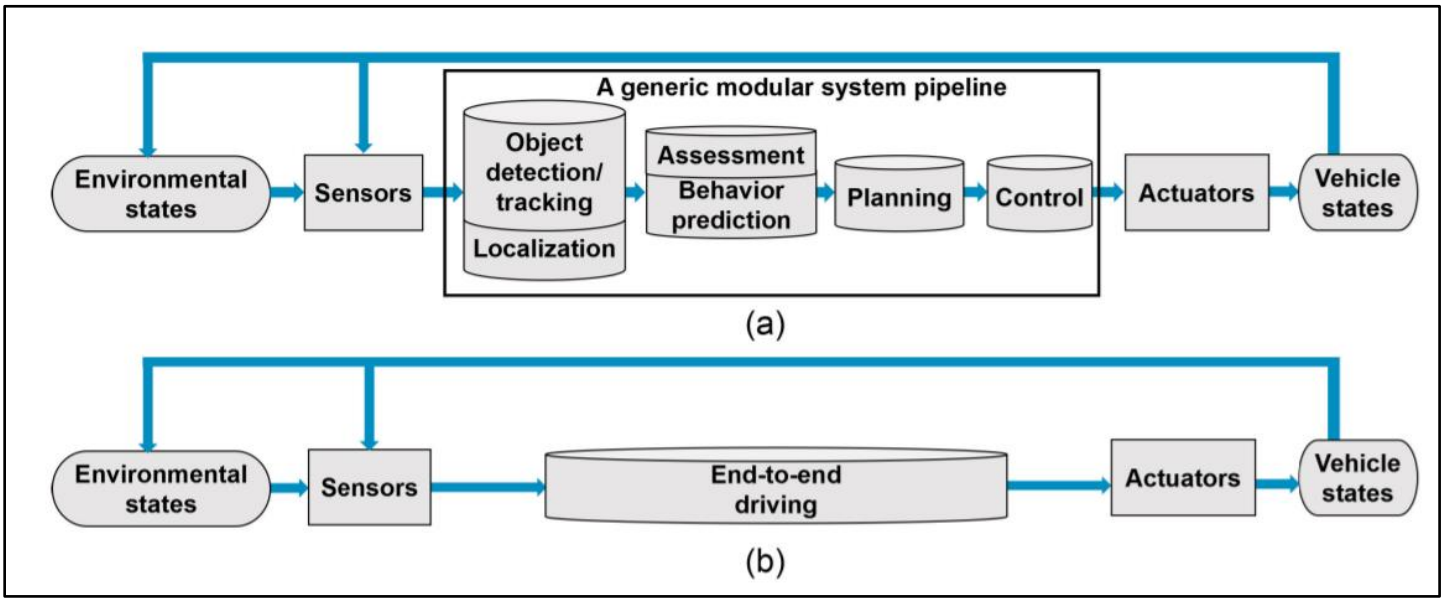


Figure 2: System architecture of (a) Modular system, and (b) End-to-End driving system

Modular System:

Modular systems sometimes called the mediated approach are structured as a pipeline of separate components linking sensors' inputs to actuators output.

The main functions of the modular approach can be summarized as perception, localization and mapping, planning and decision making, and vehicle control. The pipelines start by feeding sensor inputs to perception and prediction

modules, followed by planning and decision making. Finally, the motor command is generated at the end of the pipeline.

Developing individual modules separately divides the challenging task of automated driving into an easier-to-solve set of problems. These sub-tasks have their corresponding literature in robotics, computer vision, and vehicle dynamics, which makes the accumulated know-how and expertise directly transferable. This is a major advantage of modular systems. In addition, functions and algorithms can be integrated or built upon each other in a modular design. E.g, a safety constraint can be implemented on top of a sophisticated planning module to force some hardcoded emergency rules without modifying the inner workings of the planner. This enables designing redundant but reliable architectures.

The major disadvantages of modular systems are being prone to error propagation and over-complexity. In the unfortunate Tesla accident, an error in the perception module in the form of a misclassification of a white trailer as sky, propagated down the pipeline until failure, causing the first ADS-related fatality

End-to-End Driving System:

End-to-end driving, referred as direct perception, generates ego-motion directly from sensory inputs. Ego-motion can be either the continuous operation of the steering wheel and pedals or a discrete set of actions, e.g, acceleration and turning left. There are three main approaches for end-to-end driving: direct supervised deep learning [5] - [6], deep reinforcement learning [7] - [8], and neuroevolution [9] - [10]. The comparison of the approaches is given in Table.

Related Works	Learning strategy	Pros/Cons
[5], [6]	Direct supervised deep learning	Imitates the target data: usually a human driver. Can be trained offline. Poor generalization performance.
[7], [8]	Direct reinforcement learning	Learns the optimum way of driving. Requires online interaction. Urban driving has not been achieved yet
[9],[10]	Neuroevouation	No backpropagation. Requires online interaction. Real-world driving has not been achieved yet.

End-to-end driving is promising, however it has not been implemented in real-world urban scenes yet, except limited demonstrations. The biggest shortcomings of end-to-end driving in general are the lack of hard-coded safety measures

and interpretability. In addition, DQN and neuroevolution have one major disadvantage over direct supervised learning: these networks must interact with the environment online and fail repeatedly to learn the desired behavior. On the contrary, direct supervised networks can be trained offline with human driving data, and once the training is done, the system is not expected to fail during operation.

5- System requirements

5.1 Enumerated Functional Requirements:

REQ-x	Description
Choose destination	Enter the destination

5.2 Enumerated Nonfunctional Requirements

REQ-x	Description
Performance and scalability	Calculate the required information in real time with significant precision.
Portability and compatibility	The system is able to adapt on any type of car.
Reliability, availability, maintainability	In a modular approach system, an error doesn't propagate to the point of critical failure.
Usability	The system is very easy to use for user

5.3 Functional Requirements Specification:

5.3.a Stakeholders:

- User

5.3.b Actors and goals:

- User (Participating)

5.3.c Use case diagram:

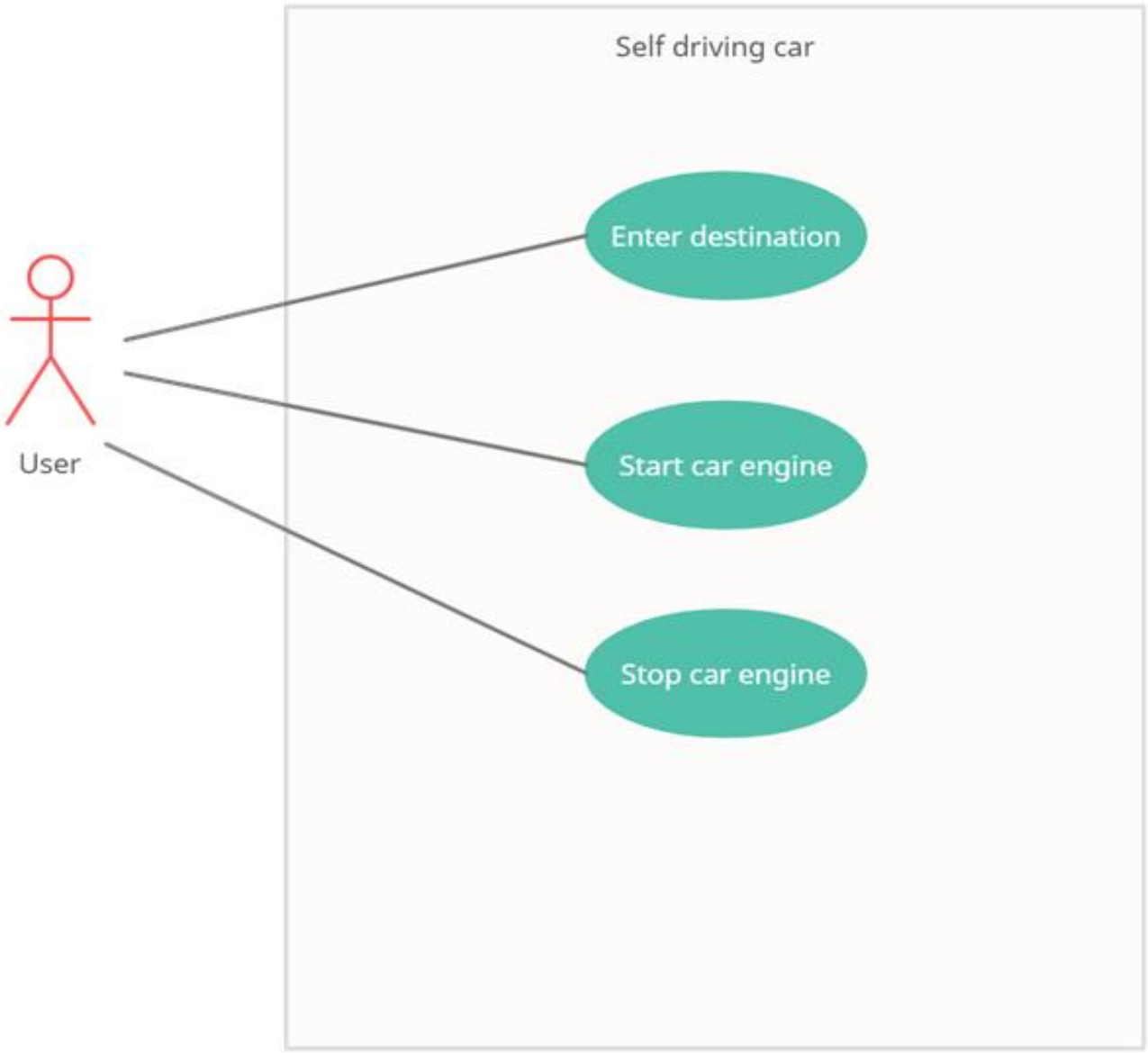


Figure 3: Use case diagram

Use case description

Use case	Description
Enter destination	The user enters his required destination

5.3.d Class diagram:

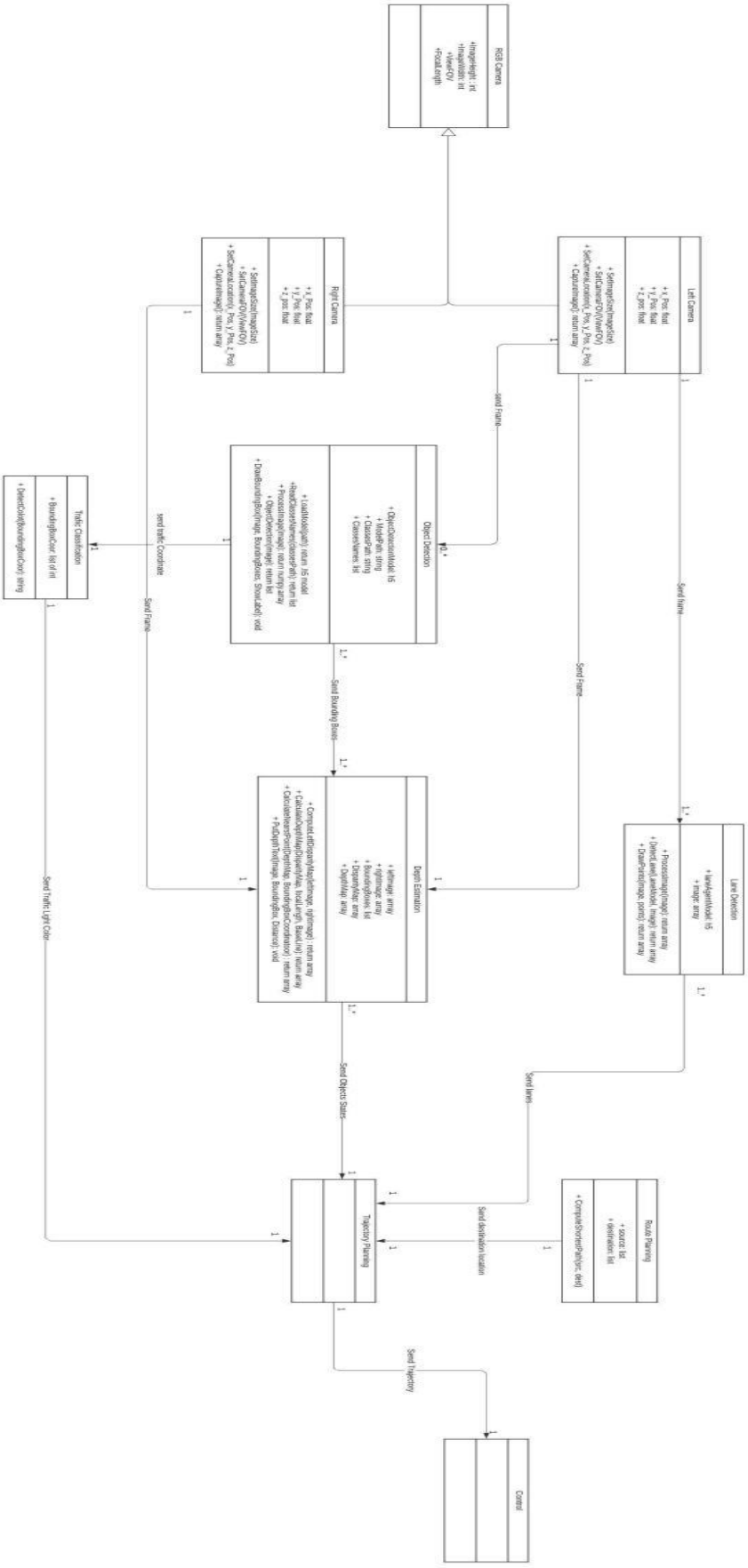


Figure 4: Class diagram

5.3.e Sequence Diagram:

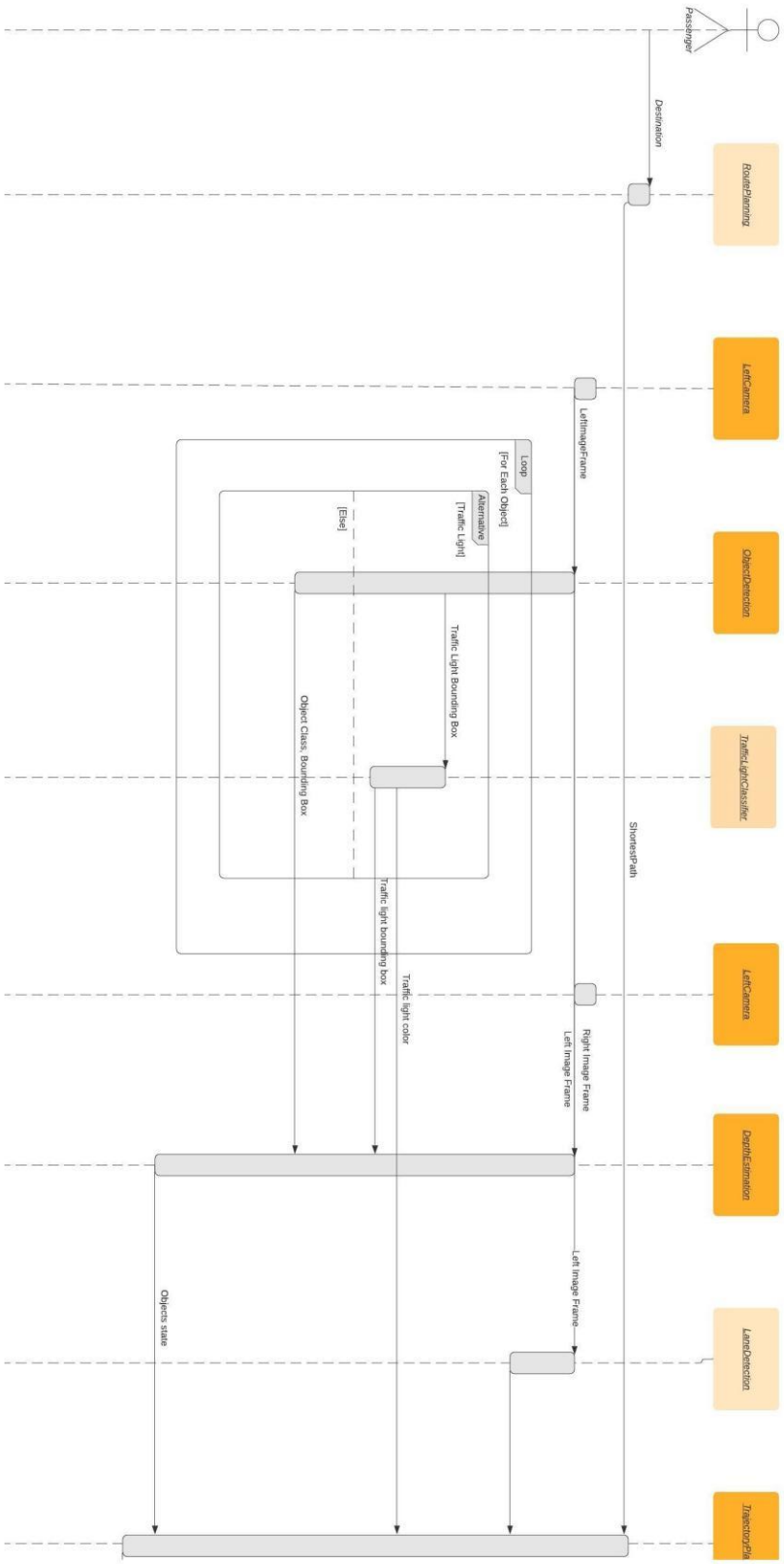


Figure 5: Sequence diagram

5.4 Block Diagram:

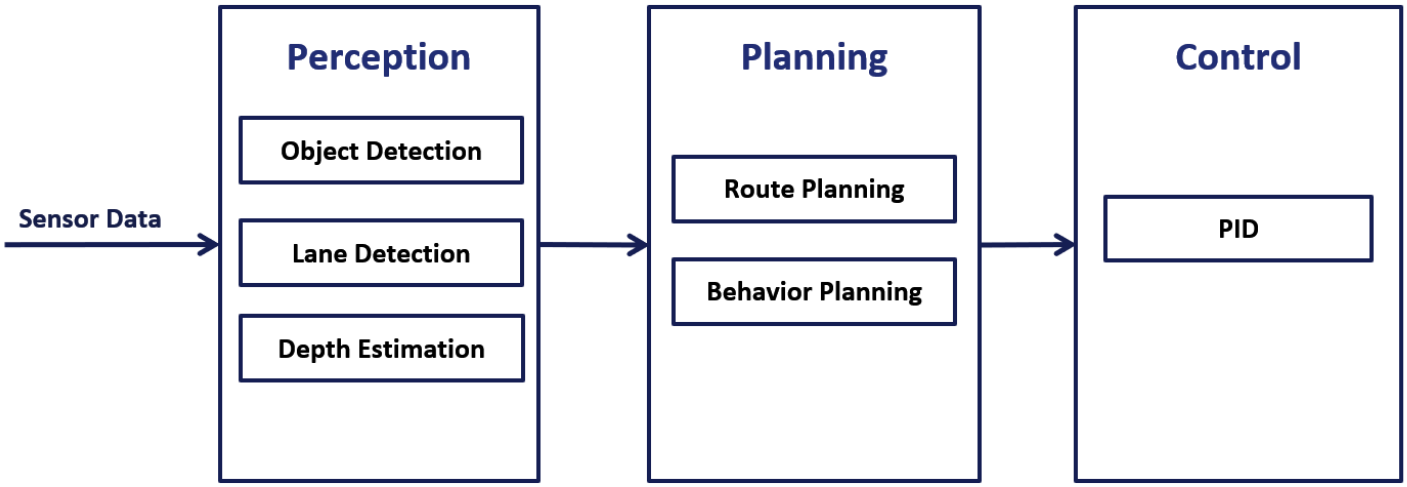


Figure 6: Block Diagram

6- Tools and technologies

In this section we will show all tools that will be used in our project

Technologies:

- Deep learning: Using techniques of deep learning for creating models for example, using convolutional neural network (CNN)
- Python: The programming language used

Tools:

1. **Software:**
   - CARLA simulator: The simulator used for simulating world and car

2. **Hardware:**
   - Camera: Used as a sensor in perception, planning and control modules

7-

Module 1 | Perception

01 Introduction

The process of perception in self-driving cars uses a combination of high-tech sensors and cameras, combined with state-of-the-art software to process, and comprehend the environment around the vehicle, in real-time.

Perception how the vehicle should move next; such that, the car is headed in the right direction (to its destination) and is not risking the lives of humans in, and around the car, while doing so.

Sensors like radars (a device that uses bouncing radio waves to map the world) and LIDARs (a device that uses echoing lasers to map the world), in combination with a series of cameras, are the car's eyes.

Software systems, like convolutional neural networks, act as the 'brains' of the vehicle, Together, the car can piece together a detailed understanding of what's happening in the world.

In self-driving cars is crucial to the safe and reliable operation because the data received from these processes fuel the core decision-making that figures out

Camera sensor:

The process of perception in self-driving Unlike any other sensor, cameras provide the richest information about the world around the car.

Radar and LIDAR sensors use various types of waves to map the world based on how far objects are from the sensor. In the process, these sensors can't actually see small (but hugely important) details.

Things like, what color the traffic light is, or if the road sign says a top speed of 60 km/h or 90 km/h, can't be determined, because these elements are depthless.

Another way to put it, machines need to replicate human vision to comprehend this information — after all, navigating roads, at least for now, was centered around a human driver.

On top of understanding depthless information, camera data is also paired with readings from LiDAR and radar sensors — through a process called sensor fusion — to confirm and label objects.

For example, a radar might identify a body of points directly in front of a vehicle, and conclude its one moving objects, with x velocity and y distance from the driverless car. However, a camera feed in the same direction will be used to confirm that this indeed is a car, and not, for example, a cyclist. This process is crucial in self-driving vehicles because knowing the difference between objects will change what decisions are made in terms of the next movements the car should make.

Figure 7: Example of camera sensor

Cameras are used to identify the presence and meaning of four main attributes of the driving environment:

1. Traffic light detection and classification (so the car knows if there's a traffic light and what color it’s displaying)
2. Lane detection and classification (so the car knows it must drive in a specific area, and if it’s allowed to cross those lines: double yellow or broken white)
3. Road sign detection and classification (so it knows where to look for road instructions, and what exact instructions they are for a specific roadway)
4. Object detection, classification and tracking (seeing what objects pose a potential threat, figuring out what it is and if it's valuable, and tracking its position)

A video feed alone does not automatically come with labelled elements, so it’s up to the car’s computer to take in the video feed, understand what’s being looked at, detect, classify and track the important elements, process

information from other sensors and sources, and make a well-informed driving decision.

Machine learning and computer vision process are used in order to do this.

The use of convolutional neural networks, or CNNs, is quite prominent in the processing of video feed for decision-making and path planning in autonomous vehicles.

02 Object detection

Object detection involves predicting the class or the type of the object and perform localization to estimate the location(coordinates) of the object in the image.

The main difference between image classification and object detection that image classification outputs a list of the object classes present in the image, or binary image classification which predict the presence of an object in the image.

Object detection outputs a list of the object classes present in the image along with the (x, y, w, h) coordinates of the location of the object in an image.

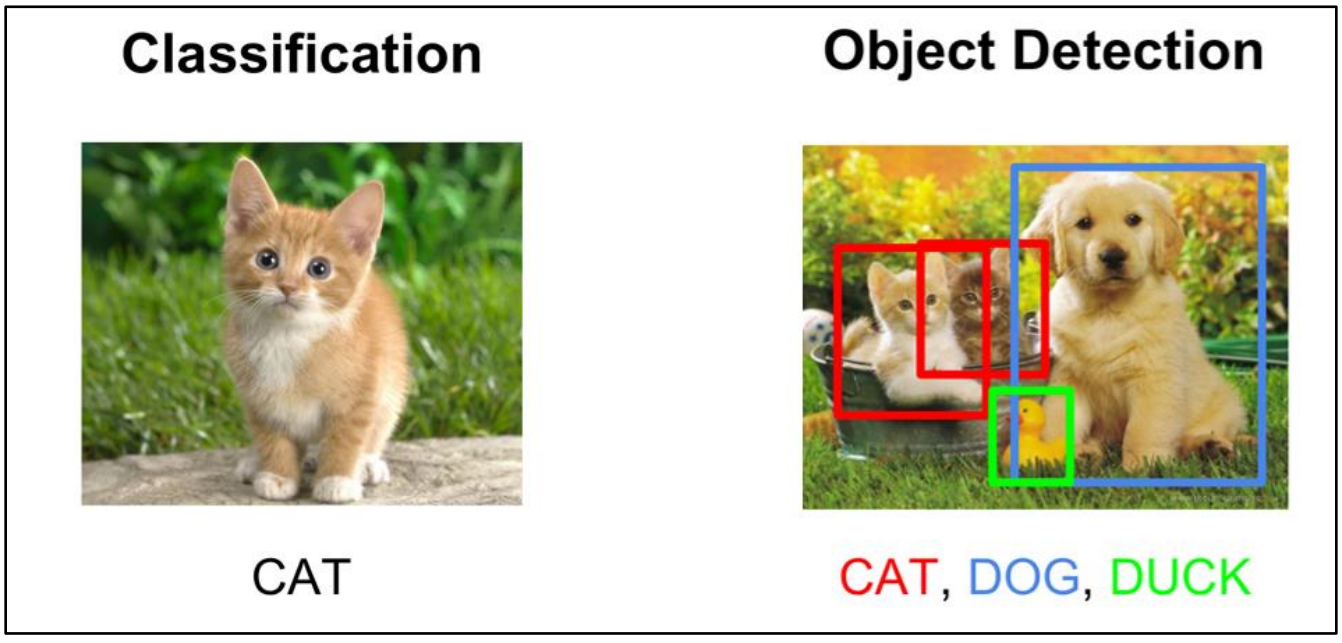


Figure 8: Image Classification & Object detection

In image classification we can use CNN (Convolution Neural Network) which follows a series of steps to classify an image:

The CNN first takes an input image then puts that image through several convolution and pooling layers the result is a set feature maps reduced in size from the original image that through the training process have learned to distil the information about the content in the original image, then we flatten these feature maps to create a feature vector that we can then pass through a series of fully connected linear layer to produce a probability distribution of class scores, from this we can extract the predicted class for the input image. So, in short an image comes in and predicted class labels comes out.

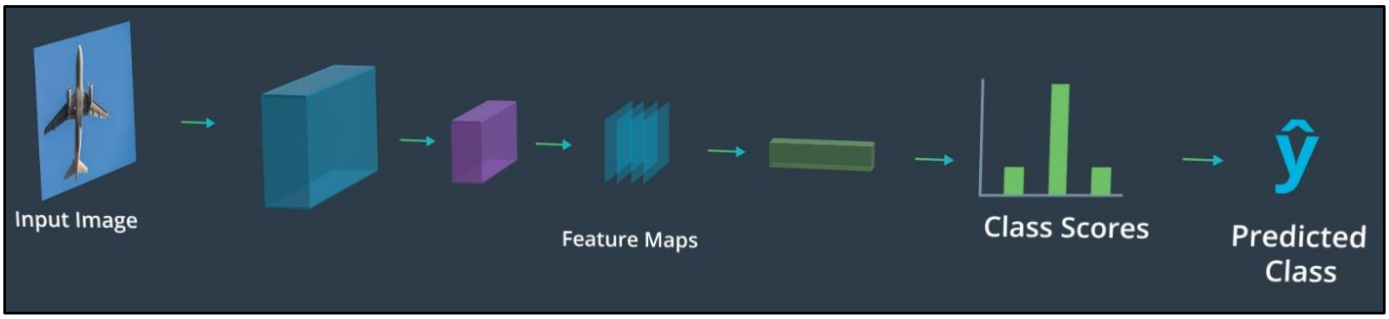


Figure 9: CNN for Image Classification

In Image classification there is usually a single object per image that the network is expected to classify, but in real world we are often faced with much more complex visual scenes with many overlapping objects.

In object detection we can perform localization by drawing a bounding box around the object which is a collection of coordinates that defines the box. The coordinates (x,y) define the center of the box and (w,h) defines the width and height of the box. To find these values we can use the same structure as in a typical classification CNN.

One way to perform localization is to first put a given image through a series of convolutional and pooling layers and create a feature vector for that image. We keep the same fully connected layers to perform classification, and we add another fully connected layer attached to the feature vector whose job is to predict the location and size of a bounding box.

So, in this one CNN, we have one output path whose job is to produce a class for the object pictured in an image and another whose job is to produce the bounding box coordinates for that object.

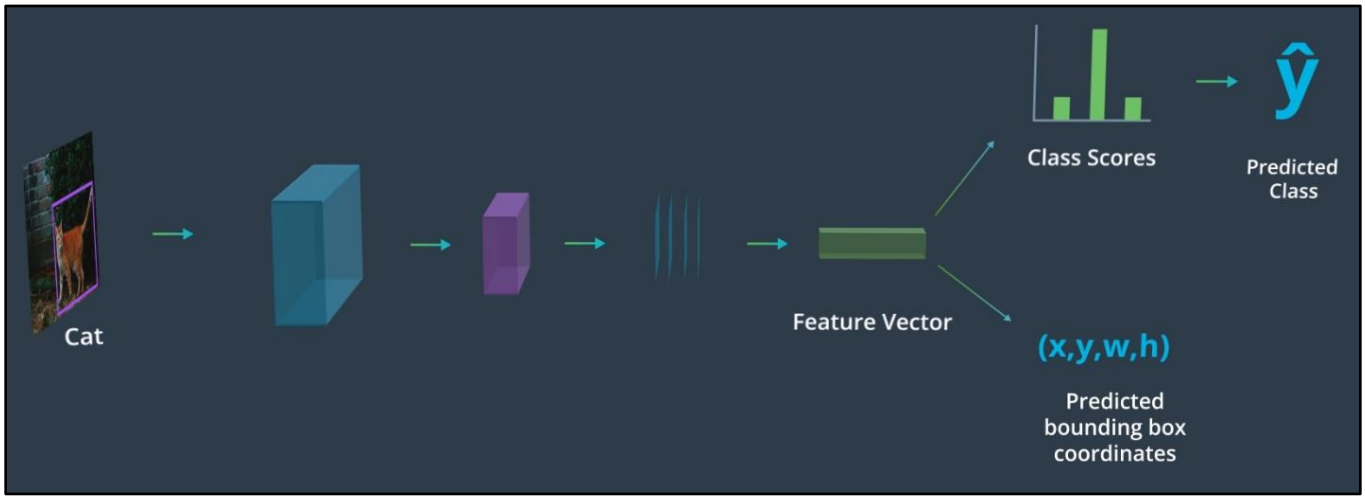


Figure 10: CNN for predicting bounding box

In this case, we're assuming that the input image not only has an associated true label, but it also has a true bounding box. This way, we can train our network by comparing the predicted and true values for both the classes and bounding boxes.

And we can use something like crossentropy loss to measure the performance of a classification model. Cross entropy operates on probabilities with values between zero and one, but for the bounding box, we need a function that measures the error between a predicted bounding box and a true bounding box.

When we look at comparing a set of points, say locations of points that define a specific region in an image, we need a loss function that measures the similarity between these coordinate values. So, this is a regression problem rather than a classification problem. Classification is about predicting a class label, and regression is about predicting a quantity.

For regression problems like predicting (x,y) coordinate locations, we need to use a loss function that compares these quantities, and gives us a measure of their closeness.

We should also note that with classification problems, we have an idea of accuracy. If our predicted class matches the true class label, then our model is accurate. But with regression problems, we can't really say whether a point is accurate or not. So, for regression problems, we often talk about models with a small error rather than models that are accurate.

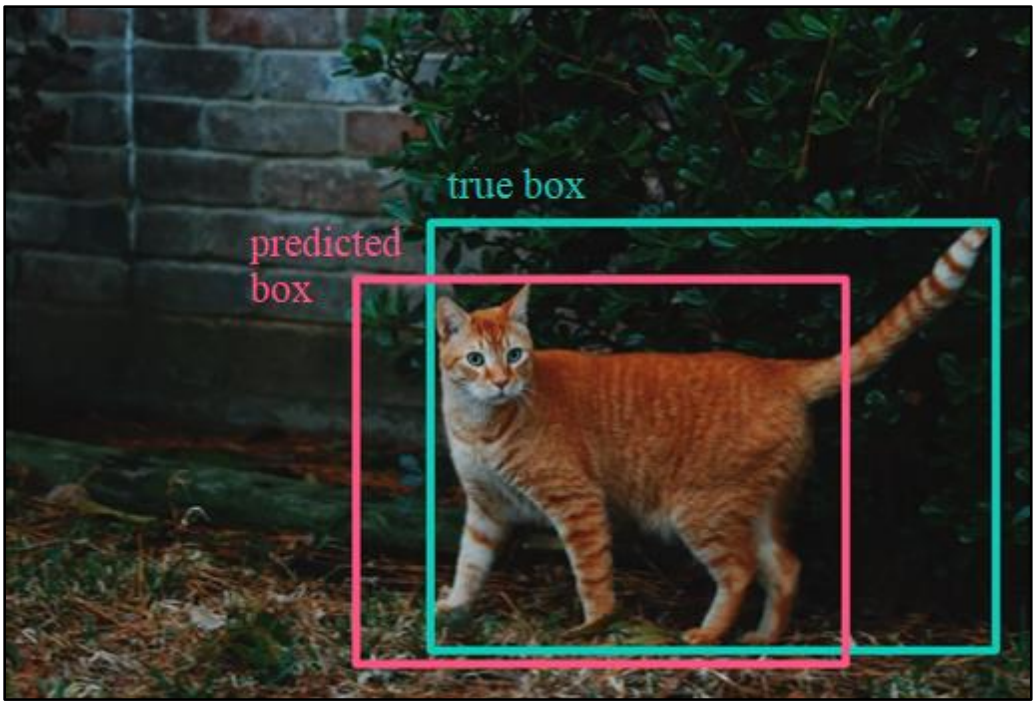


Figure 11: True & Predicted bounding boxes

To measure the error between two quantities, we have a few different types of loss functions that we can use.

The simplest measure is L1 loss, which measures the element wise difference between a predicted output (P) and a target (T).

Figure 12: P & T points for L1 Loss

In the case of (x,y) coordinate that indicates the center of an object in an image. the loss function will look at the predicted point P that was generated by a CNN and the true target location T of the center of the object and L one loss would return a value that represents the distance between the predicted and true points.

We also have MSE loss, which measures the mean squared error between the elements in a prediction P and a target T. Both of these methods can be good for measuring the distance between points, but all loss functions have their strengths and weaknesses.

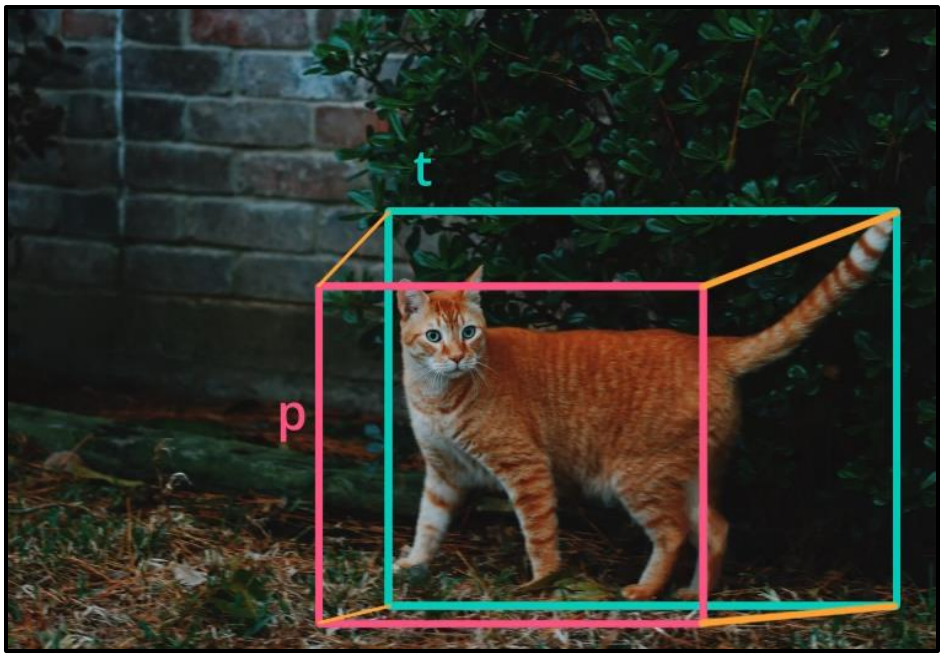


Figure 13: P & T bounding box for MSE Loss

We may consider that L1 loss can become negligible for small error values and that MSE loss responds the most to large errors, and so it may end up amplifying errors that are big, but infrequent also known as outliers.

There's also smooth L one loss, which for small differences between predicted and true values, uses a square error function and for larger errors, uses L1 loss. So smooth L1 loss tried to combine the best aspects of MSE and L1 loss.

Ideally, object detection is more complex than image classification and we face many challenges associated with object detection:

1. Variable Number of Objects

- The number of objects to be detected might vary from image to image.
- the main problem associated with this is that in Machine Learning models, we usually need to represent the data in fixed-sized vectors.

2. Multiple Spatial Scales and Aspect Ratios

- some objects that cover most of the image and yet there will be some we may want to find but are as small as a dozen pixels.
- Even the same objects can have different Scales in different images.

3. Modeling

- Doing object detection requires solving two approaches at once-Object Detection and Object Localization.

4. Speed for Real-Time detection

- Usually, a video is shot at almost 24 fps and to build an algorithm that can achieve that frame rate is quite a difficult task.

5. Overlapping objects

Object detection techniques:

Some of the recent top-performing deep learning models includes sliding window algorithm, region-based models (R-CNN, Fast R-CNN, Faster R-CNN), grid cell-based models (YOLO family), and SSD (Single Shot Detector).

Sliding window technique:

The idea is that we could just make a bunch of cropped regions to make sure we don't miss any objects in the image. This would mean defining a small sliding window and passing it over the entire image to create many different crops of the original input image. Then for each cropped region, we can put it through a CNN and perform classification.[12]

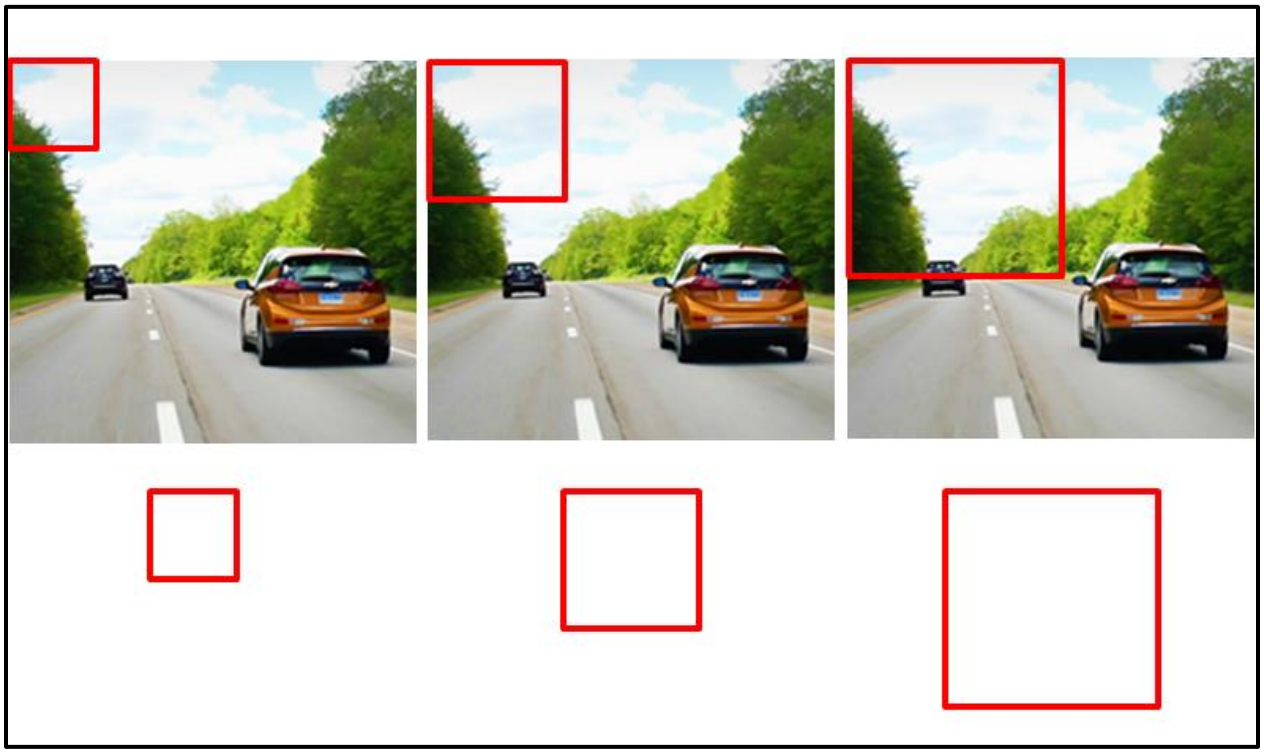


Figure 14: Sliding Window technique

However, this approach produces a huge amount of cropped images and is extremely time intensive.

Also, in this case, most of the cropped images don't even contain objects, especially when objects vary in size and location.

Region Proposals based CNN:

Region proposals give us a way to quickly look at an image and generate regions only for areas in which we think there may be an object. We can use traditional computer vision techniques that detect things like edges and textured blobs to produce a set of regions in which objects are most likely to be found, areas of similar texture or the same unifying boundary for example.

These proposals often produce noisy nonobject regions, but they are also very likely to include the regions in which objects are located, and so the noise is considered a worthwhile cost for not missing any objects.

We can use a region proposal algorithm to produce a limited set of cropped regions, often called regions of interest or ROIs, then we put these regions through a classification CNN one at a time and see what kind of class label the network predicts for each crop. This model is called an R-CNN (region convolutional neural network). [13]

The RCNN produces a class for each region of interest, so it can identify the region that is an object in the image. In this case, we also include a class called background that's meant to capture any noisy regions.

Since these regions are often different sizes, they first need to be transformed and warped into a standard size that a CNN can accept as input.

The main shortcoming of this method is that it's still time intensive because it requires that each cropped region goes through an entire CNN before class label can be produced.

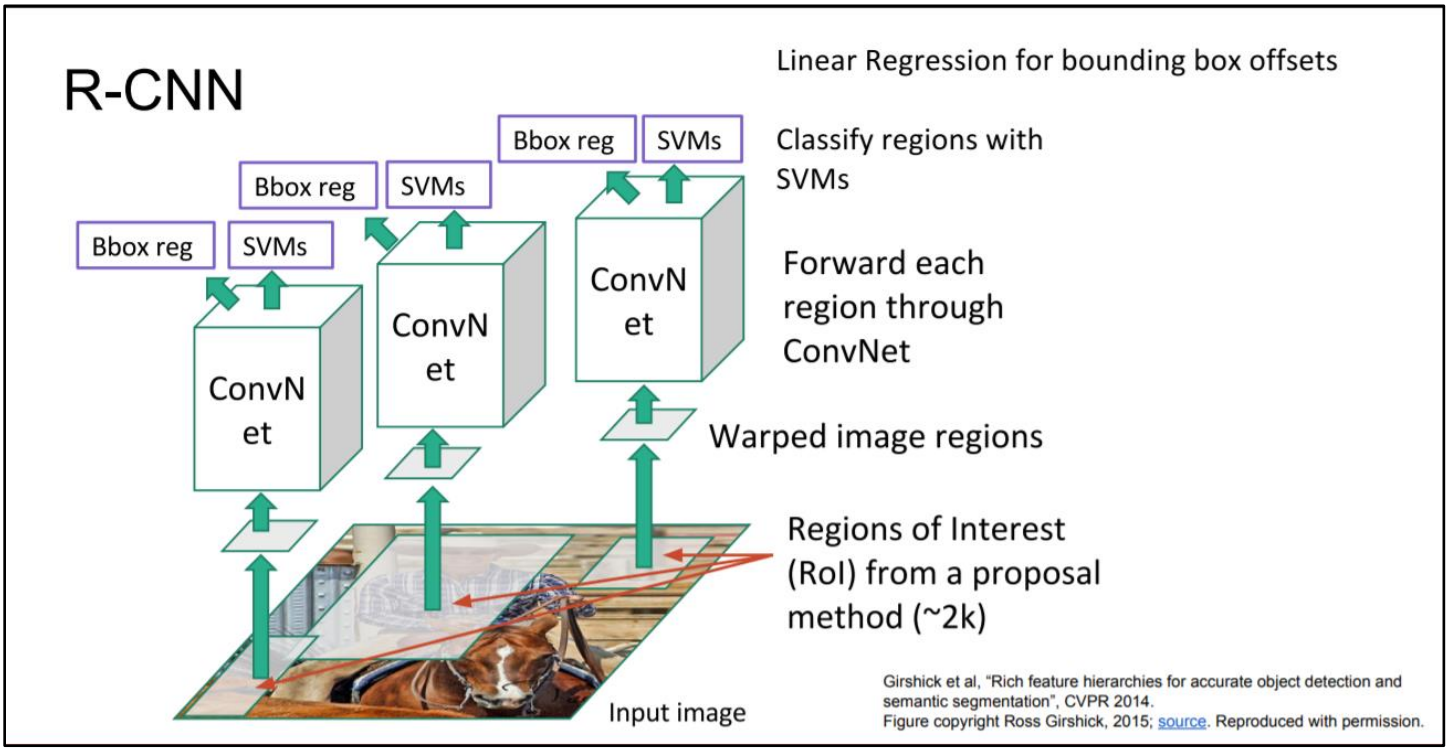


Figure 15: R-CNN

Fast R-CNN:

The next advancement in region-based CNNs came with a Fast R-CNN architecture. **[14]**

Instead of processing each region of interest individually through a classification CNN, this architecture runs the entire image through a classification CNN only once. The image goes through a series of convolutional and pooling layers, and at the end of these layers we get a stack of feature maps. We still need to identify regions of interest, but instead of cropping the original image, we project these proposals into the smaller feature map layer.

Each region in the feature map corresponds to a larger region in the original image, so we can grab selected regions in this feature map and feed them one by one into a fully connected layer that generates a class for each of these different regions.

In this model, we complete the most time consuming steps, processing in an image through a series of convolutional layers only once, and then selectively use that map to get our desired outputs.

Again, we have to handle the variable sizes in these projections, since layers further in the network are expecting input of a fixed size. So we do something called ROI pooling to Warp these regions into a consistent size before giving them to a fully connected layer.

This network is faster than RCNN, but it's still kind of slow when faced with a test image for which it has to generate region proposals. And it's still looking at regions that do not contain objects at all.

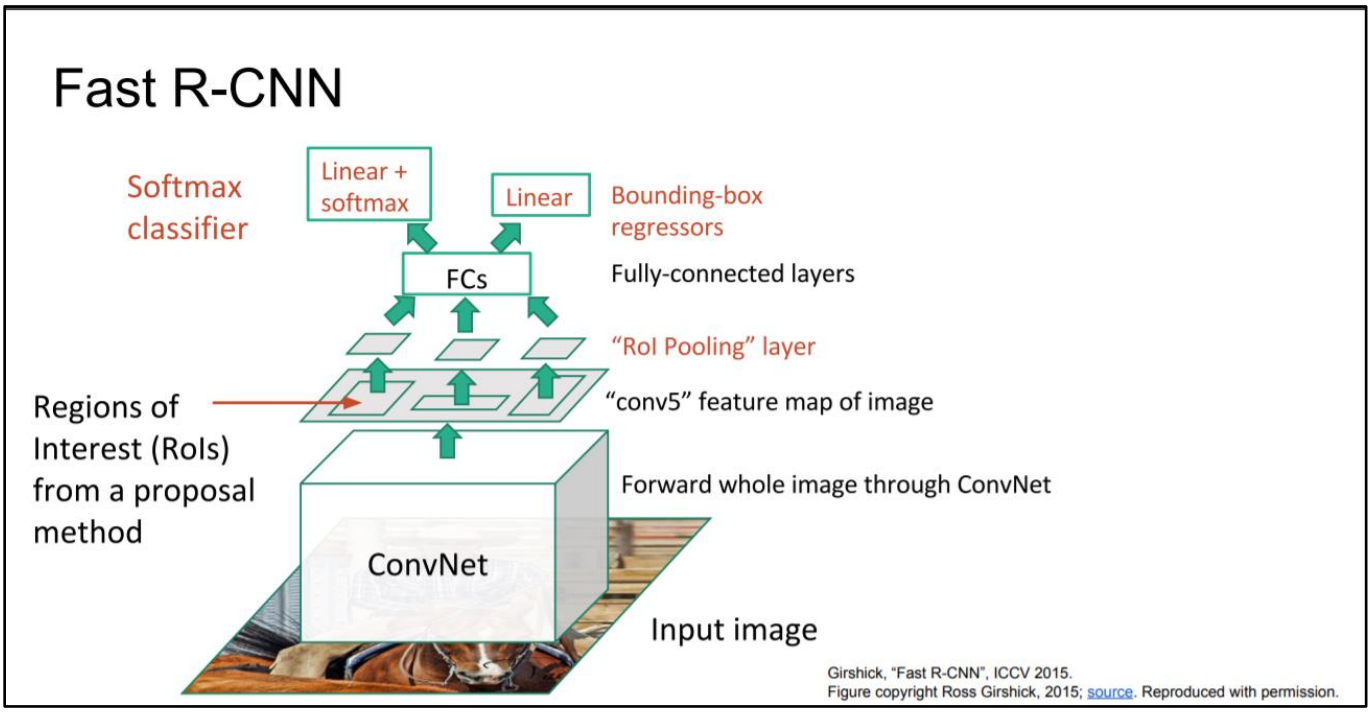


Figure 16: Fast R-CNN

Faster R-CNN:

Faster RCNN learns to come up with its own region proposals. It takes in an input image, runs it through a CNN up until a certain convolutional layer, just like Fast R-CNN. But this time it uses the produced feature map as input into a separate region proposal network, so it predicts its own regions from the features produced inside the network. [15]

If an area in the feature map is rich in detective edges or other features, it's identified as a region of interest, then this part of the network does a quick binary classification.

For each ROI it checks whether or not that region contains an object. If it does, then the region will continue on and go through the classification steps. And if it doesn't, then the proposal is discarded.

Once we have the final region proposals, the rest of the network looks the same as Fast-RCNN. It takes cropped regions from the feature map and learns to classify those regions.

By eliminating the analysis of nonobject regions. This model is the fastest of all the region based CNNs above.

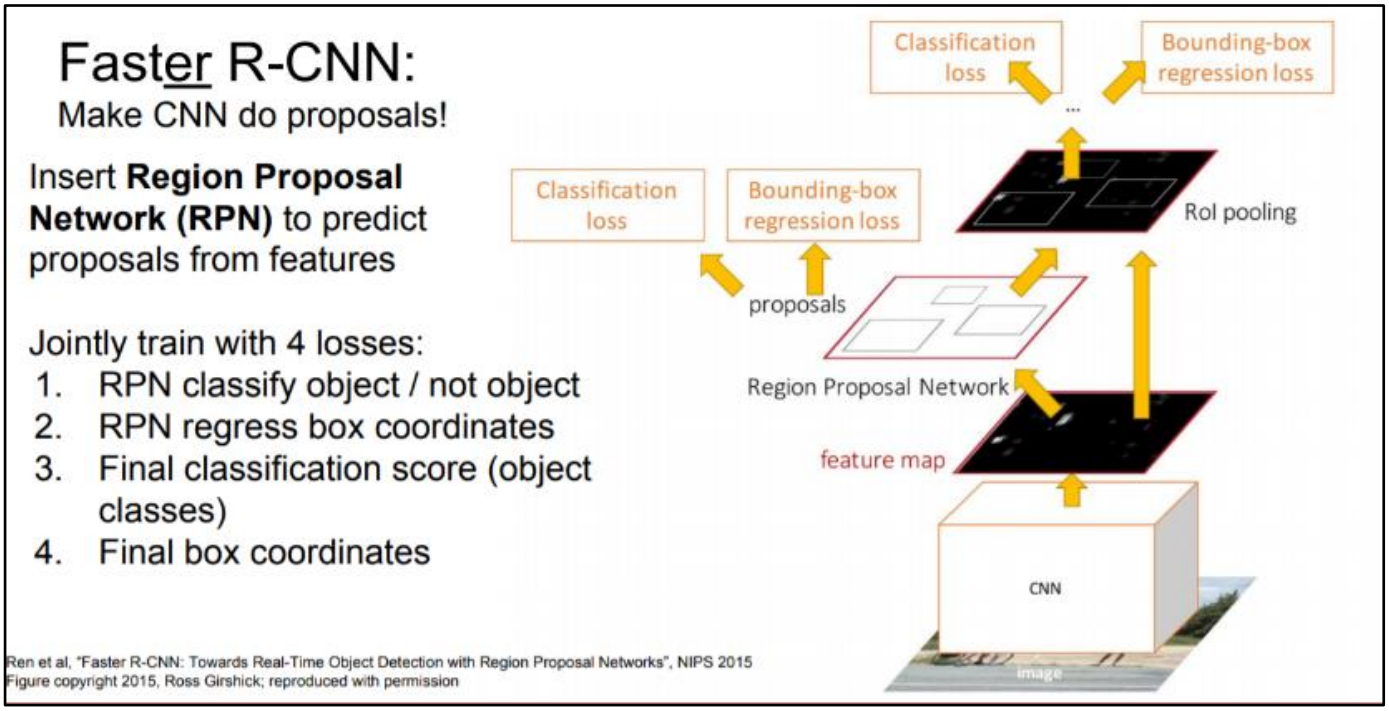


Figure 17: Faster R-CNN

YOLO (You Only Look Once):

YOLO algorithm gives a much better performance on all the parameters we discussed along with a high fps for real-time usage.

YOLO algorithm is an algorithm based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in one run of the Algorithm.

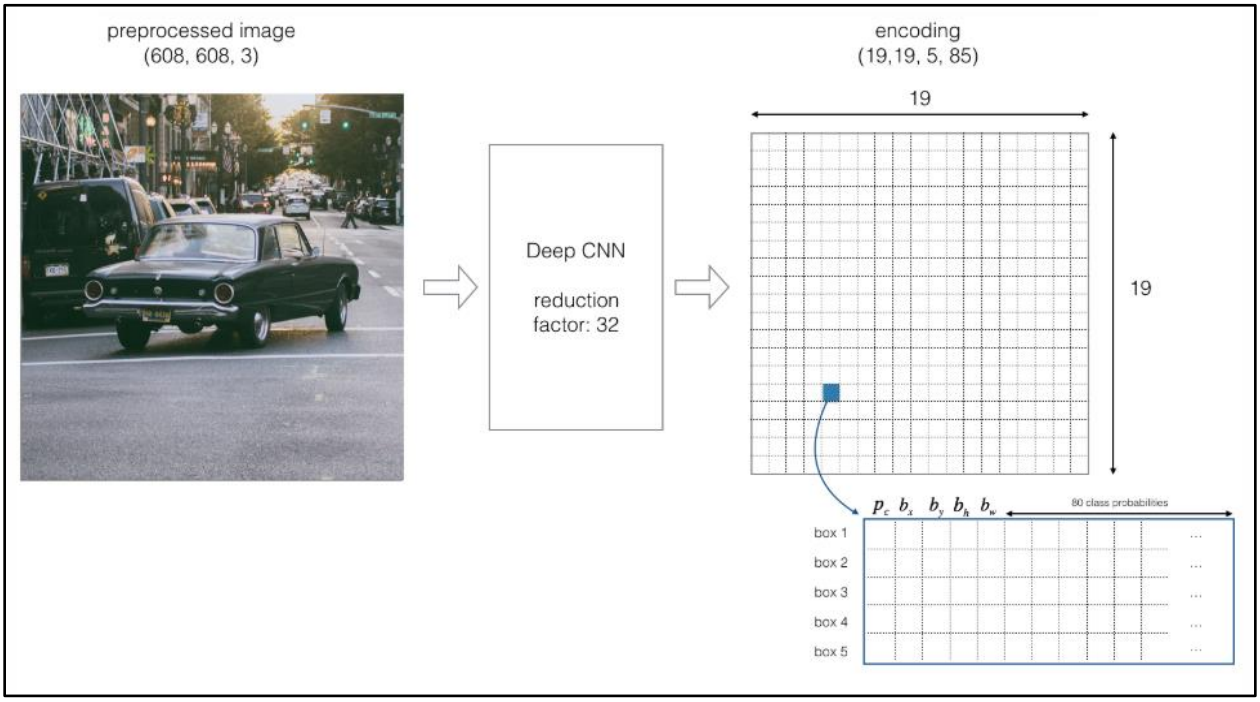


Figure 18: Grid Cells in YOLO algorithm

YOLO doesn't search for interested regions in the input image that could contain an object, instead it splits the image into cells, typically 19x19 grid. Each cell is then responsible for predicting K bounding boxes. The class with the maximum probability is chosen and assigned to that particular grid cell. And each cell has an associated vector that tells us:

- If an object in that cell.
- The ‘’class’’ if that object.
- The predicted bounding box (can be larger than the cell) for that object.

So, YOLO uses a convolutional approach to sliding windows technique.

Let’s assume we have a 16 x 16 x 3 image, like the one shown below. This means the image has a size of 16 by 16 pixels and has 3 channels, corresponding to RGB.

Figure 19: 16 x 16 RGB image

Let’s now select a window size of 10 x 10 pixels as shown below:

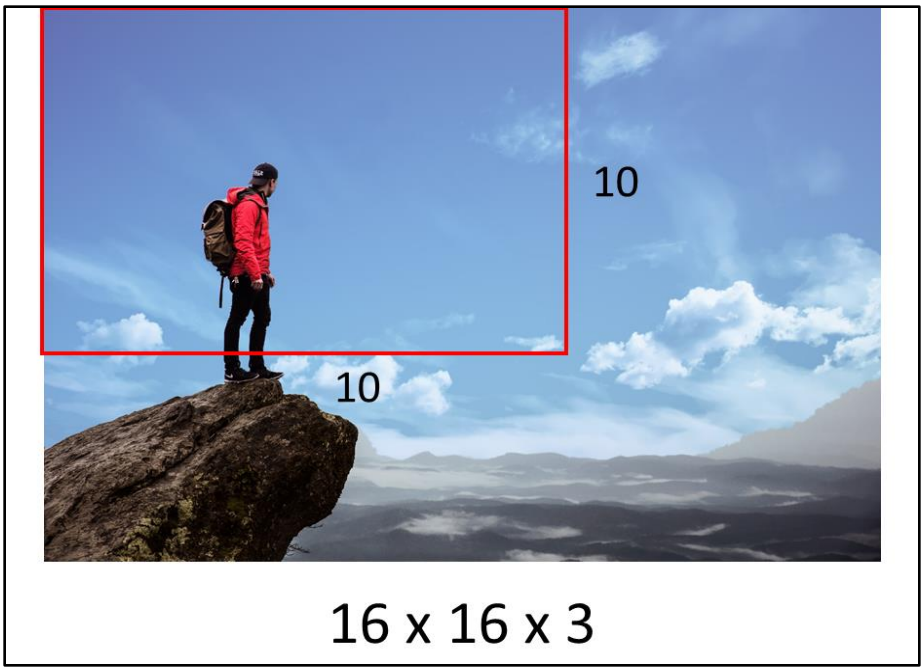


Figure 20: 10 x 10 window size

If we use a stride of 2 pixels, it will take 16 windows to cover the entire image, as we can see below.

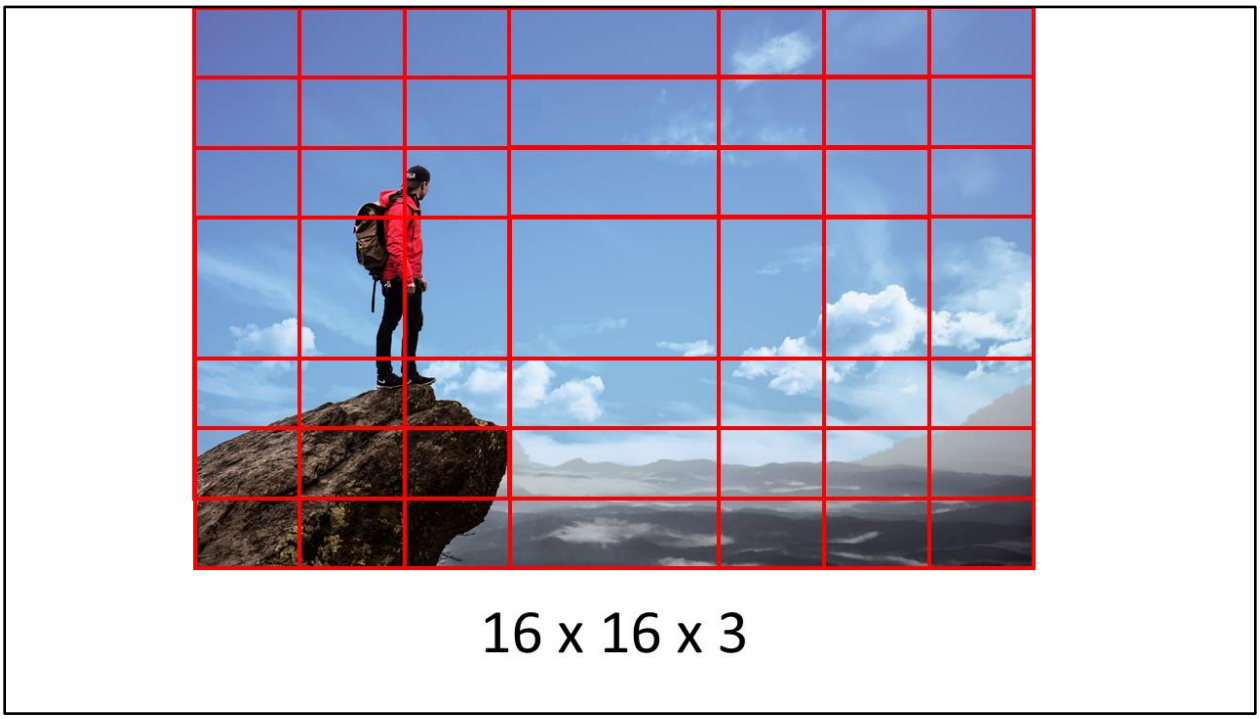


Figure 21: Windows over the full image

In the original Sliding Windows approach, each of these 16 windows will have to be passed individually through a CNN. Let's assume that CNN has the following architecture:

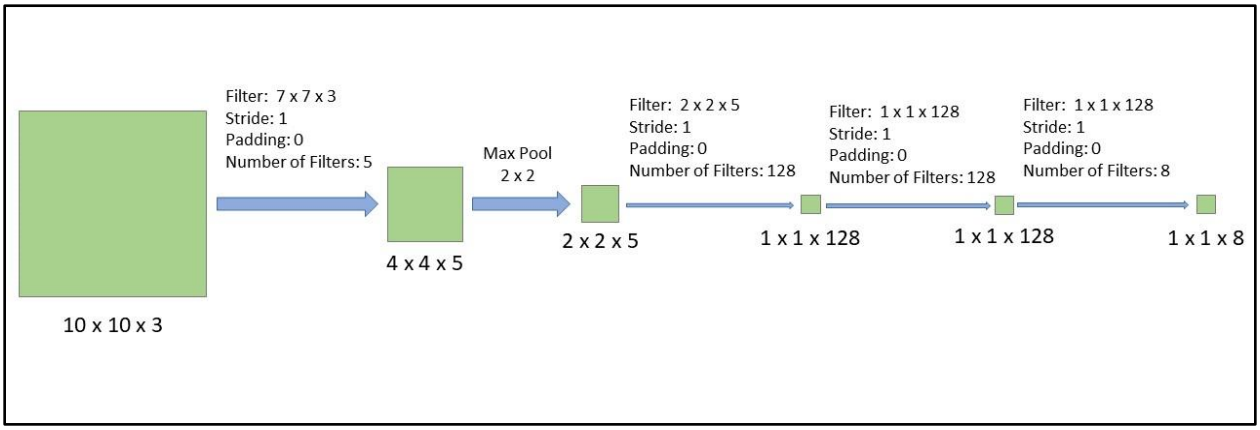


Figure 22: Traditional CNN architecture

The CNN takes as input a 10 x 10 x 3 image, then it applies 5, 7 x 7 x 3 filters, then it uses a 2 x 2 Max pooling layer, then is has 128, 2 x 2 x 5 filters, then is has 128, 1 x 1 x 128 filters, and finally it has 8, 1 x 1 x 128 filters that represents a SoftMax output.

If we change the input of the above CNN from 10 x 10 x 3, to 16 x 16 x 3 The result is shown below:

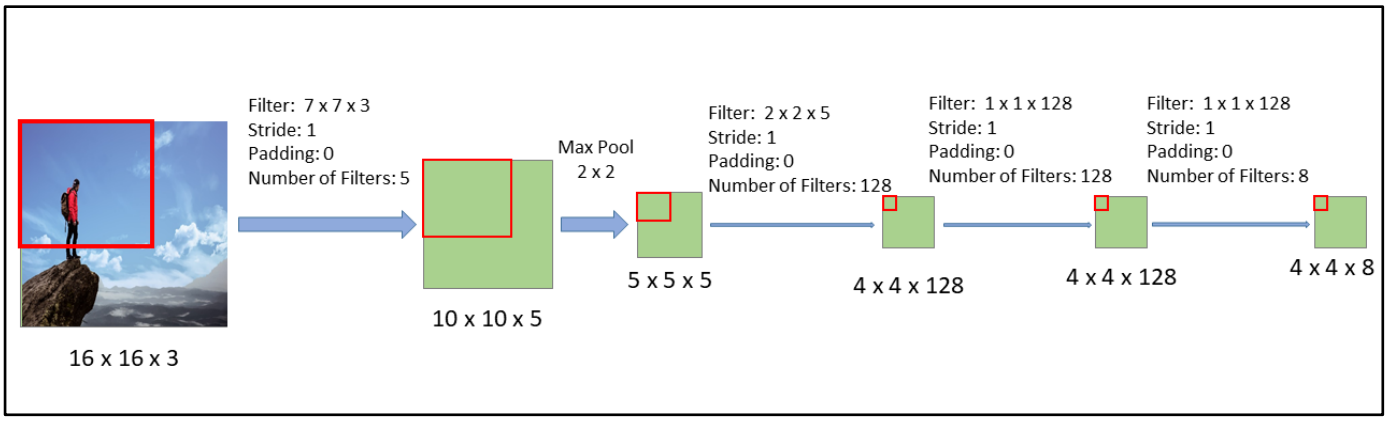


Figure 23: CNN architecture

As we can see, this CNN architecture is the same as the one shown before except that it takes as input a 16 x 16 x 3 image. The sizes of each layer change because the input image is larger, but the same filters as before have been applied.

If we follow the region of the image that corresponds to the first window through this new CNN, we see that the result is the upper-left corner of the last layer (see image above). Similarly, if we follow the section of the image that corresponds to the second window through this new CNN, we see the corresponding result in the last layer:

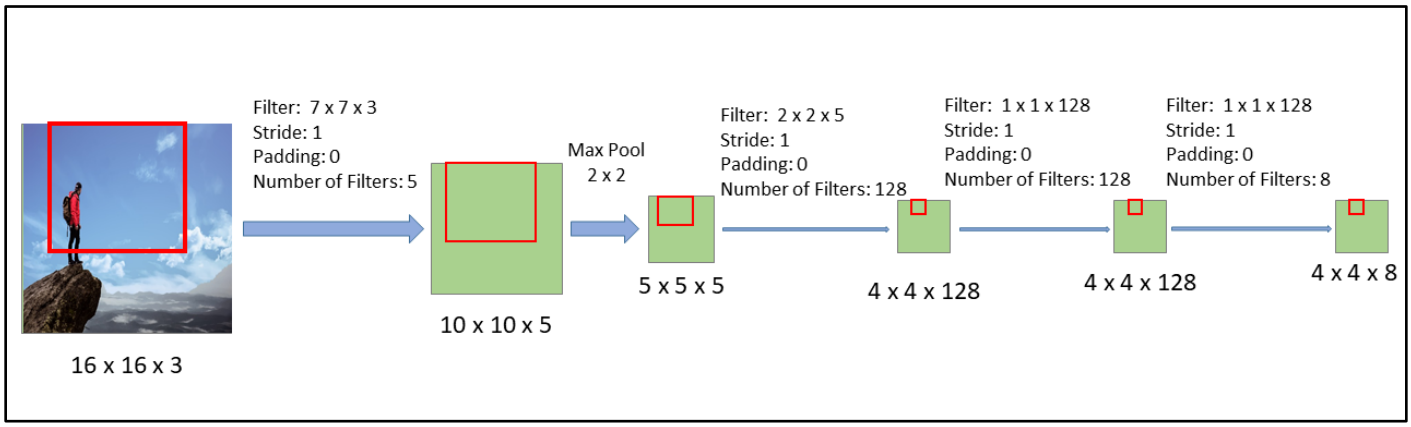


Figure 24: CNN architecture

Likewise, if we follow the section of the image that corresponds to the third window through this new CNN, we see the corresponding result in the last layer, as shown in the image below:

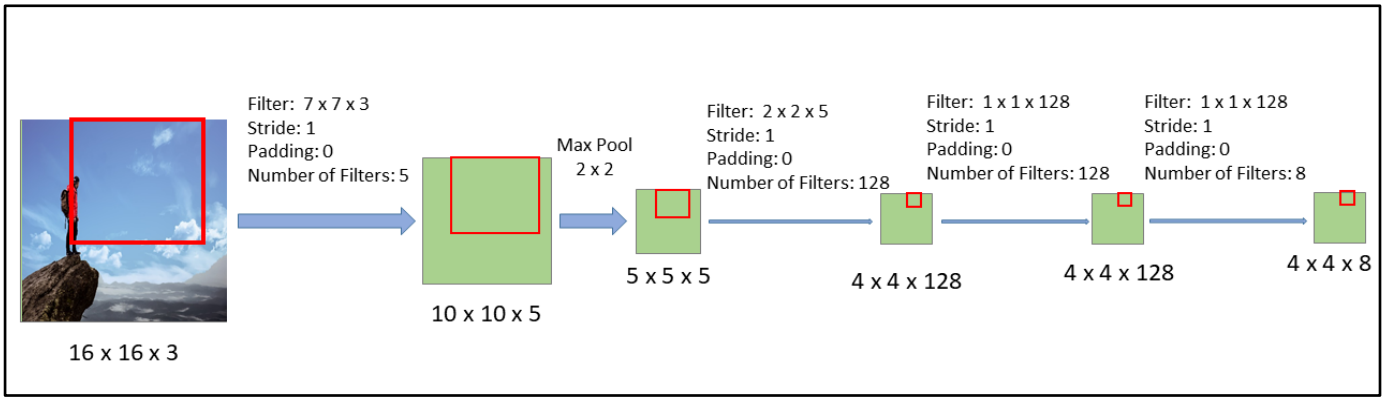


Figure 25: CNN architecture

Finally, if we follow the section of the image that corresponds to the fourth window through this new CNN, we see the corresponding result in the last layer, as shown in the image below:

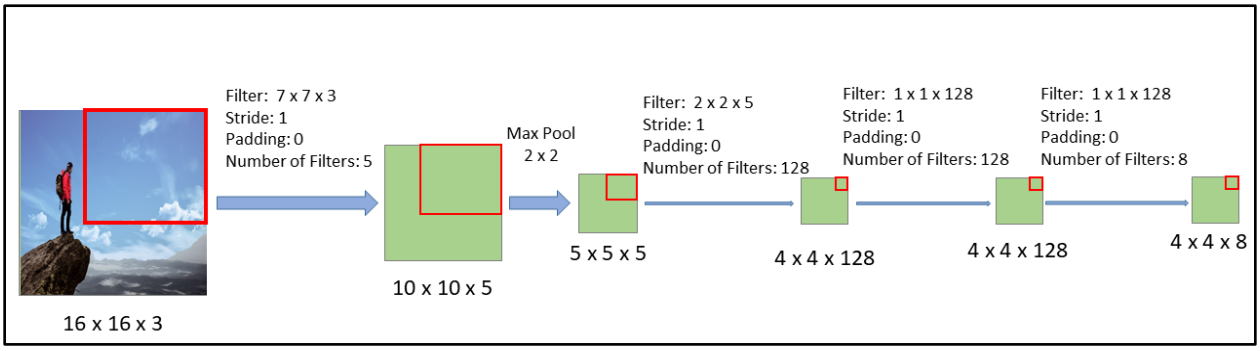


Figure 26: CNN architecture

In fact, if we follow all the windows through the CNN we see that all the 16 windows are contained within the last layer of this new CNN. Therefore, passing the 16 windows individually through the old CNN is exactly the same as passing the whole image only once through this new CNN.

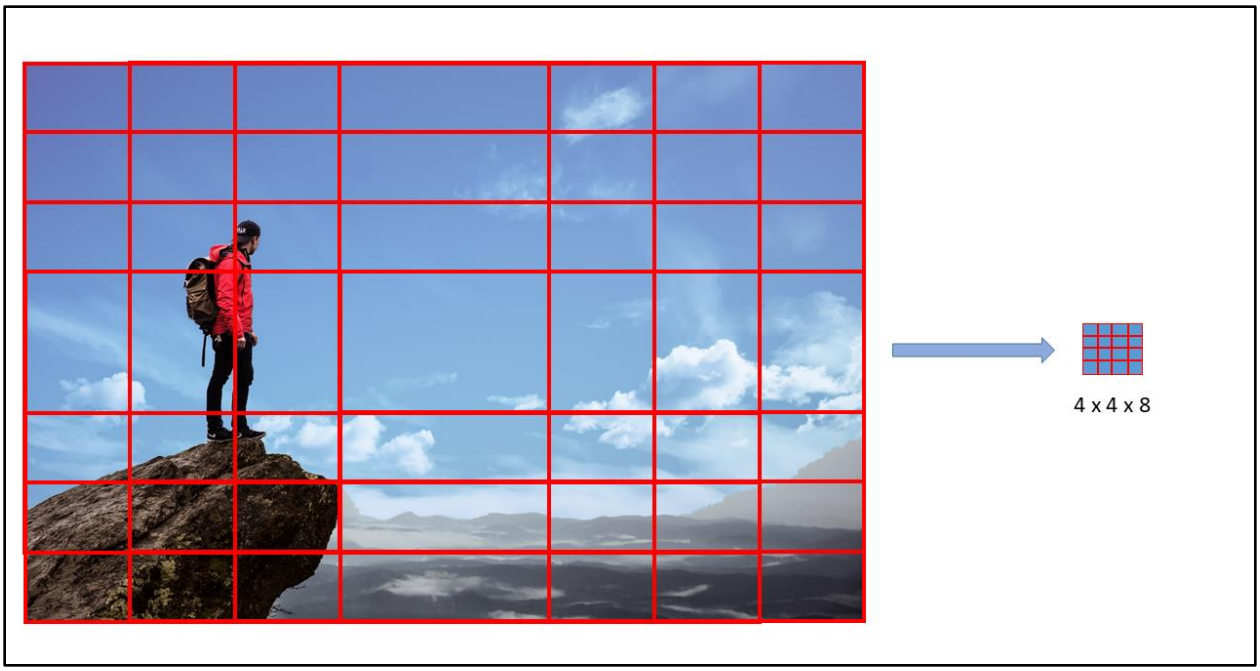


Figure 27: Windows over the full image

This is how we can apply sliding windows with a CNN. This technique makes the whole process much more efficient. However, this technique has a downside: the position of the bounding boxes is not going to be very accurate. The reason is that it is quite unlikely that a given size window and stride will be able to match the objects in the images perfectly. In order to increase the accuracy of the bounding boxes, YOLO uses a grid instead of sliding windows, in addition to two other techniques, known as Intersection Over Union and Non-Maximal Suppression.

The combination of the above techniques is part of the reason the YOLO algorithm works so well. Before diving into how YOLO puts all these techniques together, we will look first at each technique individually.

Steps of YOLO algorithm:

YOLO algorithm uses grids instead of sliding windows, and each cell has an associated vector that tells us:

- If an object in that cell.
- The ‘’class’’ if that object.
- The predicted bounding box (can be larger than the cell) for that object.

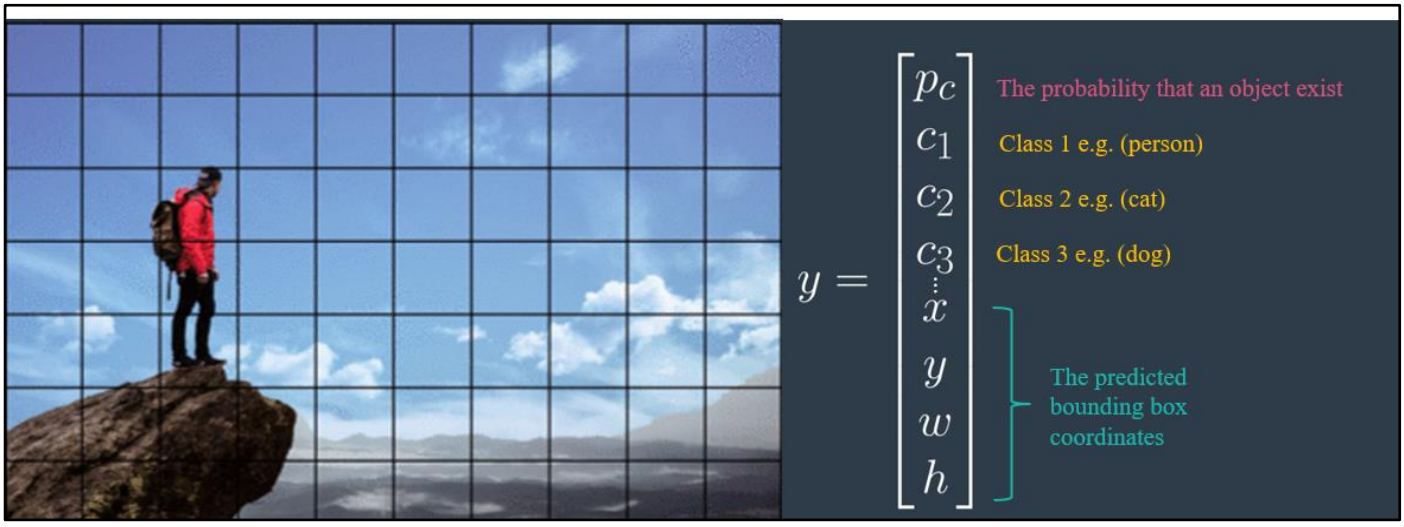


Figure 28: YOLO grid cells vector

In the Yolo algorithm, X and Y determine the coordinates of the Center of the bounding box relative to the grid cell. And W and H determined the width and the height of the box relative to the whole image.

The convention is that the upper left corner of a grid cell has coordinates (0, 0), while the bottom right hand corner has coordinates (1, 1).

So, in this example, the CenterPoint relative to the grid cell coordinate system is X equal to about 0.5 and Y equal to 0.3.

Now, the width of the predicted bounding box W is 0.1 because its width is about 10%, the width of the entire image and the height H is 0.4 because its height is about 40% of the height of the entire image.

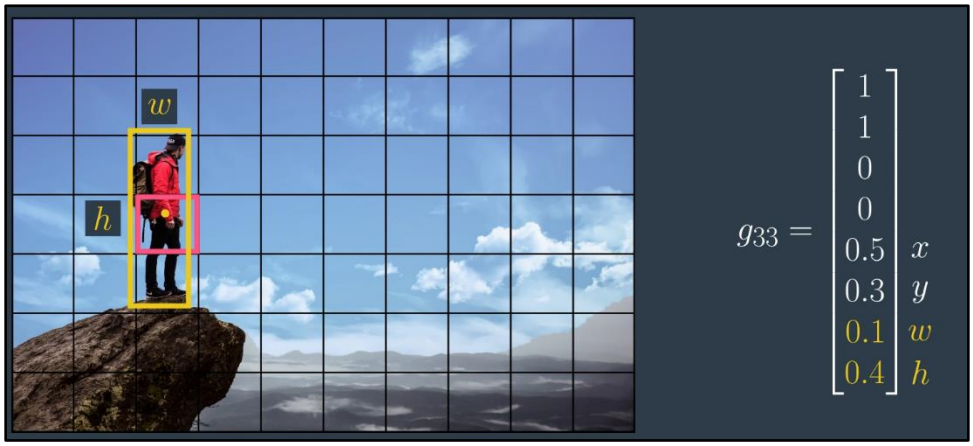


Figure 29: YOLO Coordinates

We can Notice that in the system all of the bounding box coordinate values fall between zero and 1, and the width and height of the bounding box can be bigger than the size of the grid cell.

But when the CNN faced with a new test image, will often produce multiple grid cell vectors that are all trying to detect the same object. This means lots of output vectors that all contain slightly different bounding boxes for the same object.

Figure 31: Test Image

Figure 30: Too many bounding boxes

To solve this problem, we use a technique called nonmaximal suppression, which tries to find the bounding box that best matches an object in an image.

But first we need to define IoU or Intersection over Union which is a technique used in non maximal suppression. To compare how good two different bounding boxes are for a given object.

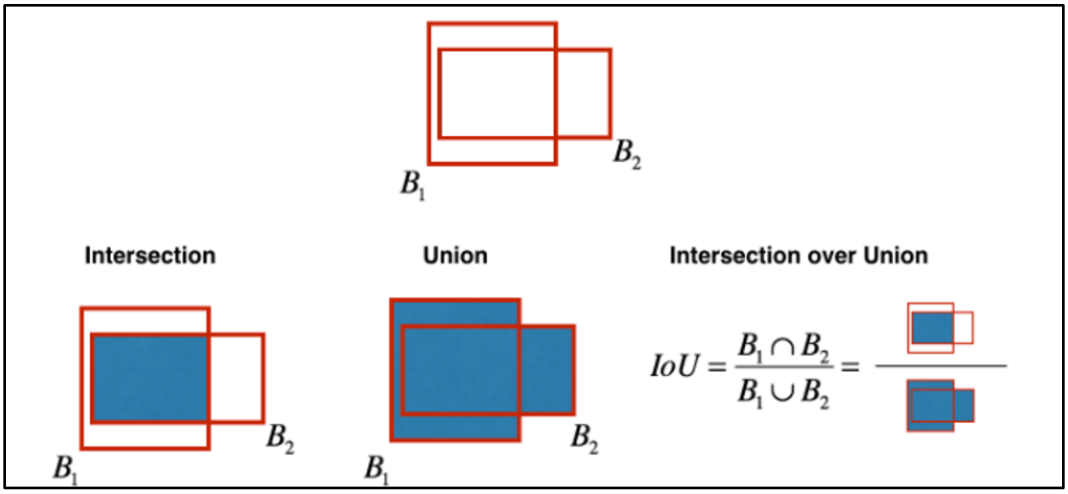


Figure 32: IoU

Non max suppression uses the IOU between two predicted bounding boxes to Select the best bounding box which has high Pc. Pc is a measure of confidence that there has been an object detected, and so a high PC means a high confidence of object detection.

So non maximal suppression selects only the bounding box with the highest PC value. Then removes all the bounding boxes that have a high IOU value when compared to this best bounding box that it just selected. In this way, it gets rid of the boxes that are too similar to the selected bounding box. Then we are left with just one bounding box, which should correspond to the best prediction.

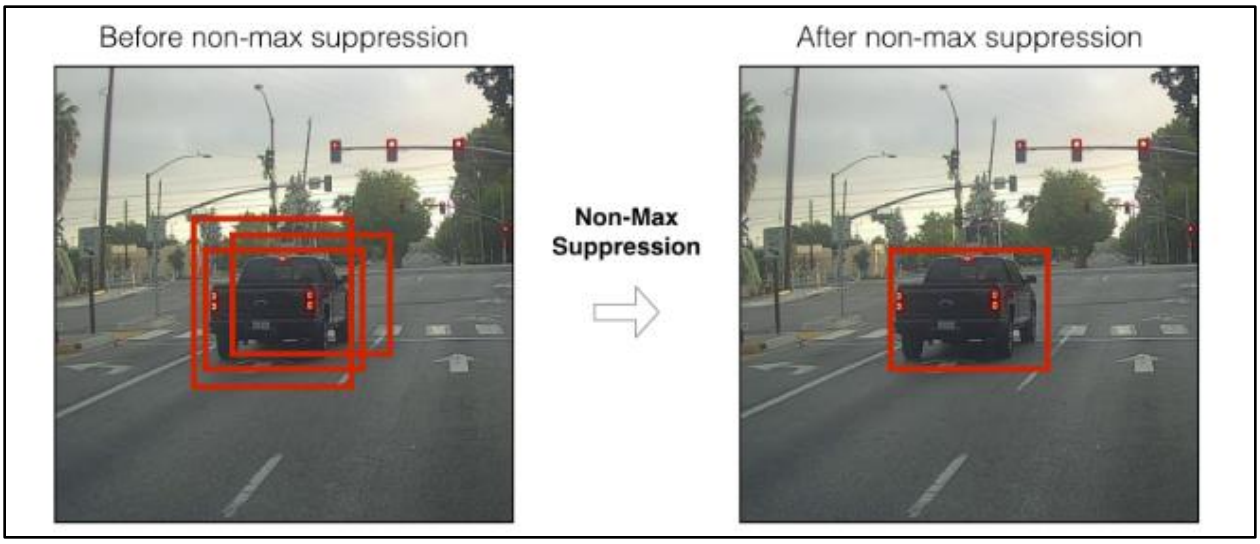


Figure 33: Non max suppression

We can also use anchor boxes to handle the case of overlap, in which one grid cell actually contains the center points of two different objects (Car & Person).

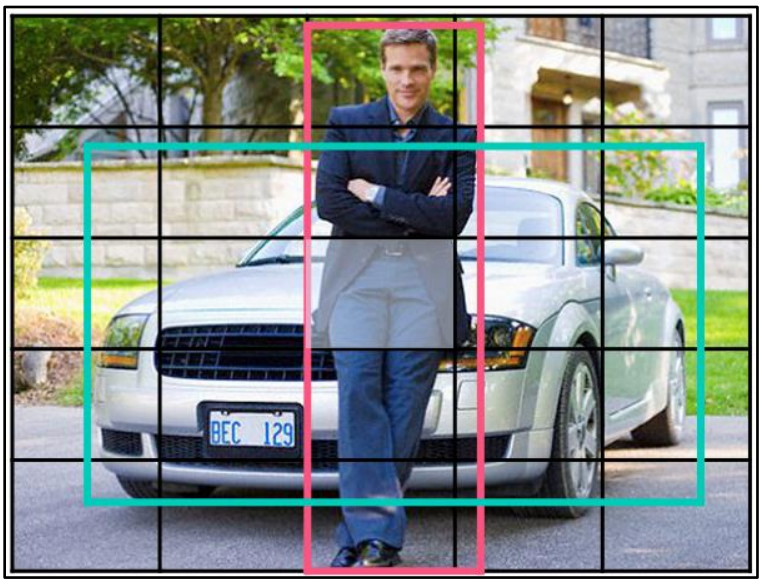


Figure 34: Two overlapping objects

Since the output vector of each grid cell can only have one class, then it will be forced to pick either the car or the person. But by defining anchor boxes, we

can create a longer grid cell vector and associate multiple classes with each grid cell. In practice, we can define as many anchor boxes as we want.

We typically choose anchor boxes so that they nicely fit any object you put in them, and we want to define anchor boxes so that they span the variety of object shapes we want to detect. Anchor boxes have a defined aspect ratio, and they try to detect objects that nicely fit into a box with that ratio.

We can define one anchor box that is roughly the shape of a car. This box will be wider than it is tall, and we'll define another anchor box that can fit a standing prison inside of it, which will be taller than it is wide. We can now modify the output vector of each grid cell to contain both anchor boxes, which have coordinates and class scores. So, the output vector will now contain 16 elements. The first eight elements will correspond to anchor box one, and the last eight will correspond to anchor box two.

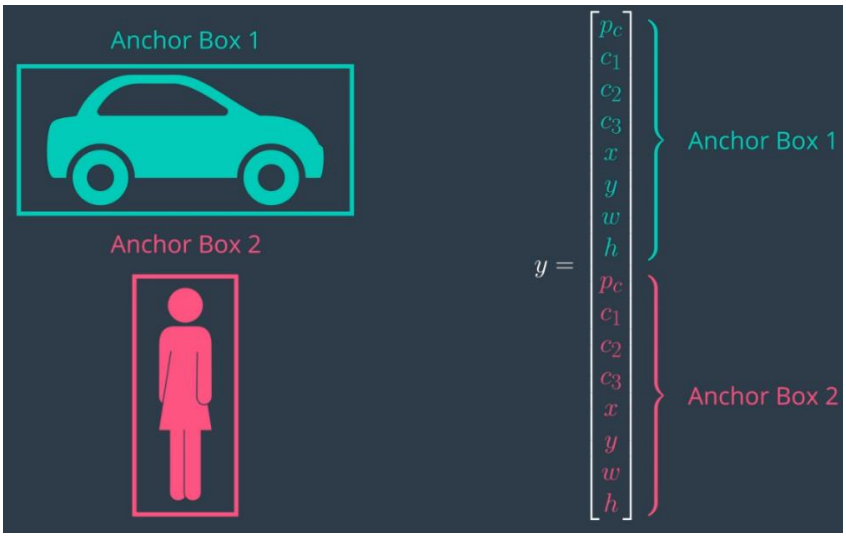


Figure 35: Anchor Boxes

And because the bounding box around the car has a higher IOU with anchor box one, the anchor one elements of our output vector will include the classification and box parameters of the car.

Similarly, since the bounding box around the person has a higher IOU with anchor box two, the anchor two elements will include the parameters of the person.

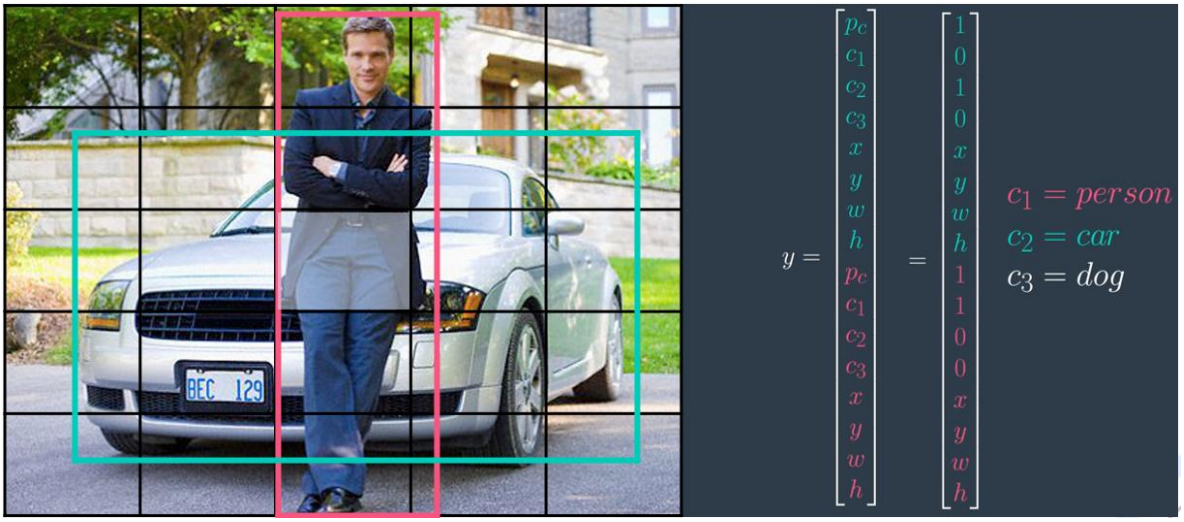


Figure 36: New Vector for two anchor boxes

YOLO Limitations:

Like any object detection methods YOLO algorithm has its limitations.

- Consider the case of two overlapping objects with the same anchor box size, YOLO can associate one object with each type of anchor box.
- If we define two anchor boxes but have three different overlapping objects, then YOLO will be forced to identify only two of the objects.

But the above cases are somewhat rare.

YOLO Versions:

- YOLO was proposed in 2016 by **Joseph Redmon**, a graduate from the University of Washington.
- due to its capability of detecting objects in real-time with a better accuracy, The paper
  describing YOLO won the OpenCV People's Choice Award at the Conference on
  Computer Vision and Pattern Recognition (CVPR) in 2016.

YOLO versions by Joseph Redmon:

1. YOLO v1 in 2016 [16]
2. YOLO v2 in 2017 [17]
3. YOLO v3 in 2018 [18]

YOLO v3

YOLO is based on **Darknet**, which is an open-source neural network framework written in C and CUDA, introduced by Redmon himself.

PyTorch translation for YOLO v3 has been introduced by **Glenn Jocher** of Ultralytics LLC. [19]

Model	Train	Test	mAP	FPS
YOLO	VOC 2007+2012	2007	63.4	45
Fast YOLO	VOC 2007+2012	2007	52.7	155
YOLOv2	VOC 2007+2012	2007	76.8	67
YOLOv2 544x544	VOC 2007+2012	2007	78.6	40
Tiny YOLO	VOC 2007+2012	2007	57.1	207
YOLOv2 608x608	COCO trainval	test-dev	48.1	40
Tiny YOLO	COCO trainval	-	-	200
YOLOv3-320	COCO trainval	test-dev	51.5	45
YOLOv3-416	COCO trainval	test-dev	55.3	35
YOLOv3-608	COCO trainval	test-dev	57.9	20
YOLOv3-tiny	COCO trainval	test-dev	33.1	220
YOLOv3-spp	COCO trainval	test-dev	60.6	20

Figure 37: YOLO models statistics

After yolo v3 and in February 2020, Joseph Redmon, the creator of YOLO announced that he has stopped his research in computer vision, because his works were used in military applications and privacy concerns tasks that eventually became impossible to ignore.

YOLO v4:

The 4th generation of YOLO has been introduced by Alexey Bochkovskiy in a paper titled ‘YOLOv4: Optimal Speed and Accuracy of Object Detection’ in April 2020. [20]

YOLO v4 also based on the Darknet and has been considered the fastest and most accurate real-time model for object detection.

When compared with YOLO v3, the AP and FPS have increased by 10 percent and 12 percent, respectively.

The official YOLO Github account released an updated YOLO v4.

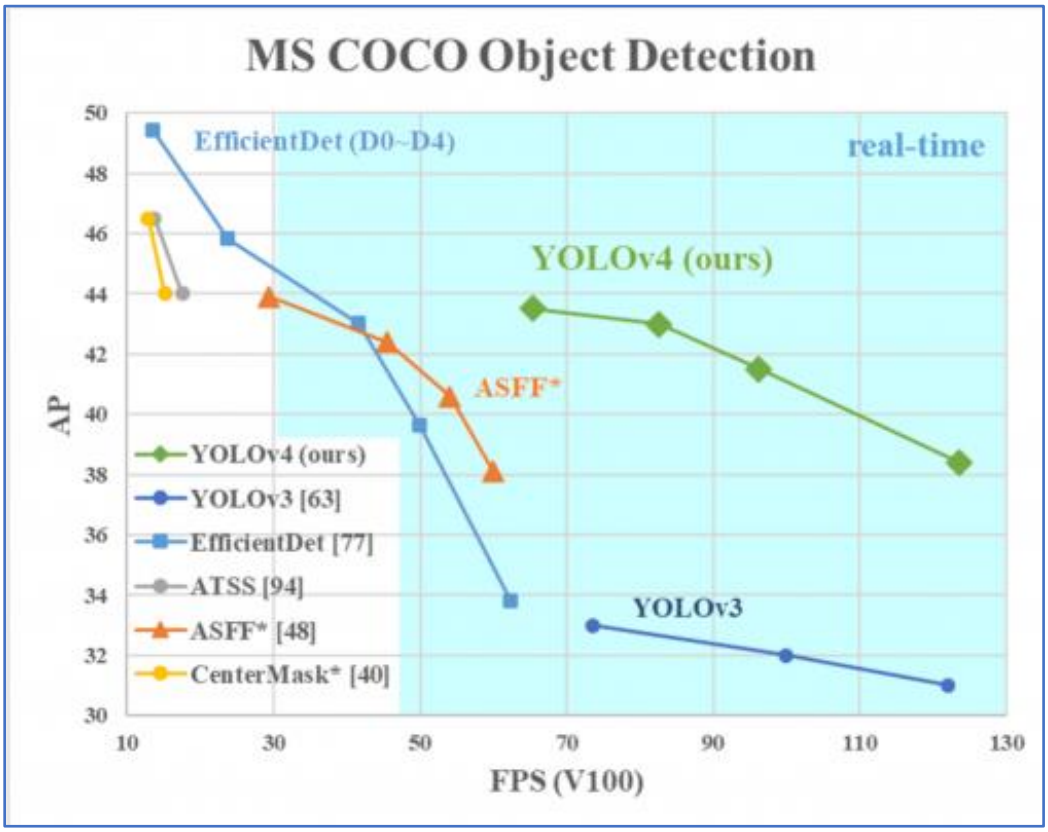


Figure 38: YOLO v4 performance

Other versions of YOLO:

PP YOLO:

PP-YOLO has been introduced in July 2020, via a paper titled "PP-YOLO: An Effective and Efficient Implementation of Object Detector", by Xiang Long et al. It is based on PaddlePaddle (Parallel Distributed Deep Learning), an open-source deep learning platform originally developed by Baidu scientists. [21]

The notable changes include the replacement of Darknet53 backbone of YOLO v3 with a ResNet backbone.

According to the paper, the PP-YOLO can achieve a mAP of 45.2% COCO dataset which exceeds the 43.5% of YOLO v4. When tested on a V100 with batch size = 1, the PP-YOLO can achieve an inference speed of 72.9 FPS, which is also higher than 65 FPS of YOLO v4.

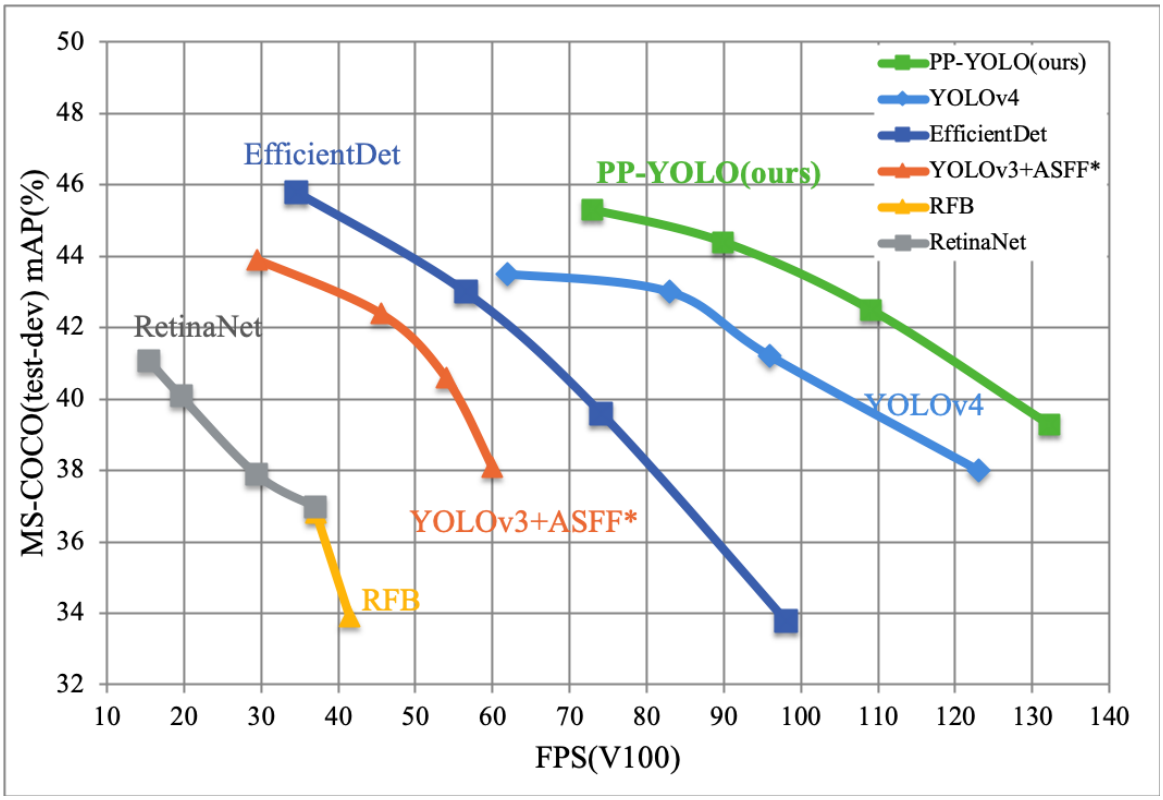


Figure 39: PP-YOLO

YOLO v5:

On June 2020, by Glenn Jocher, who already known among the community for creating the popular PyTorch implementation of YOLO v3.

However, Glenn did not publish a paper to accompany his release, when initially releasing this new version.

Followings are some quotes from that blog post by Joseph Nelson and Jacob Solawetz.

“Running a Tesla P100, we saw inference times up to 0.007 seconds per image, meaning 140 frames per second (FPS)! By contrast, YOLO v4 achieved 50 FPS after having been converted to the same Ultralytics PyTorch library.”

“YOLO v5 is small. Specifically, a weights file for YOLO v5 is 27 megabytes. Our weights file for YOLO v4 (with Darknet architecture) is 244 megabytes. YOLO v5 is nearly 90 percent smaller than YOLO v4.”.

Traffic Light Classification:

After detecting the traffic lights using object detection techniques, we can use the properties of the HSV color palette to classify traffic light colors.

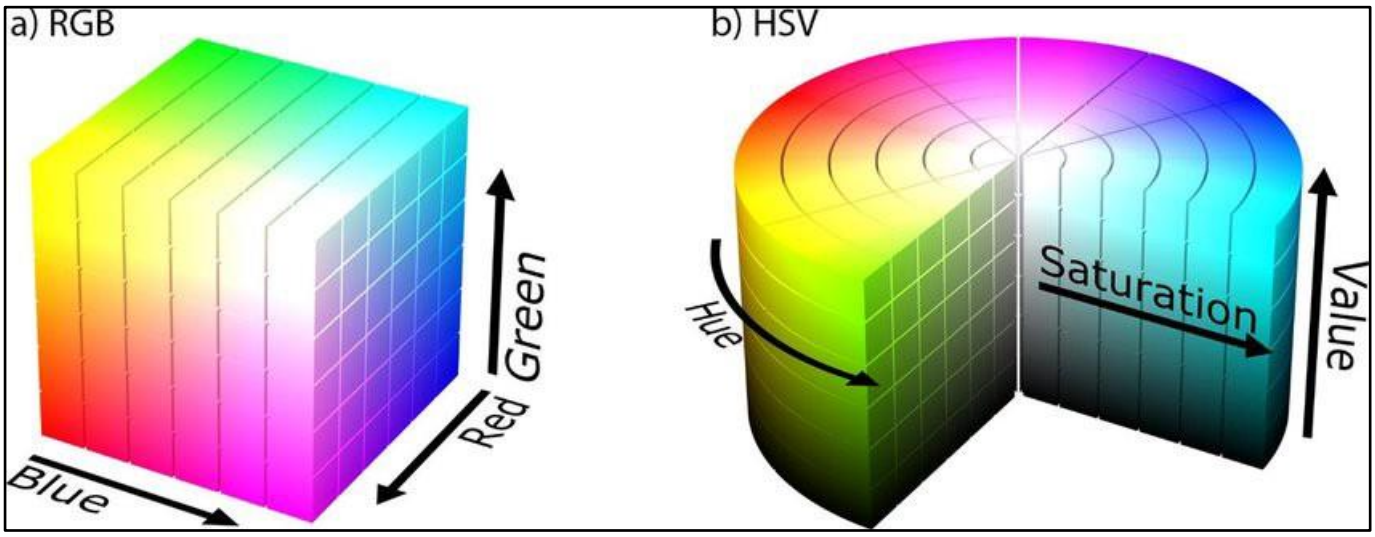


Figure 40: RGB & HSV

We transform the image from RGB to HSV, then define three filters (green, yellow, red) to filter out the different colors.

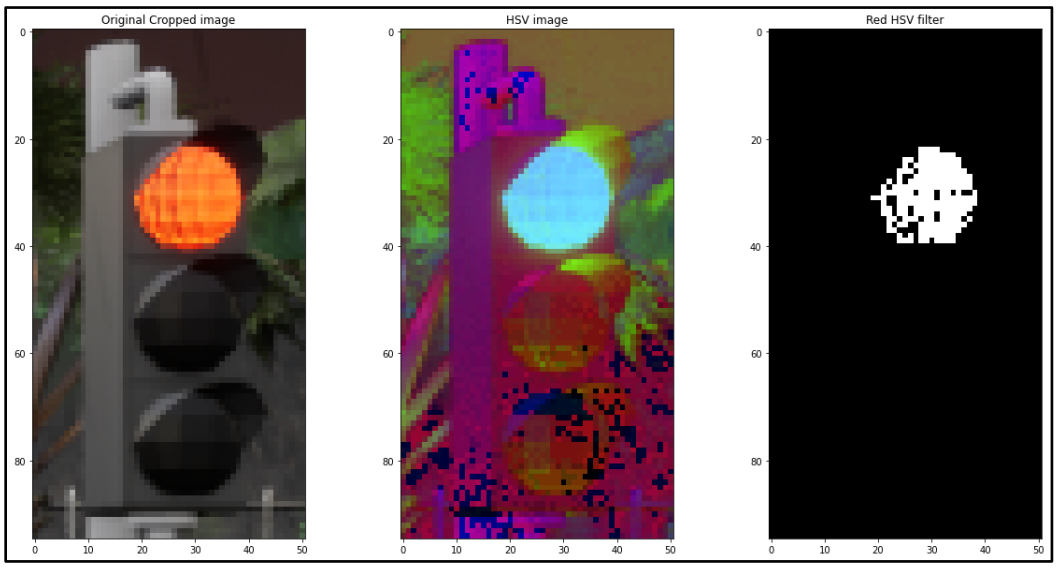


Figure 41: From RGB to HSV

Then we can sum the values of the filter images (white color equals 255), and the filter with the max value indicates the color of the traffic light

Transfer learning:

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems

In our project we will collect our own data from Carla simulator (about 3k images) and then apply transfer learning in order to create a custom yolov4 model to meet our system specifications.

Transfer learning process made by replace the output layer in the network by our output layer based on output classes of our system as follows:

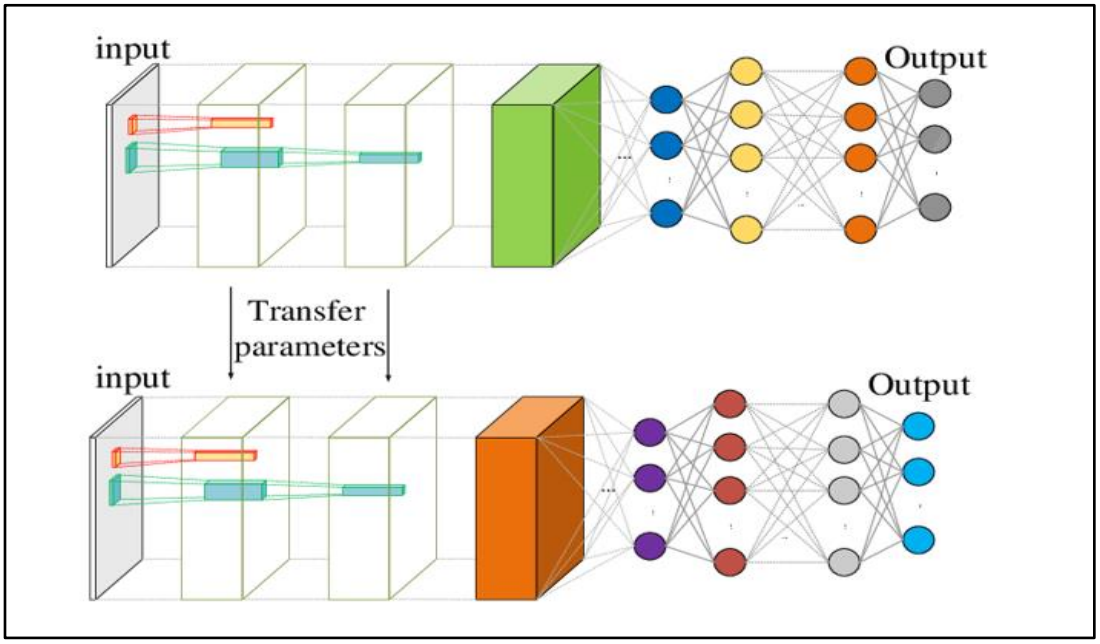


Figure 42: Transfer learning

03 Depth estimation

After detecting objects and determine their classes we need to measure the distance between the autonomous vehicle and the detected object, so depth estimation is essential to determine the object estate.

Hardware:

Camera, LiDAR, Radar, Ultra-sonic are the most common sensors used to determine the distance.

Figure 43: Depth estimation hardware tools

Each sensor has its advantages and disadvantages. A detailed comparison is demonstrated in the table below.

	LiDAR	Camera	Radar	Ultra-sonic
Range	Medium	-	Short	Short
Cost	Heigh	Low	Medium	Low
Size	Large	Small	Small	Small
Color	-	✔	-	-
Illumination	-	✔	-	-
Weather	✔	✔	-	-

1. **<u>Monocular Camera:</u>**

Cameras can sense color and are passive, i.e. they do not emit any signal for measurements. Sensing color is extremely important for tasks such as traffic light recognition. Furthermore, 2D computer vision is an established field with remarkable state-of-the-art algorithms. Moreover, a passive sensor does not interfere with other systems since it does not emit any signals. However, cameras have certain shortcomings. Illumination conditions affect their performance drastically, and depth information is difficult to obtain from a single camera. There are promising studies [23] to improve monocular camera-based depth perception.

2. Radar:

Radar, lidar, and ultrasonic sensors are very useful in covering the shortcomings of cameras. Depth information, i.e. distance to objects, can be measured effectively to retrieve 3D information with these sensors, and they are not affected by illumination conditions. However, they are active sensors. Radars emit radio waves that bounce back from objects and measure the time of each bounce. Emissions from active sensors can interfere with other systems. Radar is a well-established technology that is both lightweight and cost-effective. For example, radars can fit inside side mirrors.

3. LiDAR:

Lidar operates with a similar principle to that of radar but it emits infrared light waves instead of radio waves. It has much higher accuracy than radar under 200 meters. Weather conditions such as fog or snow have a negative impact on the performance of lidar. Another aspect is the sensor size: smaller sensors are preferred on the vehicle because of limited space and aerodynamic restraints and lidars are generally larger than radars.

When considering the medium range, LiDAR and Camera are the best suitable sensors. But due to the disadvantage of LiDAR cost and size, we realize our system using the Camera and preprocess the image to determine the distance.

Monocular Camera Depth Estimation:

1. Supervised Learning Method:

It involves a convolution neural network trained to predict the depth map of a colored image from a monocular camera [24].

It mostly generates a depth map scaled which tells how objects far from each other in scale view. but without the actual distance.

Figure 44: Depth map

2. Perspective Transformation Method:

The wrapped image generated by transforming the image to the bird-eye view using the Perspective Transformation. Then, on the wrapped image there is a direct correlation between the pixel position and distance in meters.

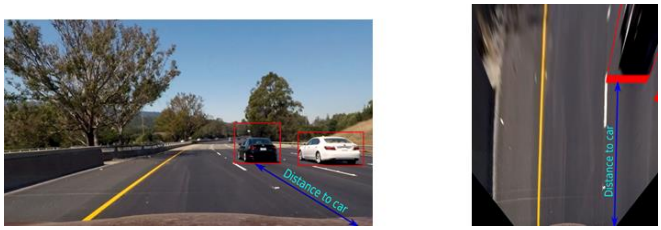


Figure 45: bird-eye view

3. Using Bounding Box Width:

The smaller the bounding box, the further the object is and the bigger the bounding box, the closer the object is.

If we used that terminology we can approximate the distance of the object using linear equations but, the distance wouldn't be accurate and would be subject to errors.

Stereo Camera Depth Estimation:

The stereo camera is the type of camera with two or more image sensors. This allows the camera to simulate human binocular vision and therefore gives it the ability to perceive depth.

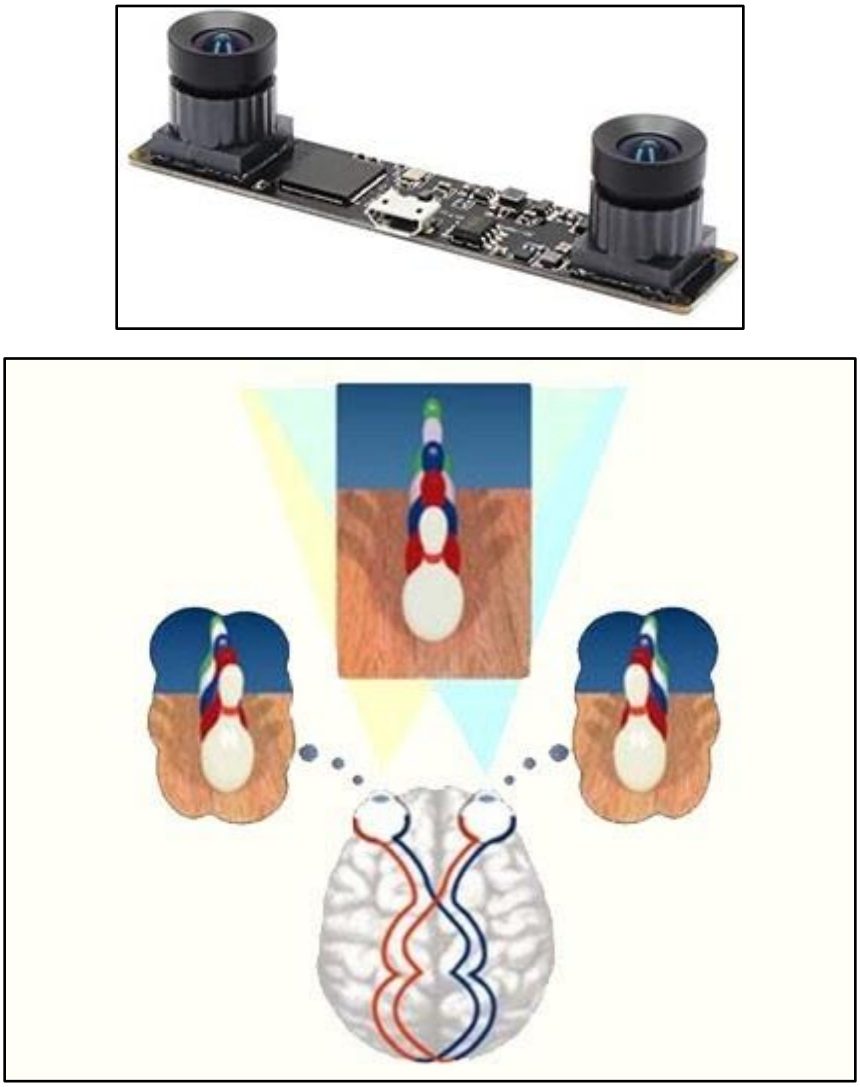


Figure 46: Stereo camera

The stereo vision system is one of the popular computer vision techniques, The idea here is to use the parallax error to our advantage.

A single scene is recorded from two different viewing angles, and depth is estimated from the measure of parallax error. This technique is more than a century old and has proven useful in many applications. This field has made a lot of researchers and mathematicians to devise novel algorithms for the accurate output of the stereo systems. This system is particularly useful in the field of robotics. It provides them with the 3D understanding of the scene by giving them estimated object depths.

Stereo Depth Estimation Model

The two-camera must be identical as possible. As demonstrated in the below image, left camera (CL), right camera (Cr), focal length (f), the distance between two-camera called base line (b).

The point in the 3D coordinate (O) is translated to the image coordinate using the Rotation and Translation matrix [R|T].

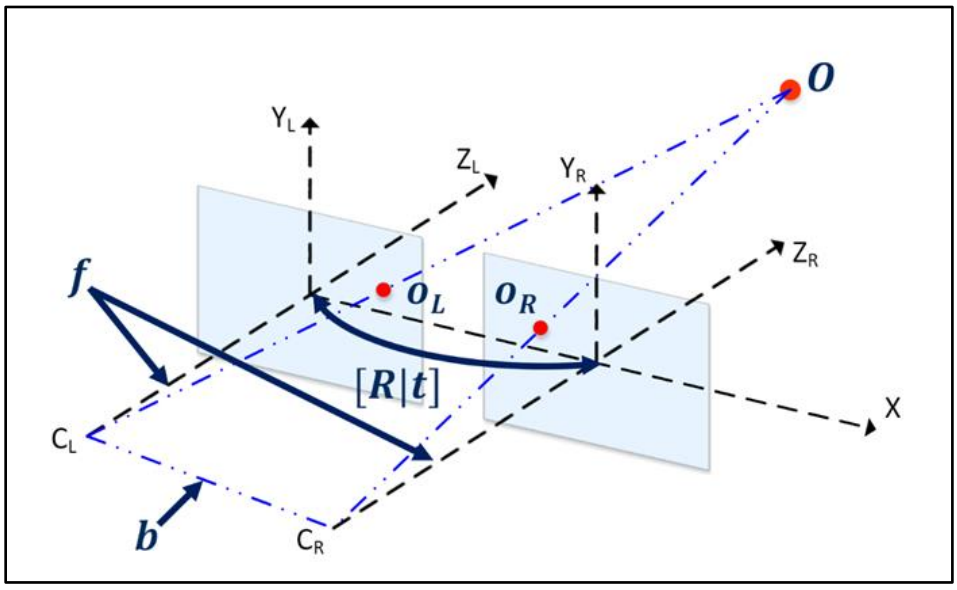


Figure 47: Stereo depth estimation model

If the view transformed to the bird-eye view as demonstrated in the image below, the projection of point O in X-axis will be XL for the left camera and XR for the right camera.

Then we can compute a relation for the point depth (Z) as described in the image.

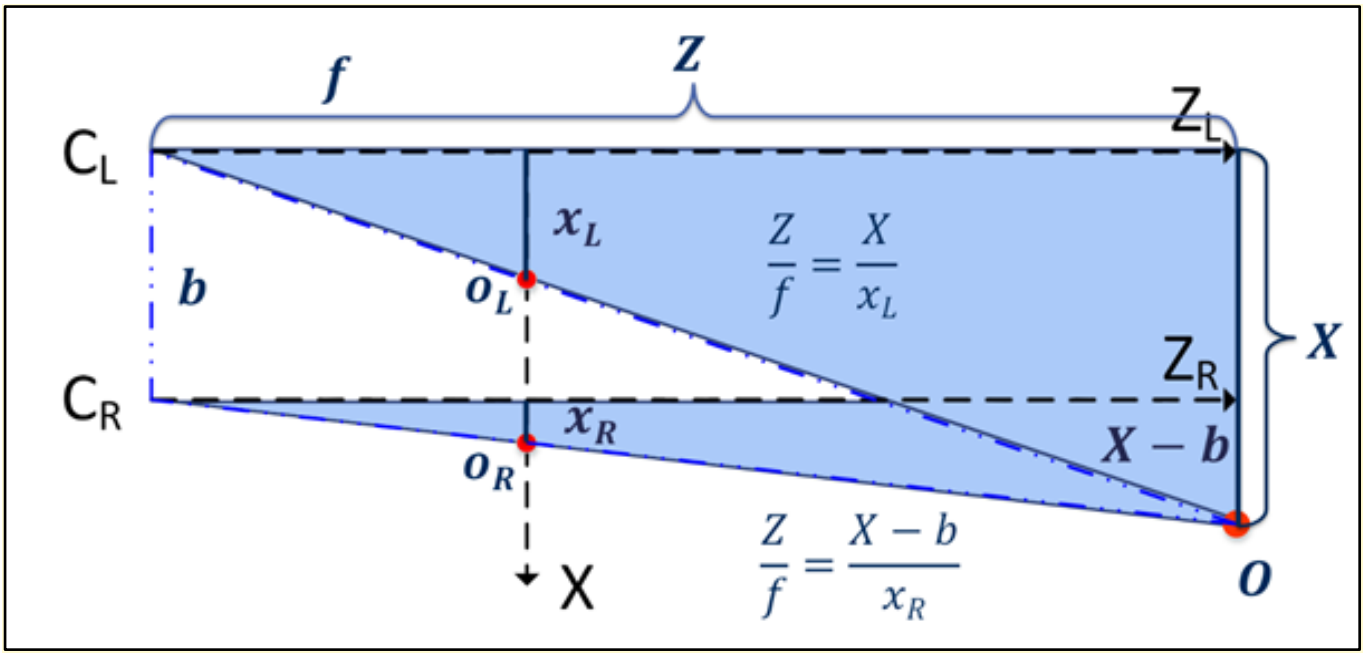


Figure 48: Depth point

The difference between the XL and XR is called a Disparity. When applying the disparity computation for every pixel we compute a Disparity Map which has the same size as the original image.

Disparity:

$$d = x_L - x_R$$

where $x_L = u_L - u_0$

$x_R = u_R - u_0$

$y_L = v_L - v_0$

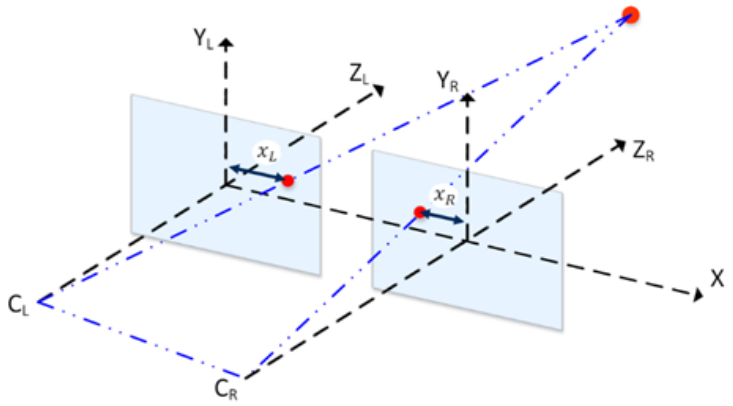


Figure 49: Disparity

Given the disparity map of every pixel and the provided relation between the focal length, baseline we can then compute the Depth Map (Z) of every point, or even compute the 3D coordinates of every point if we make the stereo-camera calibration.

Main stereo relations:

$$\frac{Z}{f} = \frac{X}{x_L} \rightarrow ZxL = fX$$

$$\frac{Z}{f} = \frac{X - b}{x_r} \rightarrow ZxR = fX - fb$$

$$Zx_R = Zx_L - fb$$

$$Z = \frac{fb}{d}$$

From above: $X = \frac{Zx_L}{f}$, $Y = \frac{Zy_L}{f}$

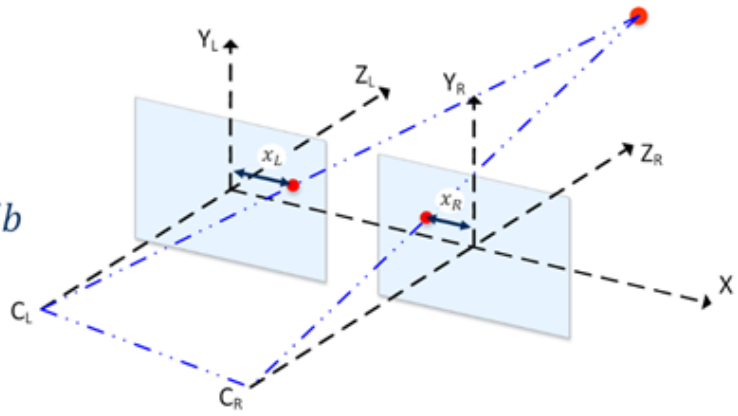


Figure 50: Stereo relations

Disparity Map Calculation:

The disparity map is the difference in image location of the same 3D point under perspective to two different cameras. Correspond pixels in the left image to those in the right image to find matches. There are two methods used to matching the two points:

1. Brute Force Solution: is to match the two points is the pick the left image and compare it to every pixel in the right image which called exhaustive search and consumes a lot of time as searching is 2D space.

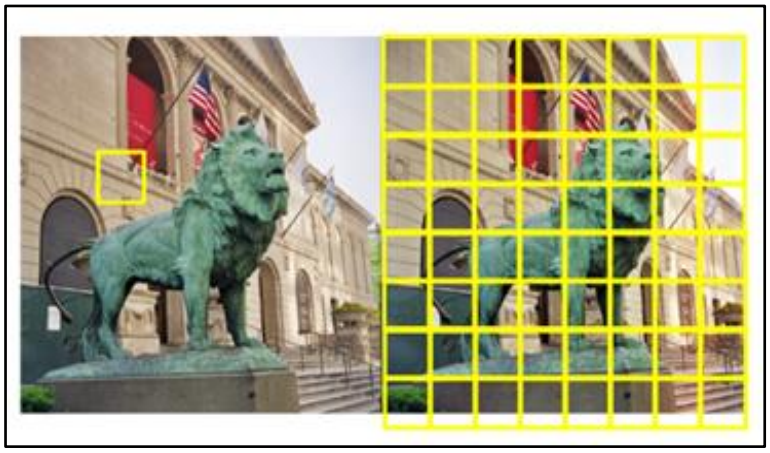


Figure 51: Brute force solution

2. Epipolar Solution - stereo camera must be posed parallel, so the projected point in the two cameras will be in the same line which called Epipolar line. This reduces search time complexity from 2D to 1D.

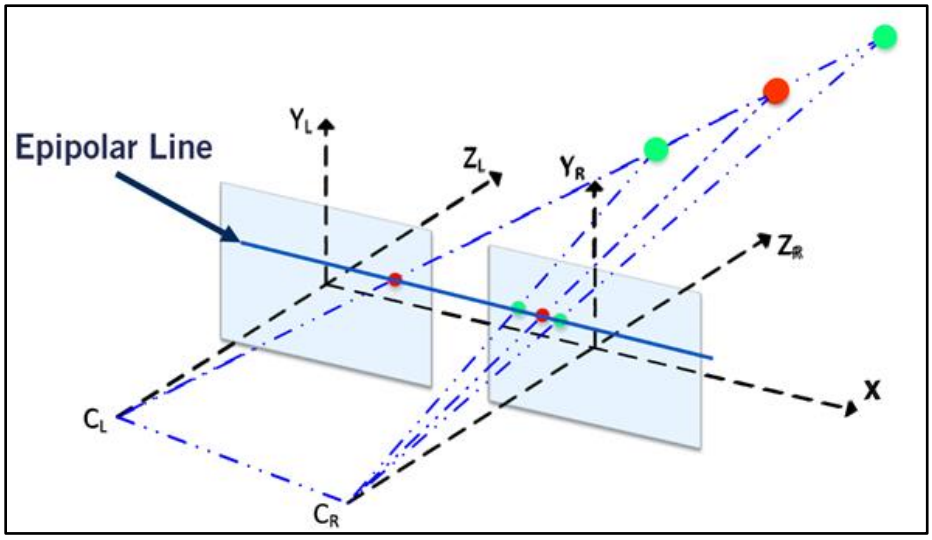


Figure 52: Epipolar solution

Stereo Algorithm steps:

Given two images capture by stereo-camera:

1. Take each pixel in this line of the left image.
2. Compare these left image pixels to every pixel in the right image on the same epipolar line
3. Pick the pixel that has the minimum cost
4. Compute disparity
5. Compute the real-world coordinates X, Y, and Z for each pixel.

The two images below captured from two cameras stand as stereo-camera and after computing the left disparity map.

Figure 53: Left & right camera image

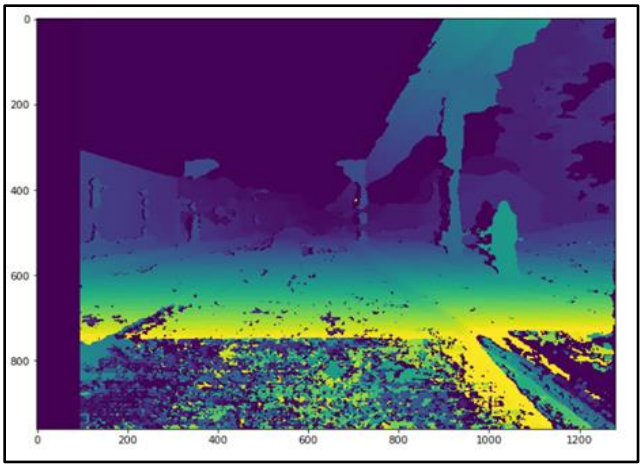


Figure 54: Disparity map

3. **Stereo Matching using Supervised Deep Learning –** Recent work has shown that disparity map estimation from a stereo pair of images can be formulated as a supervised learning task to be resolved with convolutional neural networks (CNNs). However, most architecture relies on Siamese networks [25], lacking to exploit the context information. Pyramid Stereo Matching Network (PSMNet) overcomes this problem [26].

Pyramid Stereo Matching Network (PSMNet)

PSMNet extends pixel-level features to region-level features with different scales of receptive fields; the resultant combined global and local feature clues are used to form the cost volume for reliable disparity estimation. The network contributions are listed below:

- End-to-End learning network for stereo matching without any post-processing.
- Introduced pyramid pooling module for incorporating global context information into image features.
- Introduced a stacked hourglass 3D CNN to extend the regional support of context information in cost volume.
- Achieved state-of-the-art performance on KITTI dataset.

PSMNet Architecture

It consists of a SPP module for effective incorporation of global context and a stacked hourglass module for cost volume regularization. The architecture is demonstrated in the figure 55.

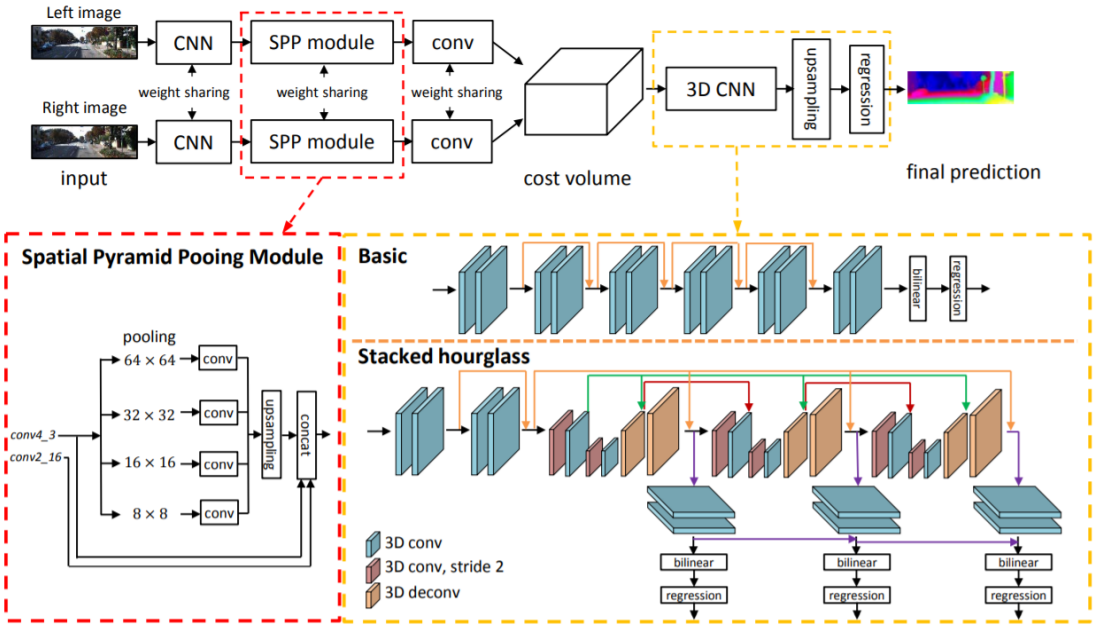


Figure 55: PSMNet Architecture

Spatial Pyramid Pooling Module – SPP was designed to remove the fixed-size constraint of CNN. In the current network SPP is designed of four fixed-size average pooling blocks: 64 × 64, 32 × 32, 16 × 16, and 8×8. Further operations including 1 × 1 convolution and upsampling.

Cost Volume – Rather than using a distance metric, it concatenates the left and right features to learn matching cost estimation using deep network which resulting in a 4D volume.

3D CNN – In order to learn more context information, a stacked hourglass (encoder-decoder) architecture is used, consisting of repeated top down/bottom-up processing in conjunction with intermediate supervision. The stacked hourglass architecture has three main hourglass networks, each of which generates a disparity map.

the stacked hourglass architecture has three outputs and losses (Loss 1, Loss 2, and Loss 3). During the training phase, the total loss is calculated as the weighted summation of the three losses. During the testing phase, the final disparity map is the last of three outputs.

PSMNet Experiment Details

The method is evaluated on three stereo datasets: Scene Flow, KITTI 2015, and KITTI 2012. However, KITTI will be considered the most suitable dataset in our problem.

- **KITTI 2015**: a real-world dataset with street views from a driving car. It contains 200 training stereo image pairs with sparse ground-truth disparities obtained using LiDAR and another 200 testing image pairs without ground-truth disparities. Image size is H = 376 and W = 1240. The whole training data is divided into a training set (80%) and a validation set (20%).

Results on Leaderboard

The best setting of PSMNet yielded a 1.83% error rate on the KITTI 2015 validation set. And achieves a state-of-art performance on KITTI dataset.

Network setting						KITTI 2015	Scene Flow
dilated conv	pyramid pooling size				stacked hourglass	Val Err (%)	End Point Err
	64×64	32×32	16×16	8×8			
						2.43	1.43
√						2.16	1.56
	√	√	√	√		2.47	1.40
√	√					2.17	1.30
√	√	√	√	√		2.09	1.28
√	√	√	√	√	√	1.98	**1.09**
√*	√	√	√	√	√	**1.83**	1.12

Figure 56: Evaluation of PSMNet with different settings

The Two images below are captured from a stereo camera and the result is predicted from PSMNet.

Figure 57: Left Image

Figure 58: Right Image

Figure 59: Right Image

04 Lane detection

History of Lane Detection:

Most traditional methods of traffic line detection extract low-level traffic line feature using various hand-craft features like color, or edges. These low-level features can be combined using a Hough transform, or Kalman filter. These methods are simple and can be adapted to various environments without significant modification. Still, the performance of these methods depends on the condition of the testing environment such as lighting and occlusion.

Figure 60: Lane detection

Improvement:

Deep learning methods show outstanding performance for complex scenes. Using convolution neural network CNN has an outstanding performance in almost all computer vision tasks.

Semantic Segmentation:

Most people in the deep learning and computer vision communities understand what image classification is: we want our model to tell us what single object or scene is present in the image. Classification is very coarse and high-level.

Many are also familiar with object detection, where we try to *locate and classify multiple objects* within the image, by drawing bounding boxes around them and then classifying what's in the box. Detection is mid-level, where we have some pretty useful and detailed information, but it's still a bit rough since we're only drawing bounding boxes and don't really get an accurate idea of object shape.

Semantic Segmentation is the most informative of these three, where we wish to classify *each and every pixel in the image*, just like you see in the gif above! Over the past few years, this has been done entirely with deep learning.

Limitations:

- Although semantic segmentation methods can predict lines that have complex shapes, during training and testing they require pixel-level labeled data and post-processing to extract exact points on lines
- Not all pixels are required to recognize traffic lines; an exact line can be predicted from a few key points.

Due to the limitation of semantic segmentation in lane detection, we tend to use another method which is Key point estimation

Key point estimation:

Key points estimation techniques predict from input images certain important points called key points. Human pose estimation is a major research topic in the key points estimation area. Stacked hourglass networks consist of several hourglass modules that are trained simultaneously. The hourglass module can transfer various scales' information to deeper layers, helping the whole network obtain both global and local features. Because of this property, an hourglass network is frequently utilized to detect centers or corners of objects in the object detection area. Not only network architecture or loss function but also refinement methods adapted to existing networks are developed for keypoint estimation.

Figure 61: Key point estimation

For the purpose of lane detection, Key Points Estimation and Point Instance Segmentation Approach for Lane Detection is used in the project which has PINet architecture.

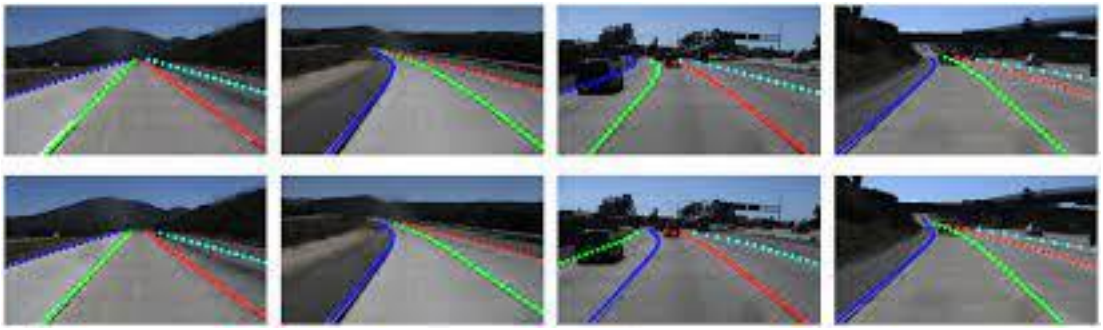


Figure 62: Lane detection key point estimation

PINet Architecture [27]:

The figure below shows the proposed framework for traffic line detection. It has three output branches and predicts the exact location and instance features of points on traffic lines. These are the primary contributions of this study:

- Using the key points estimation approach, A novel method for traffic line detection is purposed. It produces a more compact size prediction output than those of other semantic segmentation-based methods.
- The framework consists of several hourglass modules, and so we can obtain various models that have different sizes by simple clipping because each hourglass module is trained simultaneously using the same loss function.

- The proposed method can be applied to various scenes that include any orientation of traffic lines, such as vertical or horizontal traffic lines, and arbitrary numbers of traffic lines.
- The proposed method has lower false positives and the noteworthy accuracy performance. It guarantees the stability of the autonomous driving car.

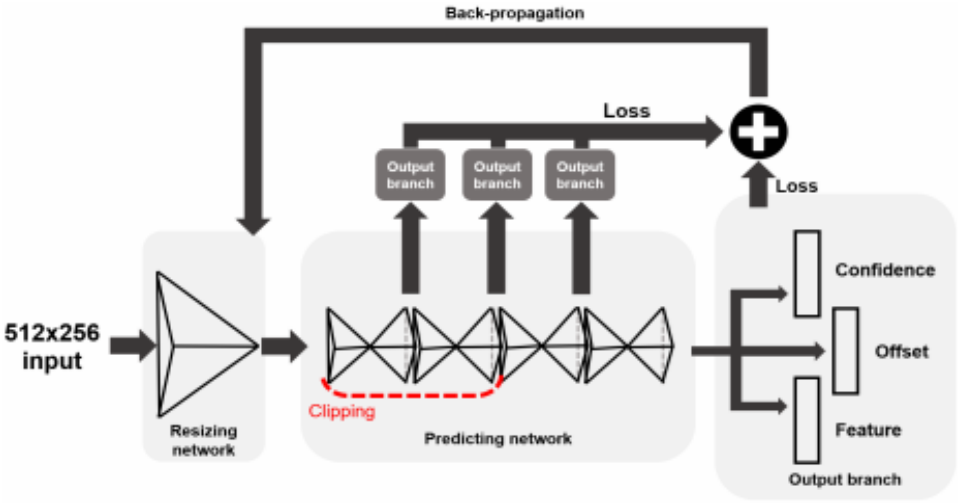


Fig. 2. Proposed framework with three main parts. 512 × 256 size input data is compressed by the resizing network; the compressed input is fed to the predicting network, which includes four hourglass modules. Three output branches are applied at the ends of each hourglass block; they predict confidence, offset, and embedding feature. The loss function can be calculated from the outputs of each hourglass block. By clipping several hourglass modules, required computing resources can be adjusted.

Figure 63: PINet architecture

Resizing Network:

The resizing network reduces the input image's size to save memory and inference time. First, the input RGB image size is 512×256. This network consists of three convolution layers. All convolution layers are applied with filter size 3×3, stride 2, and padding size 1. Prelu and batch normalization are utilized after each convolution layer. Finally, this network generates resized output with 64 × 32 size. Table below shows details of the constituent layers.

DETAILS OF RESIZING NETWORK

Layer	Size/Stride	Output size
Input data		3*512*256
Conv+Prelu+bn	3/2	32*256*128
Conv+Prelu+bn	3/2	64*128*64
Conv+Prelu+bn	3/2	128*64*32

Figure 64: resizing network

Predicting Network:

The resizing network output is fed to the prediction part, which will be described in this section. This part predicts the exact points on the traffic lines and the embedding features for instance segmentation.

network consists of several hourglass modules, each including an encoder, decoder, and three output branches. Some skip-connections transfer the information of the various scales to deeper layers. Each colored block is a

bottle-neck module. There are three kinds of bottle-neck: same, down, and up bottle-necks. The same bottle-neck generates output that has the same size as the input. The down bottle-neck is applied for down-sampling in the encoder; the first layer of the down bottle-neck is replaced by a convolution layer with filter size 3, stride 2, and padding 1. The transposed convolution layer with filter size 3, stride 2, and padding 1 is applied for the up bottle-neck in the up-sampling layers.

Each output branch has three convolution layers and generates a 64 × 32 grid. Confidence values about key point existence, offset, and embedding feature of each cell in the output grid are predicted by the output branches. Table below shows details of the predicting network.

DETAILS OF PREDICTING NETWORK

	Layer	Size/Stride	Output size
	Input data		128*64*32
Encoder	Bottle-neck(down)		128*32*16
	Bottle-neck(down)		128*16*8
	Bottle-neck(down)		128*8*4
	Bottle-neck(down)		128*4*2
	Bottle-neck		128*4*2
(Distillation layer)	Bottle-neck		128*4*2
	Bottle-neck		128*4*2
	Bottle-neck		128*4*2
Decoder	Bottle-neck(up)		128*8*4
	Bottle-neck(up)		128*16*8
	Bottle-neck(up)		128*32*16
	Bottle-neck(up)		128*64*32
Output branch	Conv+Prelu+bn	3/1	64*64*32
	Conv+Prelu+bn	3/1	32*64*32
	Conv	1/1	C*64*32

Figure 65: Predicting network

PINet Dataset:

PINet, is trained on both TuSimple and CULane. Table below summarizes information of the two datasets. TuSimple is relatively simpler than CULane because the TuSimple dataset consists of only the highway environment and fewer obstacles. We use the official evaluation source codes to evaluate PINet.

DATASET SUMMARY

Dataset	Train	Test	Resolution	Type
TuSimple	3,626	2,782	1280×720	highway
CULane	88,880	34,680	1640×5900	urban, rual, highway, various light condition and weather

Figure 66: PINet dataset

8-

Module 2 | Planning

01 Introduction

In Planning, we have two sub tasks, choosing best path from source to destination and studying behavior of obstacles to make decisions. Thus, implementing the brain of a self-driving car.

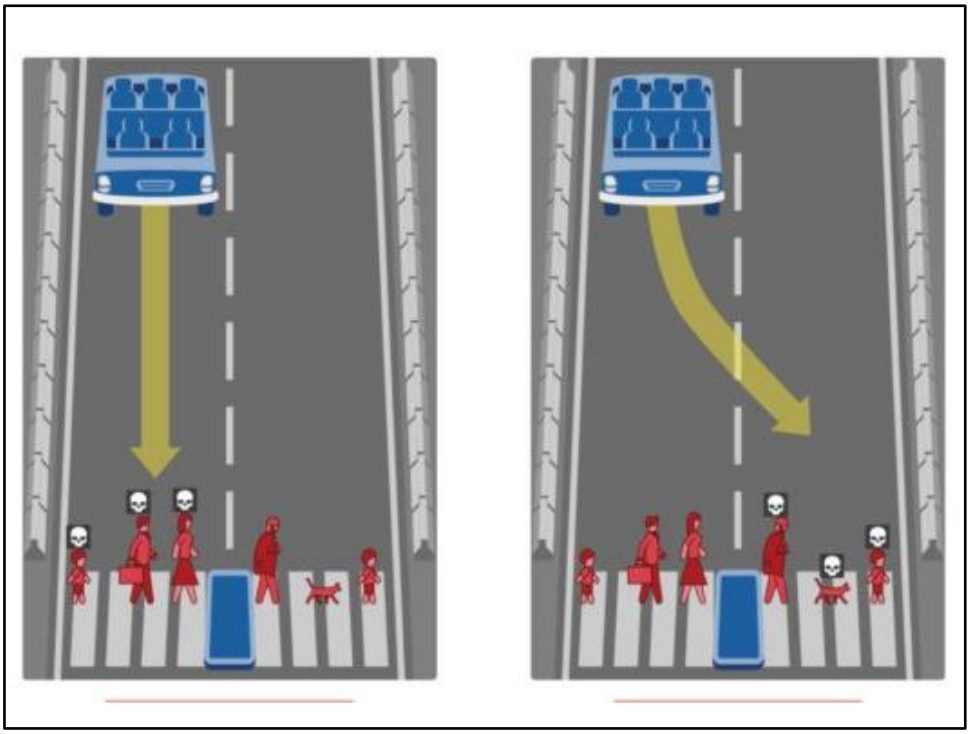


Figure 67: Planning decision

This image is a popular moral problem where we must choose who to sacrifice if we have no brakes. Although unlikely, this topic makes us wonder how a self-driving car would do if it had to take its own decisions.

02 Route planning

First task of planning module is to determine best possible path from source to the destination using a popular algorithm e.g. A*.

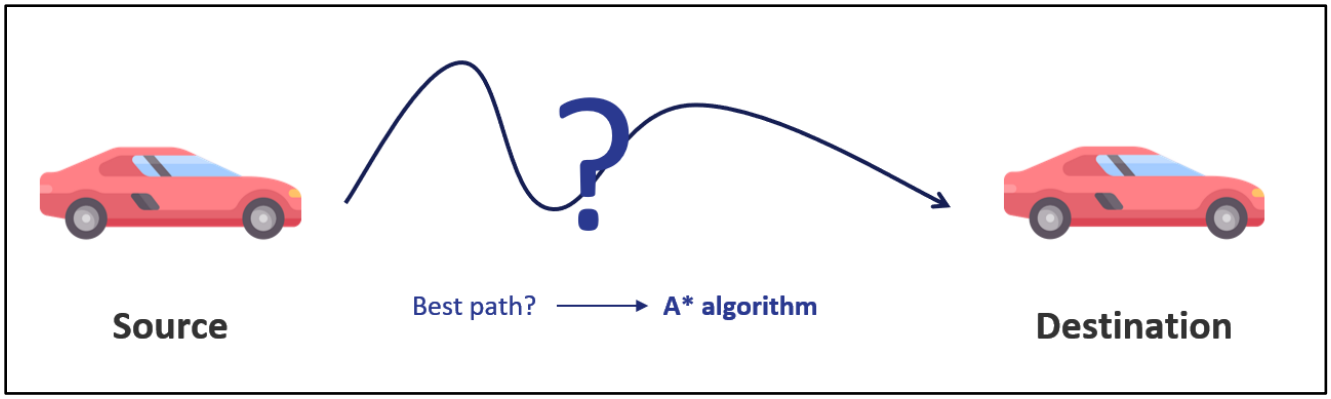


Figure 68: Route planning

A* algorithm[29]:

In the very popular **A* (A-star) algorithm**, we only explore part of the map using a heuristic function. At each point of the map, we indicate the distance to the objective. Rather than systematically exploring every possible path, **A*** chooses to explore only the paths that bring us closer to the goal**.**

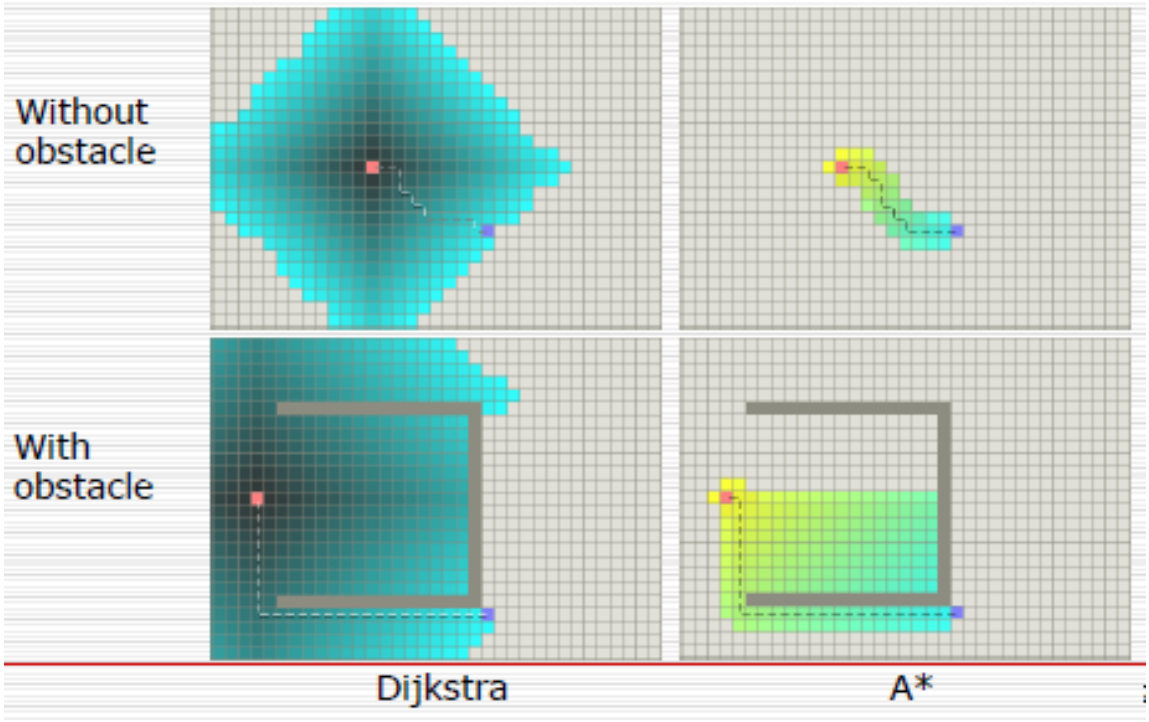


Figure 69: Dijkstra vs A[28]*

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes n.

where *n* is the next node on the path, *g*(*n*) is the cost of the path from the start node to *n*, and *h*(*n*) is a heuristic function that estimates the cost of the cheapest path from *n* to the goal. A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended. The heuristic function is problem-specific. If the heuristic function is admissible, meaning that it never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal.

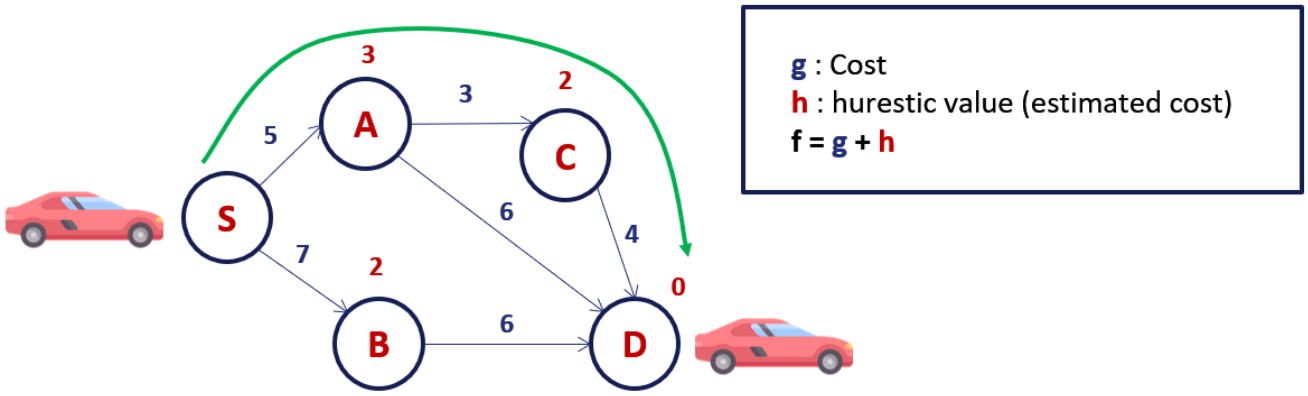


Figure 70: A example*

Finally, here's an example using A* in CARLA simulator:

Figure 71: A result in CARLA*

03 Behavior planning

Behavior planning is the brain of a self-driving car. It is the module that tries to replicate the thinking and decision-making we humans do while driving — read the map, analyze our environment (other vehicles and pedestrians), and decide the optimal action based on safety, speed, and traffic rules.

How is behavior Planning done?

STEP 1: Analyze the sensor data:

From the simulator, we get data about other vehicles. This data includes the car_id, car_position(x and y), car_velocity (vx and vy), car_s (distance along the lane) and car_d (distance along the width of the lane).
We took the data and divided all cars in the simulator into either my own lane, my left lane, or my right lane if those lanes exist.
Then we calculate the important measures we needed for making decisions. These were

- In my lane — distance to the closest car in front of me and the velocity of this closest car
- In adjacent lanes — Distance to the closest cars in front in the left lane and the same for the right lane

There's a set of states that we must take it in account as follows:

1. Traffic lights:

In this case, we have an incoming traffic light detected by object detection module and we have its depth (distance from car to traffic light), so we check the state of traffic light if it's red the car will stop else the car will continue in its lane.

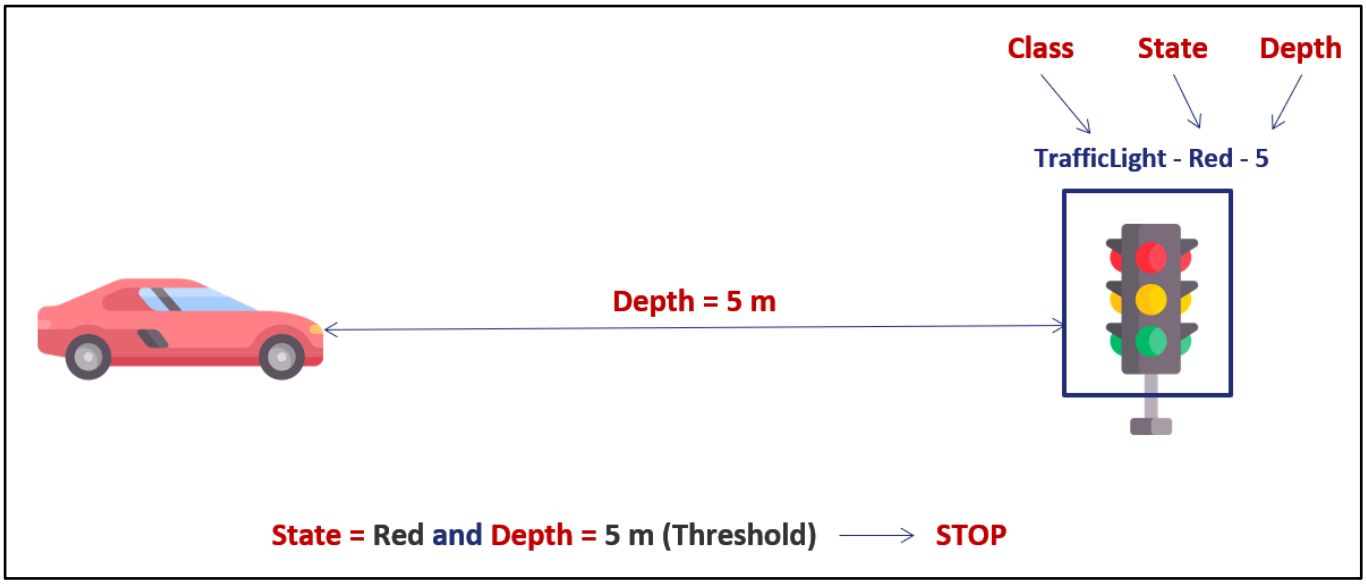


Figure 72: Traffic light behavior

2. Car avoidance:

In this case, we have an incoming car detected by object detection module and we have its depth (distance from car to another car), so we check if the ahead car is not moving so the car will stop else the car will continue follow the ahead car.

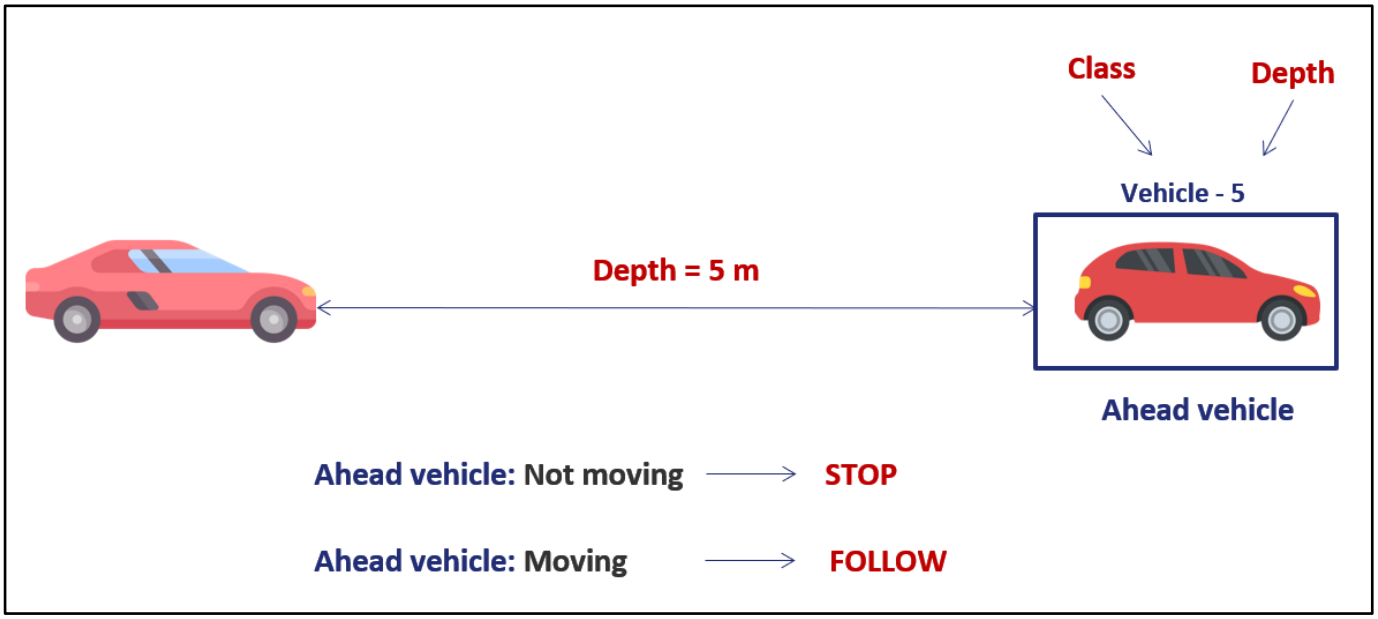


Figure 73: Car avoidance behavior

3. Pedestrian avoidance:

In this case, we have an incoming pedestrian detected by object detection module and we have its depth (distance from car the pedestrian), so we check if the pedestrian is in the car lane, if in the lane stop else continue following the lane.

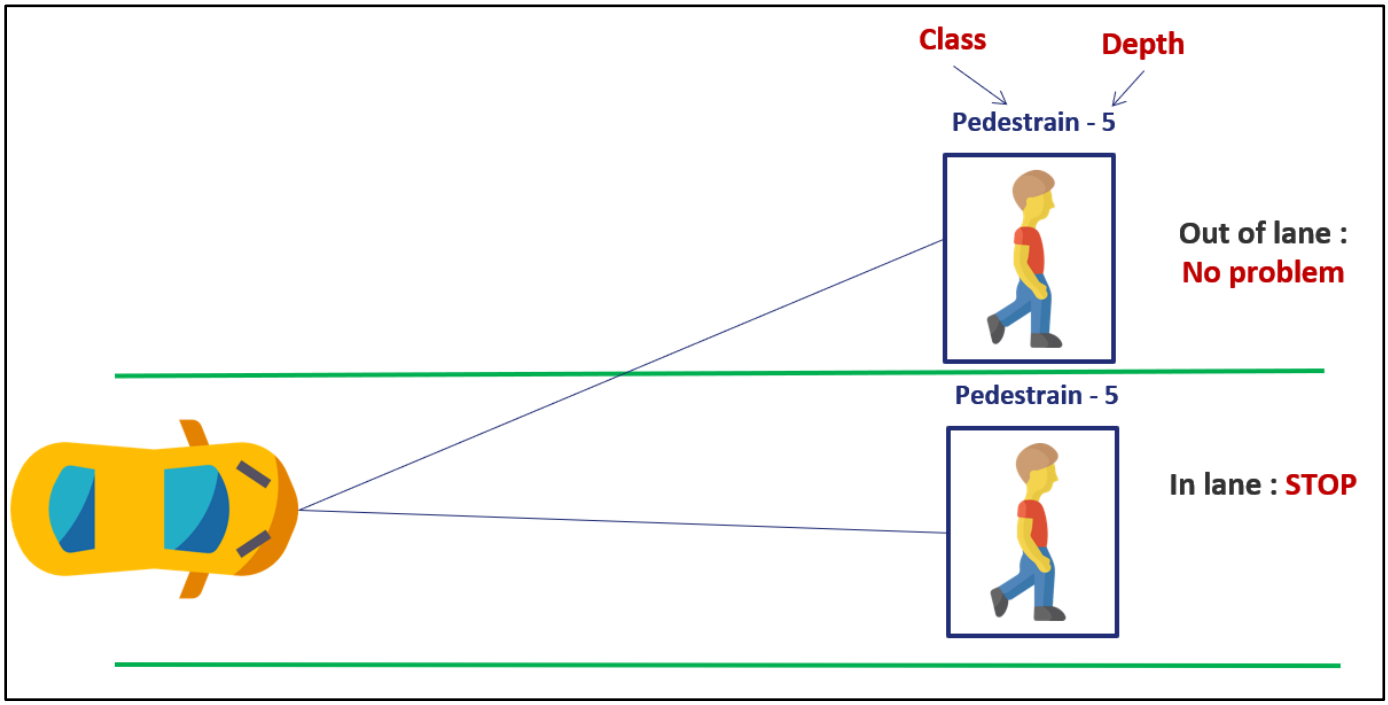


Figure 74: Pedestrian avoidance behavior

4. Lane following:

In this case there's no states from above states so the car continues in its path

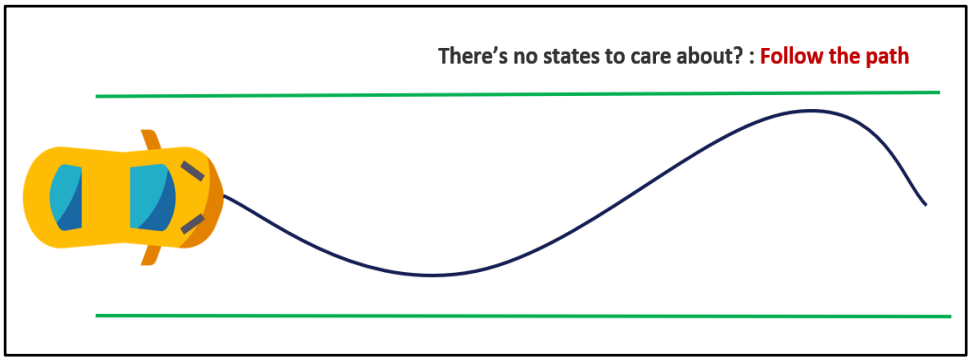


Figure 75: Lane following behavior

STEP 2: Take decisions using Finite-State Machine:

The principle is to define, according to the situations, the possible states of a car. On a highway, the state of a car may be to stay in a lane, change lanes to the left, or change lanes to the right. Depending on the traffic conditions, we change state to, for example, overtake a car.

When we take the decision to overtake a vehicle, the algorithm generates several trajectories for a decision and chooses the best one according to the criteria of feasibility, safety, legality, efficiency, comfort, …

In this case of overtaking, the red / orange trajectories are dangerous, the yellows are acceptable but incomplete, the green is the most efficient and safe.

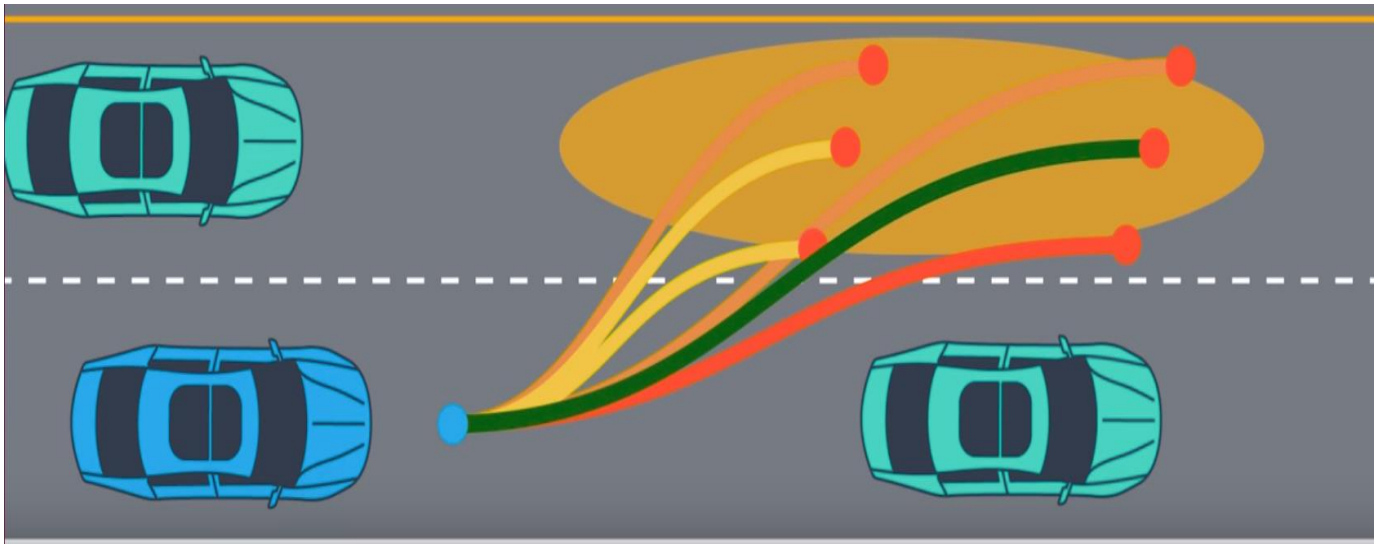


Figure 76: Path trajectory

To generate this trajectory, we create a level five polynomial that passes through waypoints. Waypoints are points on the way that contain 3 dimensions:

S: The longitudinal distance
D: The lateral distance
T: The moment at which one must pass through this point; giving speed

A trajectory is a curve that goes through all these points. These points are positioned in space and time. They tell us when to move to a specific (x;y) position and how fast. If you want to brake at a pedestrian crossing, we create points up to the pedestrian crossing and set a decreasing velocity to the speeds of the points up to the stopping zone.

9-

Module 3 | Control

01 Introduction

An autonomous car uses the perception module to know its environment, and the Planning module to make decisions and generate trajectories.
The Control module is now in charge of moving the vehicle by generating an angle for the steering wheel (lateral control) and an acceleration (longitudinal control)

Waypoint:

A path is a sequence of waypoints each containing a position (x; y) an angle (yaw) and a speed (v).
The Control step consists of following the trajectory generated as faithfully as possible. A path is a sequence of waypoints.

A Control algorithm is called a controller. The purpose of a controller is to generate instructions for the vehicle such as steering wheel angle or acceleration level taking into account the actual constraints (road, wind, wheel slip ...) and the trajectory generated.

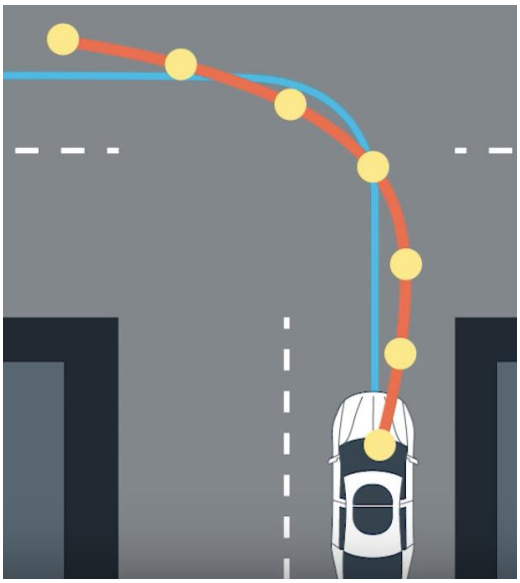


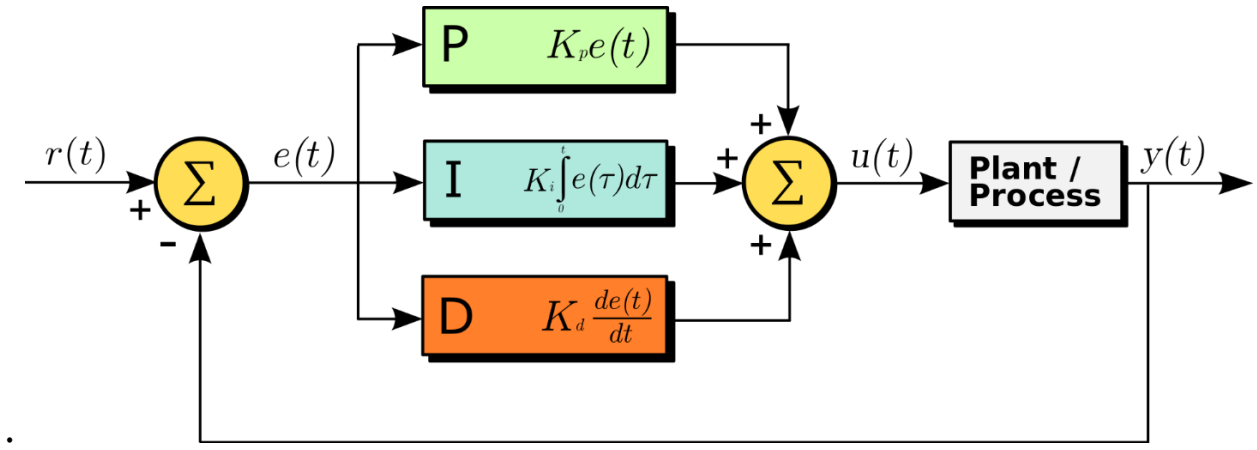
Figure 77: Waypoints

02 PID

Now to execute every control step in every waypoint in the trajectory we need to use PID controller to control the values of acceleration and steering angle Proportional-Integral-Derivative (PID) control is a control algorithm can be used in industrial control applications like pressure, flow, temperature, and speed. The popularity of PID controllers are often attributed because: their functional simplicity, and good performance in various conditions, which helps to operate in a simple, straightforward manner.

PID controller algorithm consists of three basic coefficients can be tuned to get the optimal response: proportional, integral, and derivative.

PID controller read the sensor output, then calculate the output of the actuator using proportional, integral, and derivative responses and summing the three components as shown in the following equation.



$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$

Figure 78: PID controller

Using a P controller (proportional) for example to steer the vehicale, we might find that the car overshoots the desired path. To avoid the overshoot we can use PD controller that notice the reducing error then counters steering actions ,so the car won't overshoot to reduce the error. To eliminate the steady state error we can use the I controller.

The PID controller imparts both transient and steady-state response improvements to the system, and PID controllers are often tuned using empirical rules, such as the Ziegler–Nicholas rules.

We can use PID controller for lateral control to steer the car and longtudinal control to accelerate or reduce the speed of the car, we can also use differente coefficients for city and high way speeds (high way > 50 unit speed)

P: Proportional:

This term applies a correction to the steering wheel proportional to the error. If we are too far from the goal, we turn the wheel in the other direction.
The disadvantage of a single P Controller is that it causes a constant oscillation.

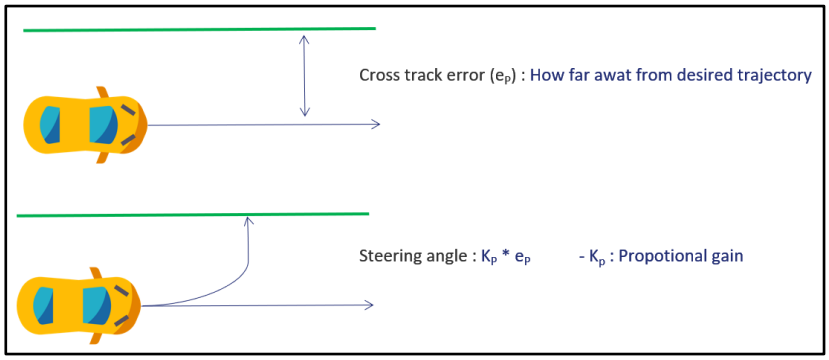


Figure 79: P controller

D: Derivative:

The purpose of the term D is to suppress this oscillation effect by adding a damping term to the formula. This term is the change of error. The PD controller understands that the error decreases and slightly reduces the angle it adopts to approach a smooth path.

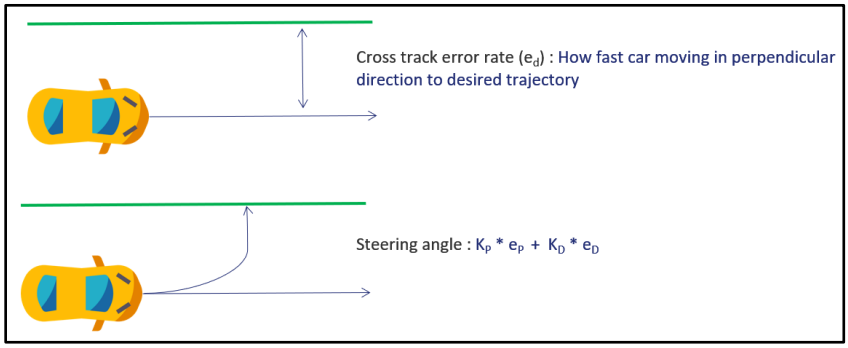


Figure 80: D controller

I: Integral:

The last term is used to correct a mechanical error that causes us to turn the wheel more or less strong depending on the vehicle to stay upright. So, we add a last term to penalize the sum of cumulative errors. The Twiddle PID curve corresponds to the use of an algorithm to find the coefficients rapidly, thus converge more quickly towards the reference trajectory.

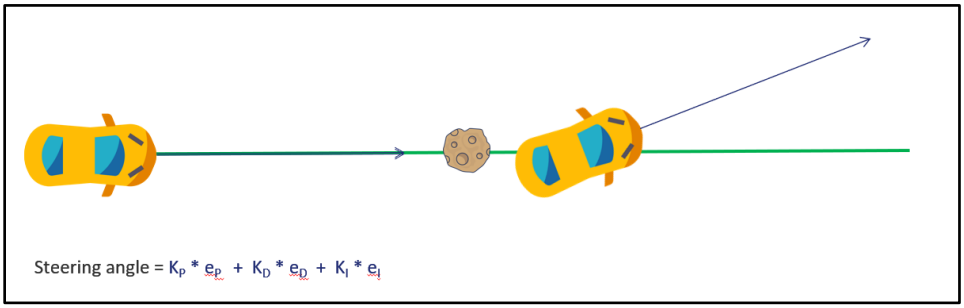


Figure 81: I controller

By tuning the three terms gain K_P , K_D and K_I we can achieve a good PID controller

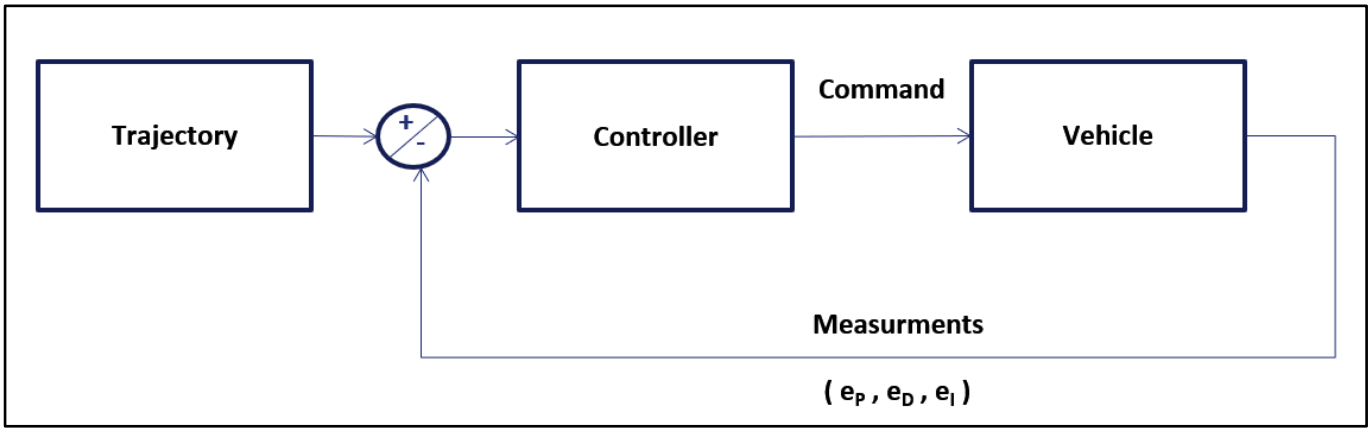


Figure 82: PID block diagram

10- Testing

In this section we will apply black box testing to our modules and other sub tasks with various test cases from our system.

9.1 Object Detection:

For Input Image:

- Image contains one object.
- Image contains more than one object.
- Image contains no objects.

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Image contains one object.	Detect one object and draw the bounding box	As Expected.	**Pass**
TC2	Image contains more than one object.	Detect all objects and draw the bounding boxes	As Expected.	**Pass**
TC3	Image contains no objects.	No Detection	As Expected.	**Pass**

9.1.1 Traffic light classification:

For Input traffic light image:

- Image contains one traffic light.
- Image contains more than one traffic light.
- Image contains no traffic lights.

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Image contains one traffic light	Classify the traffic light as green, yellow, or red	As Expected	**Pass**
TC2	Image contains more than one traffic light	Classify all the traffic lights as green, yellow, or red	As Expected	**Pass**
TC3	Image contains no traffic lights	No Classification or Unknown color for misclassified objects as traffic light	As Expected	**Pass**

9.2 Lane Detection:

For Input Image:

- Image contains one Lane.
- Image contains more than one Lane.
- Image contains no Lanes.

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Image contains one Lane	Detect the lane and draw lane boundaries	As Expected	**Pass**
TC2	Image contains more than one Lane.	Detect all lanes and draw lane boundaries	As Expected	**Pass**
TC3	Image contains no Lanes	No Detection	As Expected	**Pass**

9.3 Depth Estimation:

For input Object image:

- Image contains one object in front of the stereo camera.
- Image contains more than one object in front of the stereo camera.
- Image contains one object in any side (left or right) to the stereo camera.
- Image contains more than one object in any side (left or right) to the stereo camera.
- Image contains no objects.

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Image contains one object in front of the stereo camera	Calculate the distance to the object and show the distance	As Expected	**Pass**
TC2	Image contains more than one object in front of the stereo camera	Calculate the distance to all the objects and show the distances	As Expected	**Pass**
TC3	Image contains one object on any side (left or right) to the stereo camera	Calculate the distance to the object and show the distance.	Wrong distance	**Fail**
TC4	Image contains more than one object on any side (left or right) to the stereo camera	Calculate the distance to all theobjects and show the distances	Wrong distance	**Fail**
TC5	mage contains no objects.	No Detection	Wrong distance	**Fail**

9.4 Planning:

For Input source and destination points on the map:

- Different source and destination points.
- Same source and destination points.

Test case ID	Test scenario	Expected Results	Actual Results	Pass/Fail
TC1	Different source and destination points.	Car moves from source point and stops at destination point.	As Expected	**Pass**
TC2	Same source and destination points.	Car doesn't move.	As Expected	**Pass**

11- Future Work

- Detect the orientation of the vehicle.
- Detect 3D bounding box of the object instead of the 2D bounding box to determine its depth.
- Apply Kalman filter to track the objects' state and apply sensor fusion with the camera to get better performance.
- Improve lane detection using state of the art networks.
- Use algorithms like RRT instead of A* as they are faster, and compare between their performance.
- Use more cameras to have a 360 view of obstacles surrounding the car.
- Have a hardware implementation.
- Implement lane-change in Carla simulator using the detected lanes.
- Improve the performance of the planning module in Carla simulator.
- Try reinforcement learning for Carla simulator to navigate the map.
- Use generative adversarial networks (GANS) to predict the behavior of the surrounding vehicles.

12- Project timeline

In this section we will show our project timeline as following:

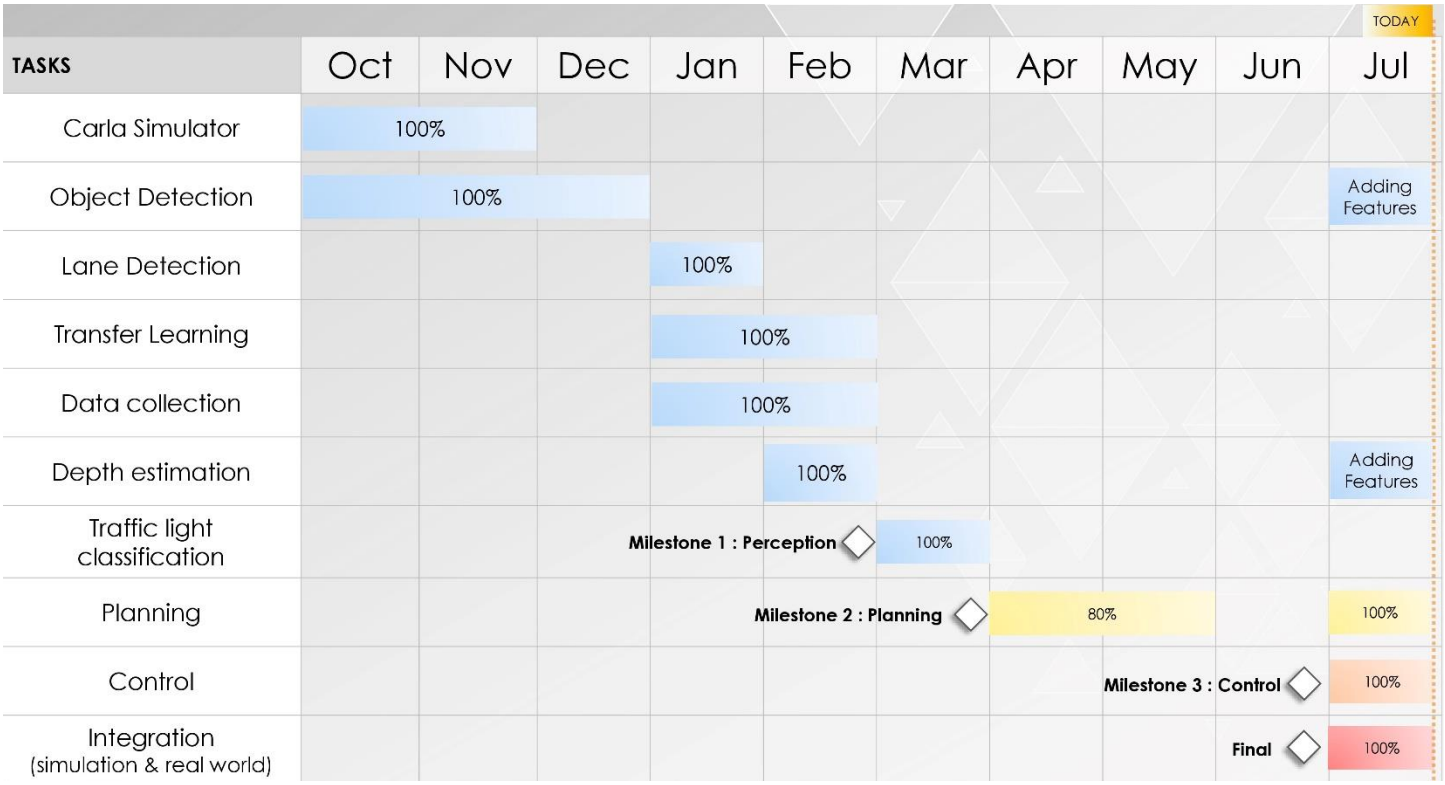


Figure 83: Project timeline

13- References

[1] National Highway Traffic Safety Administration: Annual United States Road Crash Statistics. https://www.nhtsa.gov/ , 2018.

[2] The Society of Automotive Engineers: Levels of driving automation. https://www.sae.org/ , 2018

[3] National Highway Traffic Safety Administration: Percentage of Car Accidents Are Caused by Human Error. https://www.nhtsa.gov/ , 2018.

[4] Carla Simulator. https://carla.org/ , v0.9.10

[5] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning dsdafs affordance for direct perception in autonomous driving," in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730.

[6] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," arXiv preprint, 2017.

[7] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," Electronic Imaging, vol. 2017, no. 19, pp. 70–76, 2017.

[8] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," arXiv preprint arXiv:1807.00412, 2018.

[9] S. Baluja, "Evolution of an artificial neural network based autonomous land vehicle controller," IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, vol. 26, no. 3, pp. 450–463, 1996.

[10] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving largescale neural networks for vision-based reinforcement learning," in Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, 2013, pp. 1061–1068.

[11] E.Yurtsever, et al, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies", arXiv preprint arXiv:1906.05113, 2020.

[12] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627–1645, 2010.

[13] R. Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", (CVPR) 2014. IEEE, 2014.

[14] R. B. Girshick. "Fast R-CNN", ICCV 2015.

[15] Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", arXiv preprint arXiv:1506.01497, 2015.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection", arXiv preprint arXiv:1506.02640, 2015.

[17] J. Redmon and A. Farhadi. "Yolo9000: Better, faster, stronger", arXiv preprint arXiv:1612.08242, 2016.

[18] J. Redmon and A. Farhadi. "Yolov3: An incremental improvement", arXiv preprint arXiv:1804.02767, 2018.

[19] J. Redmon. Darknet: Open-source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[20] A. Bochkovskiy et al, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, arXiv preprint arXiv:2004.10934, 2020.

[21] X. Long et al, “PP-YOLO: An Effective and Efficient Implementation of Object Detector”, arXiv preprint arXiv:2007.12099, 2020.

[22] Computer Vision Nanodegree, Udacity. https://www.udacity.com/, 2020.

[23] A. Saxena, S. H. Chung, and A. Y. Ng, “Learning depth from single monocular images,” in Advances in neural information processing systems, 2006, pp. 1161–1168.

[24] C. Godard, O. M. Aodha, M. Firman, G. Brostow, “Digging Into Self-Supervised Monocular Depth Estimation” ICCV 19.

[25] J. Zbontar and Y. LeCun. Stereo matching by training a convolutional neural network to compare image patches. Journal of Machine Learning Research, 17(1-32):2, 2016.

[26] Jia-Ren Chang, Yong-Sheng Chen “Pyramid Stereo Matching Network” CVPR 2018.

[27] Y. Ko et al, Key Points Estimation and Point Instance Segmentation Approach for Lane Detection, arXiv preprint arXiv: :2002.06604, 2020.

[28] Dijkstraa VS A* algorithm, https://towardsdatascience.com

[29] A* algorithm, https://en.wikipedia.org/wiki/A*_search_algorithm