Autonomous Vehicle Sensors, Control Research and Development for Application in Industrial and Commercial Vehicles

This paper presents work on the conversion of a regular vehicle into an autonomous one, utilizing the latest communication technologies and architectures. The proposed design focuses on the sensory and control field, by exploring the feasibility of utilizing a deep learning model of UNET architecture with a Residual Network (ResNet) encoder to enhance the vision capability of lane detection. An investigation to reduce the cross-track error of the vehicle with an extended variation of the Pure Pursuit controller and Stanley controller was simulated. The design applied computer vision techniques to acquire lane imagery from the camera sensor and trained a deep learning model to perform semantic segmentation, which will detect and distinguish between left and right boundaries. The performance of the model is captured through the observation of a dice loss plot and feeding in unseen lane images to test the lane boundary predictions and the application of inverse perspective mapping. Simulations are conducted on the CARLA simulator to test the design standard and performance of autonomous vehicles which is intended to examine the cross-track error of the vehicle, under several environmental conditions and system restrictions which from the results indicated lower cross-track error measurements.

Keywords: *machine learning, deep learning, autonomous vehicle, computer vision, vehicle control*

Introduction

The race for developing a fully autonomous vehicle has seen several companies competing, to be the first to deliver into the market, however, the industry is a long way from reaching a stage where no human interaction will be involved in an unpredictable road environment. Currently, technology is advancing to a state where the cost associated with implementation increases beyond what any average consumer will be willing to pay. Since most autonomous technologies are already incorporated with the vehicle, and to be sold as a whole unit which increases the cost of production, it provides an opportunity to create the technology that transforms any regular vehicle with a set of preliminary requirements into a self-driving vehicle, achieving the same results without comprising safety. Autonomous vehicles are classified into levels, ranging from no automation (Level 0) to full automation (Level 5) (SAE, 2018). Each stage classifies the benchmarks that the vehicle should be able to achieve.

Over 90% of accidents on the road are caused by some type of human error (Moujahid, et al., 2018). This statistic alone can be justification for allowing artificial intelligence to assist or completely take control of dangerous scenarios. Although the technology is greatly advancing, there exist other

challenges facing the field which include the public perception, safety, and legal issues that contribute a significant factor towards the success of the technology. The survey results from (Schoettle & Sivak, 2014) and (Kyriakidis, et al., 2015), clearly highlight the potential of autonomous vehicles regarding their attractiveness to users however the standards of safety measures need to be more stringent to convince the skeptics. A potential remedy would be to introduce conversion technology to transform a regular vehicle into an autonomous vehicle whilst ensuring safety is not compromised.

Literature Review

Computer Vision

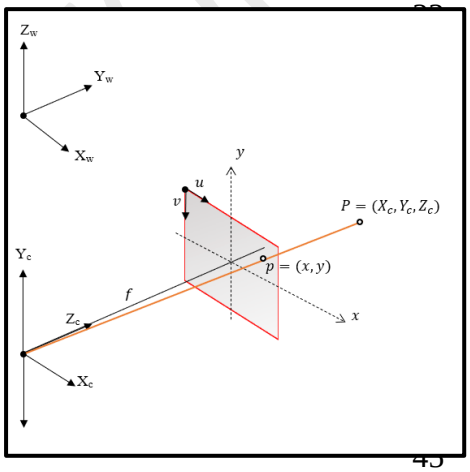
Vision in autonomous vehicles is one of the primary channels to extract critical information from the real world and utilizing image processing, its capabilities allow it to produce more appropriate actions when navigating in an unpredictable road environment.

By incorporating computer vision, provides the ability to track criteria such as road lanes, recognize road signs, object detection, and classification whilst navigating the world in real-time (Campbell, et al., 2018). This function allows scanning of the environment almost instantaneously and plays a vital part in the role of the vehicle to make the appropriate action. The concepts below provide mathematical formulae and principles that are necessary for how to interpret and extract information in a 3-Dimensional space.

Pinhole Camera Model

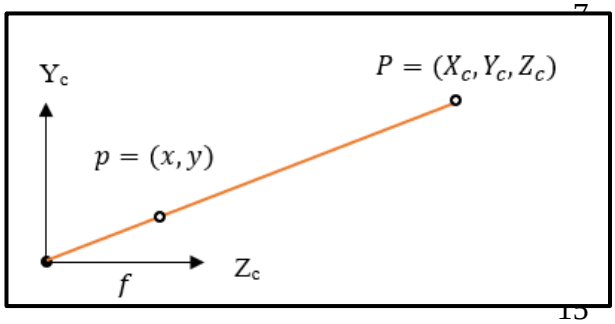
The pinhole camera model provides the mathematical relationship for how a point in 3-dimension space can be represented onto a 2-dimensional plane or the image plane.

Figure 1. *Pinhole Camera Model*



The above figure showcases how a 3-dimensional point in space P, having coordinates X_c, Y_c, Z_c respectively in the camera reference frame that can be represented as a 2-dimensional point p having coordinates x, y in the image plane.

Figure 2. *Pinhole Camera Model 2D representation*



Applying similar triangles

$$\frac{x}{f} = \frac{X_c}{Z_c}$$

which yields

$$x = f\frac{X_c}{Z_c}$$

and similarly,

$$y = f\frac{X_c}{Z_c}$$

To acquire the pixel coordinates u and v that is located at x and y, it is needed to first denote the principal point as u_0 and v_0 which is the centre of the image plane. Therefore, given an image with height H and width W, the principal point can be described as $u_0 = \frac{W}{2}$ and $v_0 = \frac{H}{2}$. The image plane consists of sensor pixels with its own width and height properties, hence if the pixel width is denoted as k_u and pixel height as k_v , the projection of the point P in 3-dimensional space carried out by the camera will correspond to pixel coordinates of:

$$u = u_0 + \frac{1}{k_u}x = u_0 + \frac{f}{k_u}\frac{X_c}{Z_c}$$
$$v = v_0 + \frac{1}{k_v}x = v_0 + \frac{f}{k_v}\frac{Y_c}{Z_c}$$

Denoting $\frac{f}{k_u}$ and $\frac{f}{k_v}$ as α_u and α_v respectively yields the matrix neglecting the skew parameter

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z_C}\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z_C}\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

Homogenous coordinates provide a means to represent an N-dimensional coordinates with N+1 numbers. This is achieved by simply adding an additional coordinate of 1. Considering the above equation is homogeneous, it can simply be represented as:

$$\lambda\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

simplified to

$$\lambda\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \quad (1)$$

where λ represents a non-zero scalar value. The matrix denoted by K is known as the Intrinsic matrix of the camera describes the properties of the camera and what happens inside (Sturm, n.d.).

Equation (1) showcases that regardless of whether coordinates P is multiplied by a non-zero value, it will represent the same pixel value on the image plane.

Camera Projection

In practical applications, the point of interest is usually in another coordinate reference frame such as the world reference frame. Therefore, it is necessary to transform the world coordinates into camera coordinates (Theers & Singh, 2020).

Euclidean transformations from world coordinate system to camera coordinate system can be expressed by rotation with a 3x3 rotation matrix denoted by R and translation denoted by the vector t:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R\left(\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} - t\right) = R\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} - Rt$$

and expressed as homogeneous coordinates as:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R^T & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I & -t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

The complete model from world coordinates to pixel coordinates in the camera reference frame

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R^T & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I & -t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

simplified to

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K(R|t) \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad (2)$$

Inverse Perspective Mapping

Through camera projection, 3-Dimensional coordinates can be mapped onto a 2-Dimensional image plane. However, there exist perspective effects on the image in which the information acquired from an image would not be useful when determining distances. To negate the perspective effects, inverse perspective mapping is utilized to acquire a bird's eye view of the image (Lee, 2010).

To perform the reverse operation, whereby a 2-Dimensional image is mapped to its respective 3-Dimensional coordinates, although impossible without the depth parameter, but by making few assumptions, the corresponding coordinates can be attained. Looking at equation (1) again:

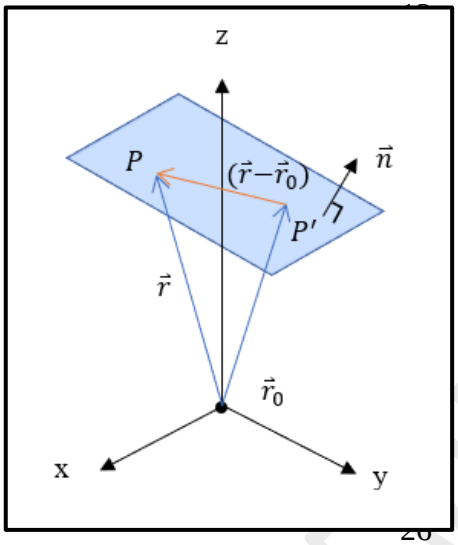
$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix}$$

To acquire the camera coordinates, the intrinsic matrix is simply inversed and multiplied by our scalar value λ and our pixel coordinates.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = K^{-1} \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (3)$$

The value of λ is typically unknown, as the case when looking at a 2-Dimesional image, it is possible to know where the light ray came from but it can never be known how long the light ray has travelled to reach the camera. However, by assuming that the coordinates of interest lie on the same plane or ground plane and knowing the location of the camera with respect to set ground plane, it becomes a geometry problem which yields the distance or depth of each pixel. Equation (3) describes the vector at which our point lies on ground plane. By defining the normal to camera reference frame n_c as a vector $R_c(0,1,0)^T$ where R_c is the rotation of the camera with respect the ground.

Figure 3. *Planar Surface Representation*



The equation for a planar surface is described below (Dawkins, n.d.). From Figure 3, since $(\vec{r} - \vec{r}_0)$ lie on the same plane, the equation of a planar surface is given by the dot product of the vector on the ground plane and an orthogonal vector to the ground plane.

$$\overrightarrow{n_c} \bullet (\vec{r}-\vec{r}_0) = 0 \qquad (4)$$

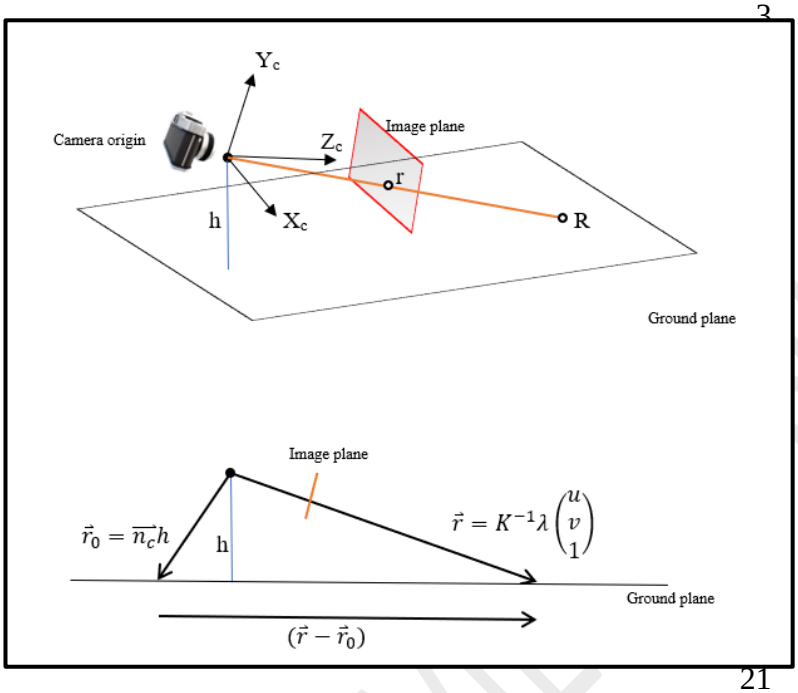
Therefore, if $\vec{r}_0$ chosen to be $\overrightarrow{n_c}h$, equation (4) becomes as follows:

$$\overrightarrow{n_c} \bullet \vec{r} - \overrightarrow{n_c} \bullet \vec{r}_0 = 0$$
$$\overrightarrow{n_c} \bullet \vec{r} - \overrightarrow{n_c} \bullet \overrightarrow{n_c}h = 0$$
$$h = \overrightarrow{n_c} \bullet \vec{r}$$

Plugging in the equation of $\vec{r} = K^{-1}\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ brings about final equation (5) which describes camera coordinates derived from pixel coordinates (Theers & Singh, 2020):

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \frac{h}{{n_c}^T K^{-1}(u,v,1)^T} K^{-1} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \quad (5)$$

Figure 4. *Inverse Perspective Mapping*



Deep Learning

Deep learning is sub branch of machine learning in which the model consists of multi-layered neural networks capable of learning high level complexity from data. Unlike machine learning, deep learning is useful in extracting features, identifying and classifying data all on its own.

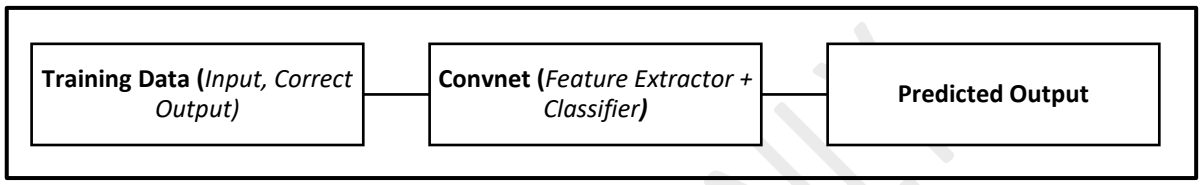
Deep learning neural networks also known as deep neural networks are modeled essentially on how the human brain works. Each layer of the network allows deep learning algorithms to progressively learn and make predictions with the accuracy increasing over time. The solution of deep learning is meant for complex tasks that cannot be solved through classical approaches in programming and software. Research into the deep learning field aims to create models that can learn representations of large unlabeled and unstructured data. Some of the concepts of deep learning are closely inspired from neuroscience in mimicking how the brain operates and functions through stimuli which falls under information processing and communication patterns (Vishnukumar, et al., 2017).

Convolutional Neural Network

A convolution neural network or convnet is a deep neural network which aims to mimic the working principles of how the visual cortex of the brain

processes and perceives images. This image recognition process can be regarded as essentially a classification task in which the aim is classifying what class images fall under. Before convnet were established, feature extractors were designed which yielded significant cost and time for a high-performance system. These feature extractors were completely independent from the field of machine learning however with the development of convnets, the feature extractor ability is included in the process rather than designing it due to the kind of neural networks being utilized (Kim, 2017).

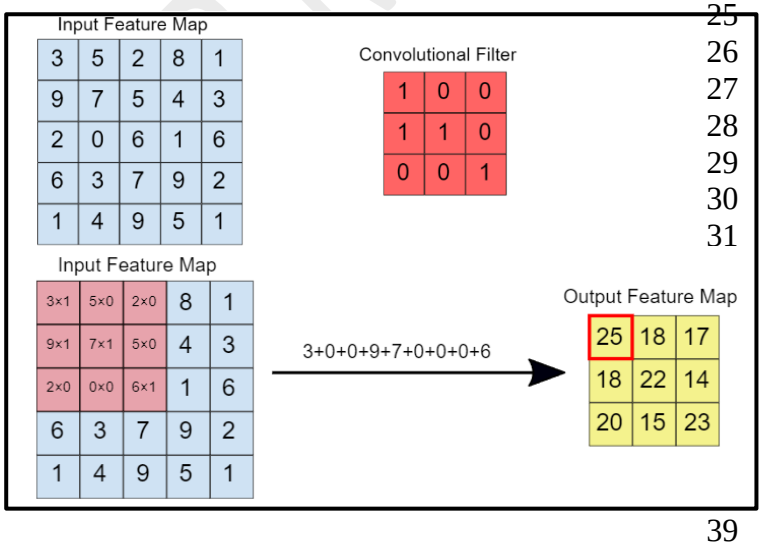
Figure 5. *Convolution Neural Network Methodology*



As with most neural networks, the more hidden layers in convnet, the higher the performance of the system in dealing with complex tasks which certainly comes with the additional difficulties of cost and time taken in the training process. A typical architecture of a convnet consists of the convolution layer, pooling layer, flattening and full connection.

Convolution layer like the name implies, takes the input image and applies the convolution operation to create feature maps. Unlike standard neural networks that make uses of weights and biases, the convolution layer utilizes filters that makes these conversions. Filter are usually two-dimensional matrices that detect features and are placed over the input image covering all areas to create set feature map

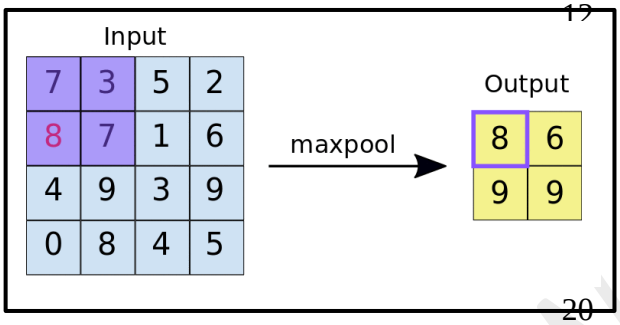
Figure 6. *Convolutional Process*



Source: (Google, 2021)

Pooling layer aims to reduce the size of the image by combining neighboring pixel values of a single area into a single representative value. Applying this layer, allows the network to have spatial variance which provides the ability of the network to detect same object images that is spatially different or slight modifications. The reduction of size also helps minimizing the computational load and prevent overfitting scenarios. Some types of pooling operations include max pooling and average pooling where max pooling takes the maximum value in an area as the representative value whereas average pooling will take the average pixel values in an area (Yu, et al., 2014).

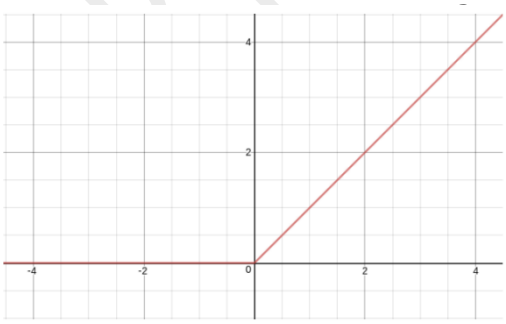
Figure 7. *Max Pooling Operation*



Source: (Google, 2021)

Activation applies non-linearity in the output of the convolutional layer which is a combination of the convolutional operation and pooling operation. Usually, a Rectified Linear Unit or ReLu function is utilized in a convolutional layer and the most the widely used activation function in neural networks. It will output the input directly if it is positive allowing it to not activate all the neurons at the same time making it computationally efficient (Agarap, 2019).

Figure 8. *ReLU Function*



Source: (Agarap, 2019)

Flattening is relatively a simple process in which, after having acquired a feature map, it is a matter of essentially just flattening the pooled feature map into a column to create a single vector of data. This process allows the network to handle the data much more efficiently than passing through a matrix of data. These small steps of adjustments, however simplistic the step maybe, provides

a big difference when dealing with the computational load of the system and the complexity of the system.

Full connection layer is the same as the hidden layer in a neural network in which the data is taken and makes the network more capable of classifying images through combining features and other attributes. The error function is also calculated in this process which is the same as the loss function previously in neural networks.

Figure 9. *Convolutional Neural Network Detailed Process*

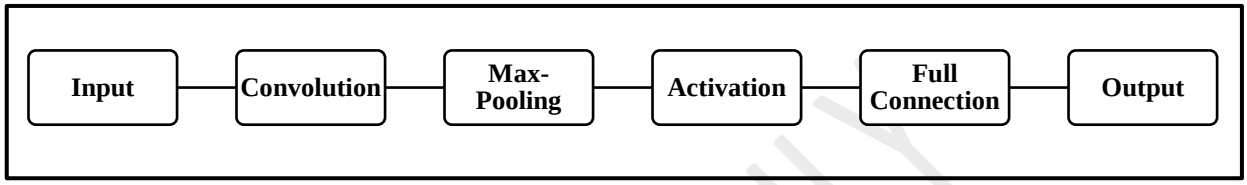
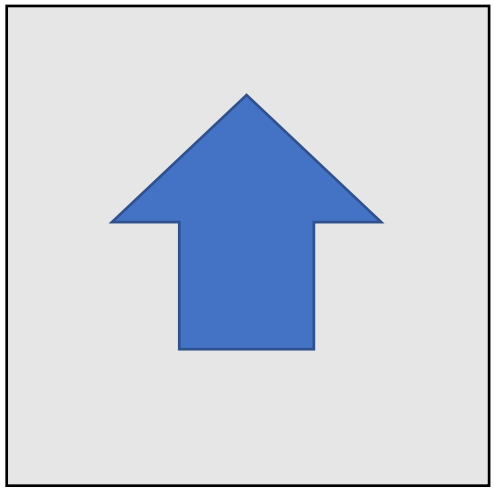


Image Segmentation

The process of classifying each pixel in an image is known as image segmentation. The aim is to assign certain objects to its own class. This approach is utilized in the computer vision applications where the need to detect objects or classify images. Image segmentation can be separated into two different categories, namely semantic segmentation and instance segmentation. In semantic segmentation, each pixel is associated with a class or label and in instance segmentation similarly it classifies each pixel but it is able to distinguish instances of the same class or label (Jordan, 2018).

Figure 10. *Image Segmentation (0-no arrow) (1-arrow)*

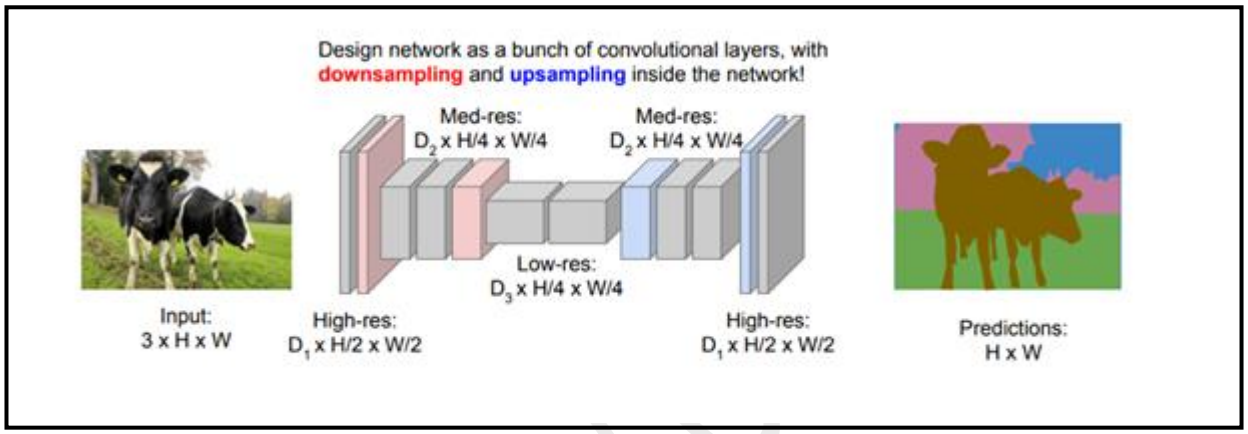
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	**1**	0	0	0	0	0
0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	0	**1**	**1**	**1**	**1**	**1**	0	0	0
0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0
0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	0	0	**1**	**1**	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0



The figure below showcases a UNET implementation for the purposes of semantic segmentation (Fei-Fei, et al., 2017). An image of shape (3xHxW) goes through the downsampling or encoding part by means of a series of convolutions and pooling operations to detect the features of the image. As the

network gets deeper by more convolutional layers, the network is able to detect more complex features. The upsampling or decoder represents the operations to return the features back into the original location usually by transpose convolutions. During the decoder part, the depth or channels of the network get reduced whilst the image increases in resolution.

Figure 11. *UNET Implementation*



Source: (Fei-Fei, et al., 2017)

Vehicle Control Methods

Ackerman Steering Angle

The Ackerman steering condition occurs when the axes of all the wheels intersect at a single turning point. In order for a vehicle to turn a corner, the front wheels of the vehicle would have to swivel at an angle. Turning both front wheels at the same angle would the concept of parallel steering however, it introduces slip onto the tires. The Ackerman condition is where the inner wheel and outer wheel are at different angles allowing for the vehicle to turn slip-free (Jazar, n.d.)

Kinematic Bicycle Model

The kinematic bicycle model is a popular control orientated approach when representing vehicles. The approach is formed by combining the two front wheel and rear wheels into a two-wheeled system. This allows the dealing with two wheels and one steering angle. The approach for the model also involves the concept of the Ackerman steering geometry which describes how an inside wheel must travel at a greater arc radius than the outer wheel allowing for the vehicle to turn. The assumptions made when using the kinematic bicycle models is that all slip angles are zero. The kinematic model is more computationally efficient when compared to a dynamic vehicle model and shown to have comparable performance (Law, et al., 2018).

Figure 12. *Kinematic Bicycle Model*

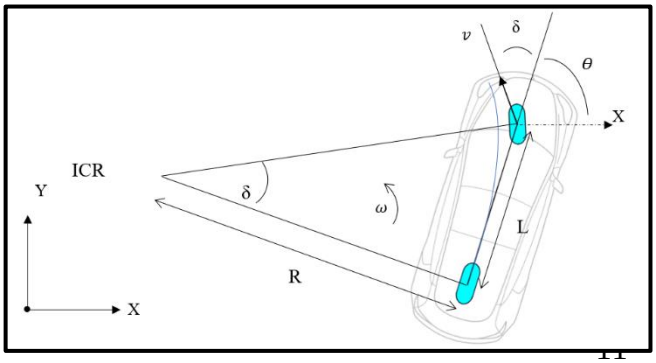


Figure 12 illustrates the kinematic bicycle model of a vehicle moving in a 2D X-Y plane with forward velocity v, steering wheel angle δ, wheelbase distance L, heading error θ and an instantaneous centre of rotation radius R. With a no slip condition, the formation of the equation of motions for the model is as follows:

$$\tan(\delta) = \frac{L}{R} \tag{6}$$

Subbing in $v = \omega R$, where ω is the angular velocity of the vehicle

$$\delta = \tan^{-1}(\frac{L\omega}{v})$$

Let (x, y) be a reference point on the rear axle of the vehicle. Since the rate of change of the heading error angle is equal to the rate of change of the

$$\dot{\theta} = \omega$$

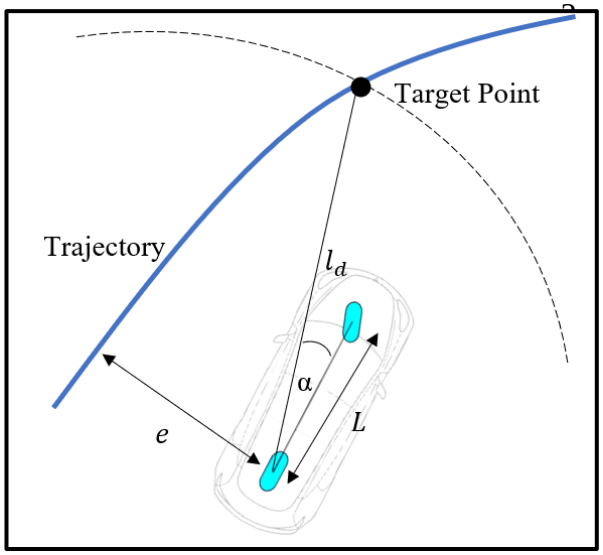
The equation of motion of the model becomes.

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \frac{v\tan(\delta)}{L}$$

Pure Pursuit

The pure pursuit method is a lateral control method used for controlling the steering wheel angle to reach a target point on the desired trajectorial path (Ding, 2020).

Figure 13. *Pure Pursuit Controller*



$$\frac{l_d}{\sin(2\alpha)} = \frac{R}{\sin(90-\alpha)}$$

$$\frac{l_d}{\sin(\alpha)\,cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

$$R = \frac{l_d}{2\sin(\alpha)}$$

From the kinematic bicycle model:

$$\tan(\delta) = \frac{L}{R}$$

Using the steering angle equation from the kinematic bicycle model, the desired steering angle becomes:

$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

To improve the performance of the controller, the lookahead distance l_d is varies with the speed of the vehicle.

$$l_d = K_{dd}v$$

where K_{dd} is a tuning constant. The steering angle for the pure pursuit controller becomes:

$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{K_{dd}v}\right) \qquad (7)$$

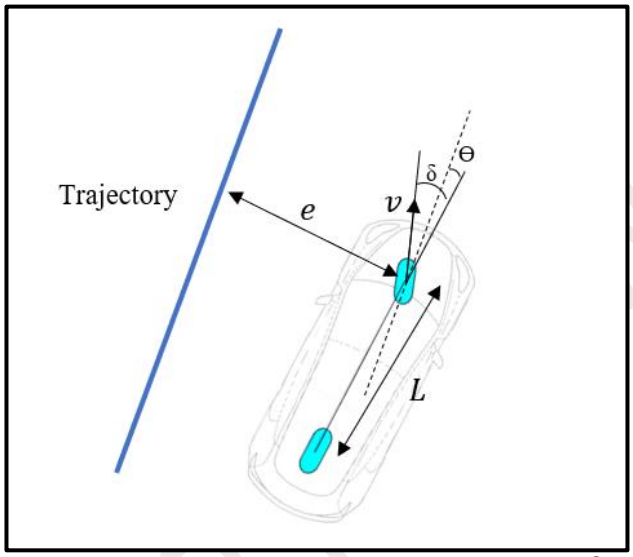
The cross-track error e is represents how far the vehicle is off-track from the trajectory. With a rear axle reference point, the cross-track error is calculated as:

$$e = l_d \sin(\alpha) \quad (8)$$

Stanley Controller

The Stanley controller was an approach used by Standford University's Grand Challenge team (Thrun, et al., 2006). It is a geometric path-tracking algorithm which, unlike the pure pursuit, looks at the cross-track error when controlling the steering wheel angle. The front axle of the vehicle is used as the reference point rather than the rear axle.

Figure 14. *Stanley Controller*



29

The Stanley controller involves three laws to be applied the steering angle. The first is to eliminate the heading error θ.

$$\delta(t) = \theta(t)$$

The next is to eliminate the cross-track error

$$\delta(t) = \tan^{-1}\left(\frac{ke(t)}{v(t)}\right)$$

and last involves ensuring that the steering angle is within the minimum and maximum bounds.

$$\delta(t) \epsilon [\delta_{min}, \delta_{max}]$$

The final equation for Stanley method becomes (Ding, 2020):

$$\delta(t) = \theta(t) + \tan^{-1}\left(\frac{ke(t)}{v(t)}\right), \quad \delta(t) \epsilon [\delta_{min}, \delta_{max}] \quad (9)$$

PID Controller

PID is an acronym for Proportional Integral Derivative. As the name suggests, these terms describe three basic mathematical functions applied to the error. It is a feedback mechanism that is widely used in the industry for control systems. The error $e(t)$ is defined as the difference between a desired target value and the measurement variable (Desborough, 2000)

$$e(t) = Target - Measurement$$

Proportional

The error value calculation, is simply applied to the output. This is known as proportional control and it is often necessary to scale the error value before adding it to the output by using the proportional gain K_p. By proving a too large gain or if the error is very large, the output may overshoot from the set value. Hence the change in output may turn out to be unpredictable and oscillating.

$$P = K_p \, \mathrm{e(t)} \qquad (10)$$

Design Methodology

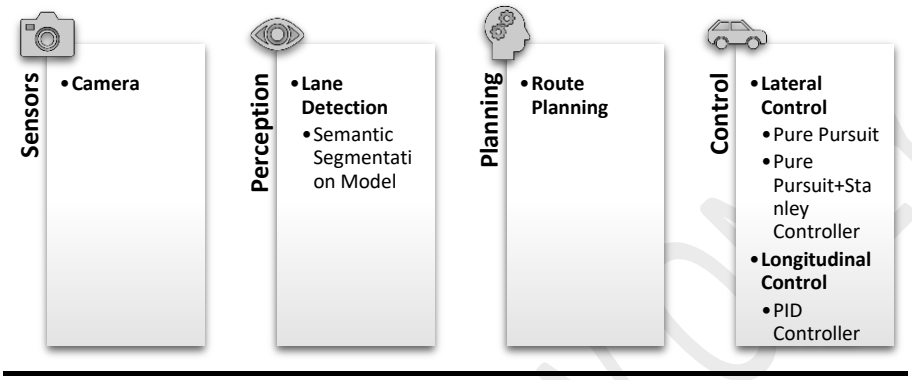
It is apparent that autonomous vehicles bring together fields from different aspects of engineering and still provide opportunities to improve and test different methods of integrating. After researching current technological trends in autonomous vehicles, a proposed design can be presented which highlights different approaches to different aspects of the autonomous vehicle. The section below will highlight the proposed structure of the autonomous vehicle.

Table 1, describes the specification details for the autonomous vehicle. An automatic transmission is a pre-requisite for the vehicle due to the feasibility of not having another modular system to assist in changing gears and can be a cumbersome process to determine ideal gear changes with different vehicles. Therefore, an automatic transmission provides an efficient platform to approach the conversion to autonomous vehicles. The location of the camera plays a key role in acquiring environmental information most efficiently. Thus, it will allow the application of inverse perspective mapping to determine the lane boundaries in a bird's eye perspective measured in metres. The wheelbase parameter will assist in facilitating the ideal steering wheel angle utilized in the lateral control methods.

Table 1. *Autonomous Vehicle Specifications*

Autonomous Vehicle specifications	
Transmission	Automatic
Chassis	Front Wheel Drive (FWD)
Wheelbase	2.8m
Camera height	1.3m
Camera Pitch	5°

Figure 15. *Autonomous Vehicle Breakdown*



Sensors

To acquire information from the environment surroundings, the autonomous vehicle is fitted with two sensors. An RGB camera is attached to the top of the front windscreen and orientated to be facing the road ahead. The camera will provide the essential information of knowing where the vehicle is located and through the field of computer vision, the necessary features of the environment can be extracted and processed by the control unit.

Lane Detection System

The detection of lanes pipeline will be made possible with the creation of a segmentation model which is a deep learning model that utilizes a CNN. The model will need to be fed training data to be able to learn the features of a lane boundary but also classify the pixels to whether it belongs to either a left lane boundary or a right lane.

Once the model predicts the left lane or right lane, the system will need to convert the pixel coordinates of the lanes into a polynomial by applying the principles of inverse perspective mapping. The assumption made when utilizing this method is that road is flat and lies entirely on the ground plane.

Route Planning

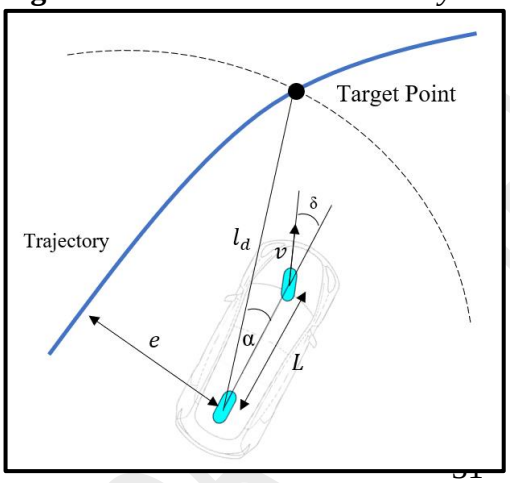
With the polynomial of the lanes being created from the lane detection system, the next step involves forming an actual path for the vehicle to navigate through. The trajectory that the vehicle will take will essentially be

the centre of the two-lane boundaries which is created from the lane boundary predictions

Control

A pure pursuit method provides a viable option to utilize on the vehicle however, to reduce cross-track error, a variation of two popular methods was explored in order to improve upon the performance of the lateral control by means of minimizing the cross-track error. To achieve this, the tail-following approach of the pure pursuit provided the base for the formation of the steering wheel angle, however, it does not consider the cross-track error when determining the angle. That was solved by incorporating part of the Stanley controller (Thrun, et al., 2006) into the equation. This will allow the path tracking algorithm to essentially be similar to a pure pursuit controller but take 'additional' steering angle towards the trajectory if the vehicle cross-track error increases.

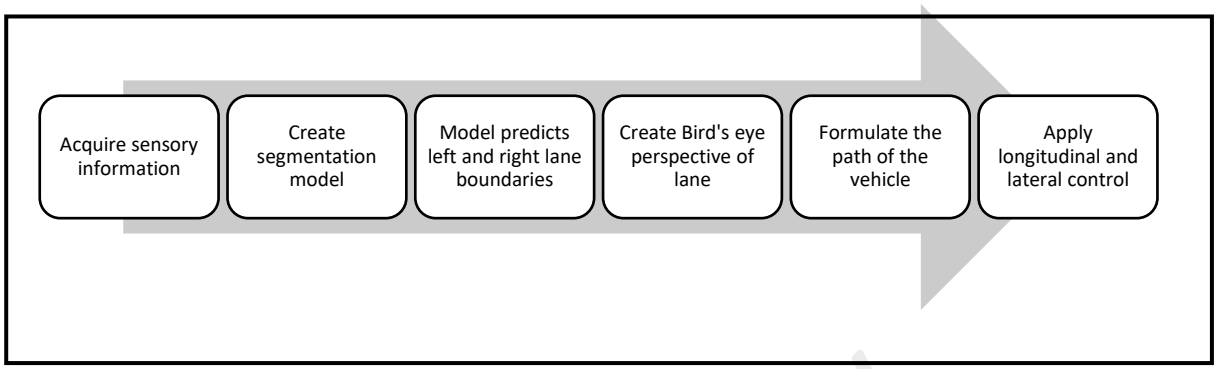
Figure 16. *Pure Pursuit + Stanley Controller Adaptation*



$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{K_{dd}v}\right) \pm \tan^{-1}(\frac{ke}{\mathrm{v}})$$

The system will need to formulate a target point on the trajectory with a variable distance depending on the speed of the vehicle. The max speed will denote the maximum lookahead distance, and a low speed will denote the lowest lookahead distance. Once the target point is created, the applicable angles can be created depending on the methods that can be calculated to formulate the steering wheel angle that the vehicle must utilize to reach the desired point. The important metric to keep track of is the cross-track error which will indicate the performance of the control method. For this design the reference point for which will indicate the cross-track error will be the centre of the vehicle.

Figure 17. *Autonomous Vehicle Design Block Diagram*



System Pre-Requisites

Deep learning requires extensive resources to train a model thus the system requires computational resources to produce effective and efficient outputs timely in order to perform the necessary actions of the vehicle. When doing deep learning tasks with Tensorflow, it can either be done utilizing the CPU or the GPU. Using the GPU to perform tasks provides a faster approach when training models, as the time taken to complete one epoch is greatly reduced compared to when using the CPU. In order to utilize the GPU for deep learning, a Nvidia GPU is required with the installation of CUDA and CuDNN libraries which will speed up the computer-intensive applications.

Table 2. *PC Specifications*

PC specifications	
CPU	Ryzen 5 3600
GPU	GeForce RTX 2060 Super
RAM	16Gb

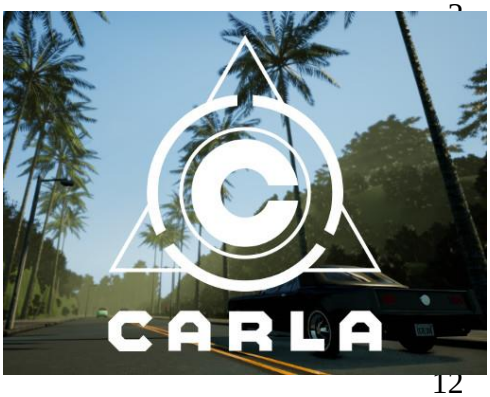
Table 3. *Software Specifications*

Software package specifications	
Operating System	Windows 10 Professional
Python	3.7.10
Microsoft Visual Studio Text Editor	1.60.2
Tensorflow	2.4.0
CUDA	8.1
CuDNN	11.0
PyTorch	1.7.1+cu110

CARLA Simulator

CARLA (Car Learning to Act) simulator is an open-source software used for training and researching autonomous vehicles.

Figure 18. *CARLA Simulator*



Source: (CARLA, n.d.)

The features of CARLA which highlighted the reason of choice for simulations of the autonomous vehicle were its flexible API, autonomous driving sensor suite, and fast simulation for planning and control. The API allows users to control almost all aspects of the CARLA environment, from sensors, vehicles, weather, and several other aspects. The Python API will allow writing python code to the CARLA server easily and be able to visualize the design and control of the vehicle (CARLA, n.d.). The software will provide the platform to add the sensors, apply the control methods discussed to observe how the vehicle responds, and analyze the feasibility of the design.

Segmentation Model Creation

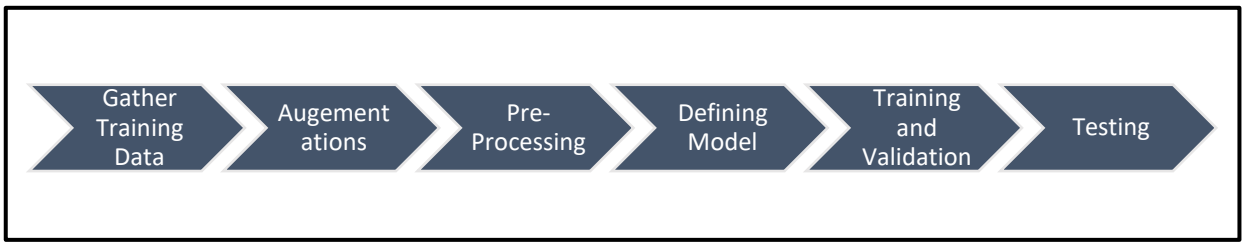
The model aims to take images of shape (1024,512,3) and output a segmented image of shape (1024,512,3) where the value of each pixel is the probability of it being in one of the three classes (No boundary, Left Boundary, and Right Boundary).

Table 4. *Model Parameters for Training Process*

Model parameters	
Architecture	UNET
Activation function	Softmax2d
Optimizer	Adam Optimizer
Learning rate	0.001
Batch size	5
Epochs	6
Classes	3
Loss function	Dice loss

The creation of the segmentation model can be broken down into 5 steps as shown below.

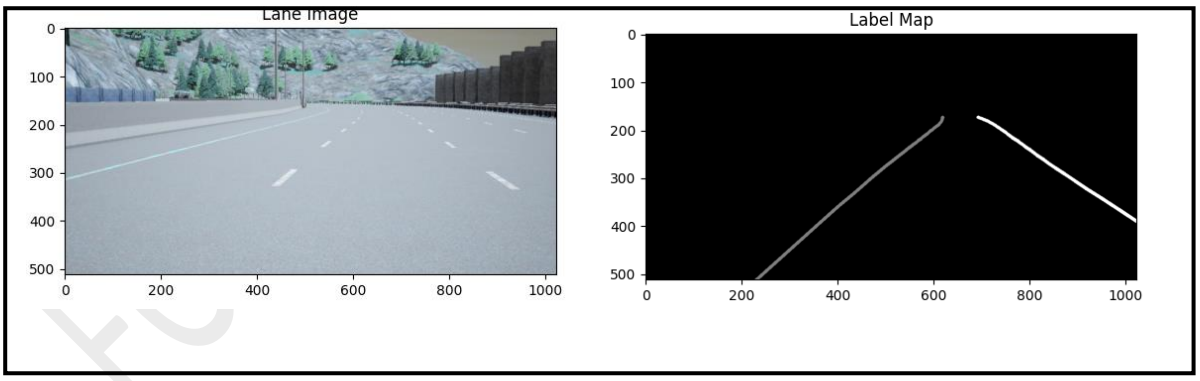
Figure 19. *Segmentation Model Creation*



Training Data

The fundamental requirement for any model is data, without data, a machine learning model will not have any knowledge or experience to learn from. The training data for our lane detection model is acquired from source (Theers & Singh, 2020) which collect images from the CARLA simulator providing lane images as well as label images. The training data will provide the baseline for the model and allow for it to utilize this dataset and learn how to identify the features of a boundary as well as the classifying of left lane boundary and right lane boundary. Below is an example of what the lane image and label image are being fed into the model. The label image seen below in the OpenCV environment is different from the actual label image as the actual image is not visible to the human eye however, due to the normalization feature of OpenCV, it is visible how the left lane and right lane boundaries are contrastingly different in which the system will understand to distinguish.

Figure 20. *Training Dataset*

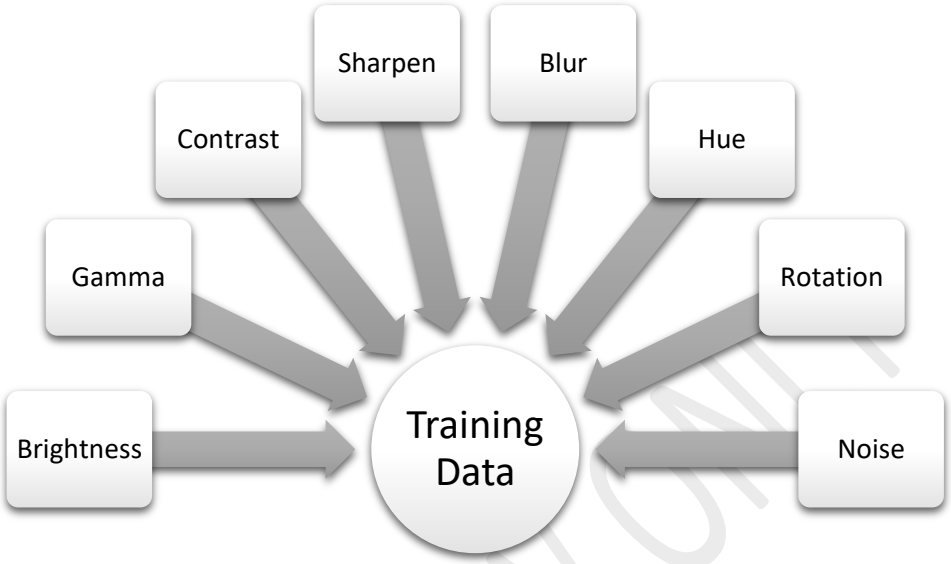


Augmentation

The more data a model has to train, the better the performance, however acquiring training data can be a tedious process when dealing with large amounts of data, however, a common practice involves when dealing these tasks involve data augmentations which is a technique to create synthetic data to increase the amount of training data available. Data augmentations involve taking the existing training datasets and applying techniques such as image rotation, flipping, blurring, or noising. These allow to multiply the size of the dataset without much effort and also allow the model to better generalize patterns and features of an image such that the model does overfit. The training

data is applied the following augmentation, with the assistance of the albumentation library which is a highly-optimized OpenCV library.

Figure 21. *Augmentations Applied to Training Dataset*



Pre-Processing

Data pre-processing ensures that the training data acquired is transformed into the correct format which is understandable for training the model.

PyTorch works with tensors which are essentially a multi-dimensional array that are used to encode inputs, outputs, and parameters of the model. What makes tensors specialized is their capability of running on GPU to allow for faster computing. The training data which was acquired needs to be transformed into a tensor format before feeding it to the defined model as an input.

Since the PyTorch model is using a model with pre-trained weights, another aspect to consider is to prepare the training data similarly to weights pre-training which will ensure better performance and faster convergence of the model.

The PyTorch model will expect input RGB images of shape (3 x H x W), loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

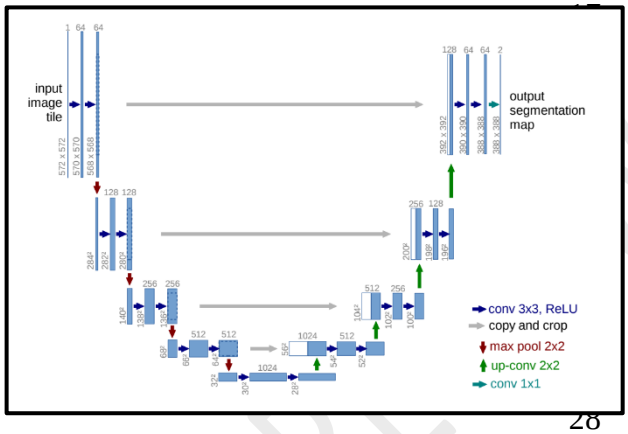
Architecture

PyTorch segmentation models' libraries allow the creation of models effortlessly with the feature of utilizing popular model architectures. The model being created is a segmentation model which is being represented by a CNN of UNET architecture.

The UNET architecture is a CNN created for semantic segmentation used in the field of biomedical. A typical CNN is useful in feature extraction however when the need for localization and classifying the pixel of an image to a class it became ineffective. The structure for this architecture can be broken

into two sections, namely the encoder/downsampling and the decoder/upsampling section. The encoder applies convolutions and pooling operations which essentially learns features of the image which, depending on how deep the network is, describes how complex features it can extract. Having detected the information of an image, it has lost the sense of localization in which the decoder functions are applied to recover the localization of the detected features. The decoder involves applying transposed convolutions with regular convolutions which starts seeing the size of the image increasing with the depth decreasing. The addition of skip connections to the architecture ensures the recovery of spatial information that is lost due to the encoding path. This encoder and decoder form a symmetrical U-shape network which is the basis of the name. Due to the impressive results, it became a popular architecture of choice in the task of semantic segmentation (Ronneberger, et al., 2015).

Figure 22.*UNET Architecture*

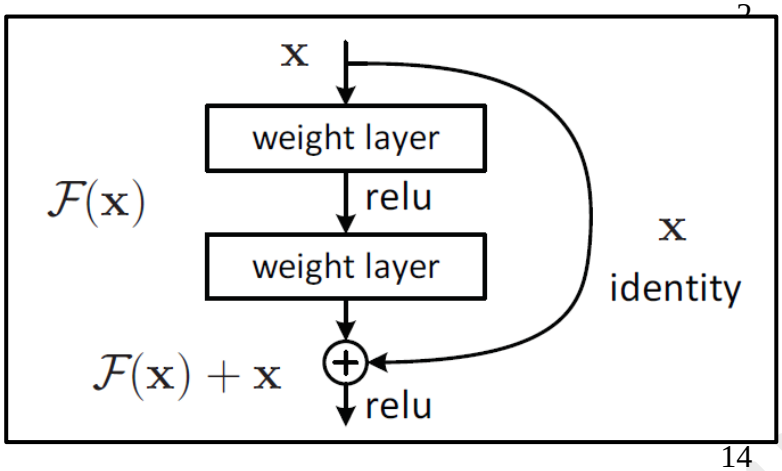


Source: (Ronneberger, et al., 2015)

ResNet

The ResNet (Residual Network) architecture can be utilized as a backbone for the UNET which essentially means it will be the encoder part of the UNET. This is beneficial due to the ability to transfer pre-trained weights into the model rather than starting from scratch which will assist in reaching convergence faster. The ResNet (He, et al., 2016) is a CNN that is made up of residual blocks with skip connections which assist in dealing with the vanishing gradient problem which arises when adding too many layers which decrease the accuracy of the model.

Figure 23. *Residual Network Encoder*



Source: (He, et al., 2016)

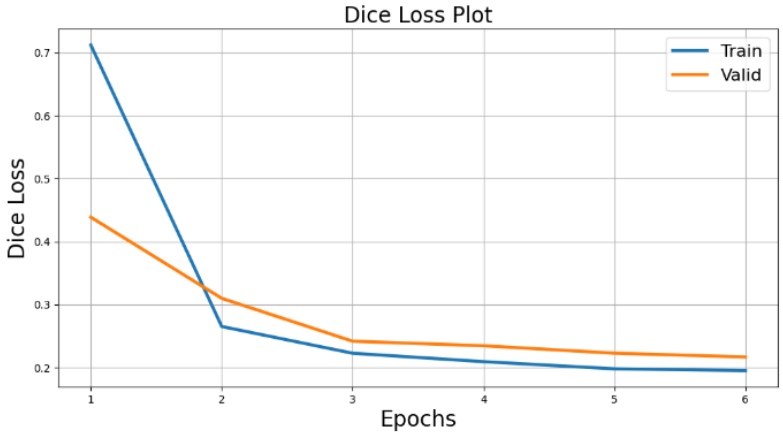
Each 'ResBlock' has two connections coming from the input, one which goes through convolutions, batch normalization, and linear functions and the other is the skip connection known as the identity which will be added together.

Simulation

To test the feasibility and integrity of the information that has been gathered throughout. The design of the autonomous vehicle requires simulations to acquire insight on potential issues that can be identified and opportunities to improve on. Approaching the simulation section will adapt the process similar to an existing project (Theers & Singh, 2020). The platform provides the means to test the feasibility of the UNET architecture (Ronneberger, et al., 2015) with a ResNet encoder (He, et al., 2016) for lane detection as well as the comparing the custom lateral control method which combines strengths of the Pure Pursuit controller and the Stanley controller to reduce the cross-track error of the vehicle. The entire software development was utilized with the Python language which was coded on the Microsoft Visual Studio Code text editor. For the creation of the deep learning model, Deep learning frameworks like PyTorch and Keras are used for the development, compilation, and training of neural networks with the support of utilizing pre-trained weights from a segmentation model package (Yakubovskiy, 2019). Standard python libraries like Open-CV and Pillow are used to implement various image processing techniques used for data augmentation and reshaping of training images. The testing of the model integrated with lateral and longitudinal control methods is achieved through the CARLA simulator (CARLA, n.d.).

Segmentation Model Results

Figure 24. *Segmentation Model Dice Loss Plot*



The dice loss plot in Figure 24 seen above showcases the segmentation model attempting to minimize the error or loss function with the training dataset and the associated validation loss with the validation dataset. During the first two epochs, it can be observed the rate at which the model is learning with the dice loss reducing from around 0.7 to under 0.3. After which the learning rate begins to slow down through the Adam Optimizer (Kingma & Ba, 2014) whereby the model improves slightly and eventually saturates towards the end of the 6th epoch with a dice loss of just under 0.2. A clear observation when comparing the training and validation dataset is the dice loss value is higher with the validation dataset. This is a normal indication with the training model and can be highlighted that the model has trained efficiently since a higher validation loss is expected due to the model parameters not being able to adjust with the validation dataset.

Figure 25. *Segmentation Model Lane Predictions*

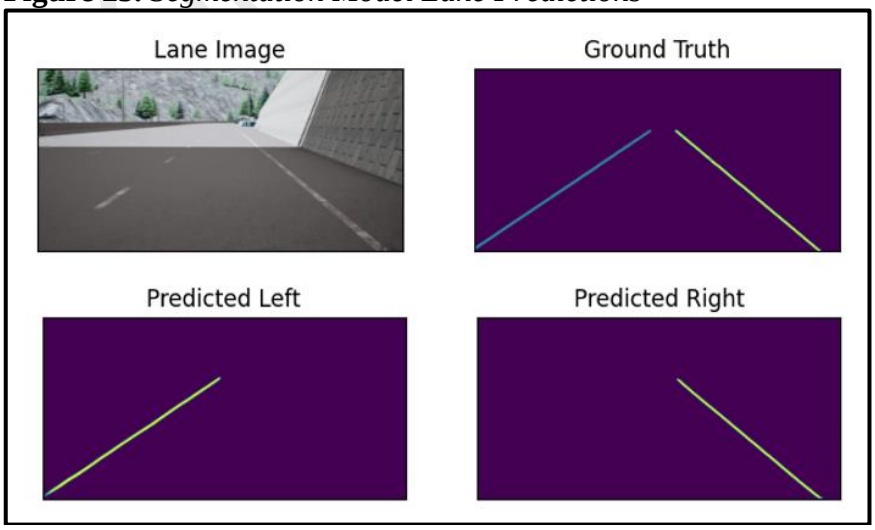
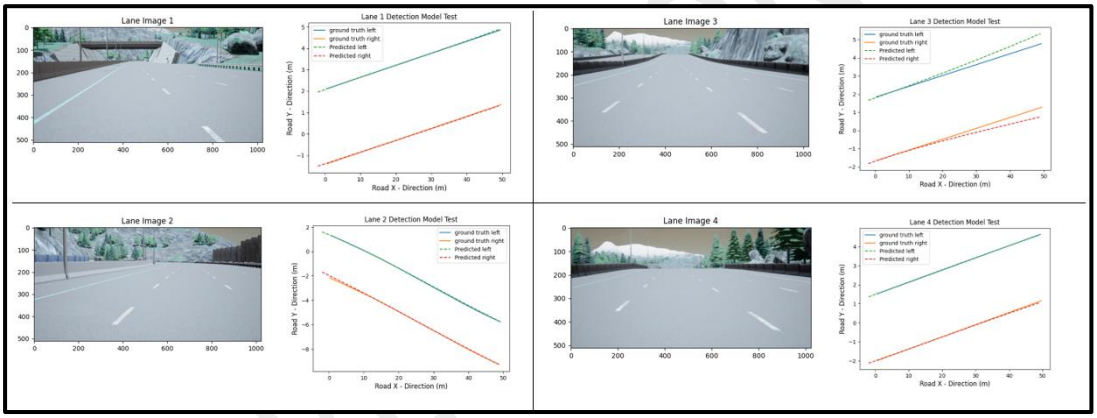


Figure 25 showcases the prediction images from the simulation of the unseen lane images which shows that the segmentation model is determining the lane boundaries fairly accurately. There were however a few obscure lines that were visible during some predictions which could be due to the broken lane boundaries of the lane image. Even so, with broken lines in a number of the images as well as other lane boundaries in the associated image, the model is still able to distinguish the correct lane boundary for which the vehicle is currently driving on. Darker images due to shadowing effects also seem to provide no visible reduction in the model's prediction ability although it is possible that during low light conditions, the model's ability will begin to show signs of detriment which can be alleviated through including low light conditions in the training dataset.

Inverse Perspective Mapping

Figure 26. *Applying Inverse Perspective Mapping*



The prediction of the lane boundaries then proceeds to the most ideal scenario for the model to test in, in which the ground plane is entirely flat allowing inverse perspective mapping techniques to be applied befittingly as shown in Figure 26.

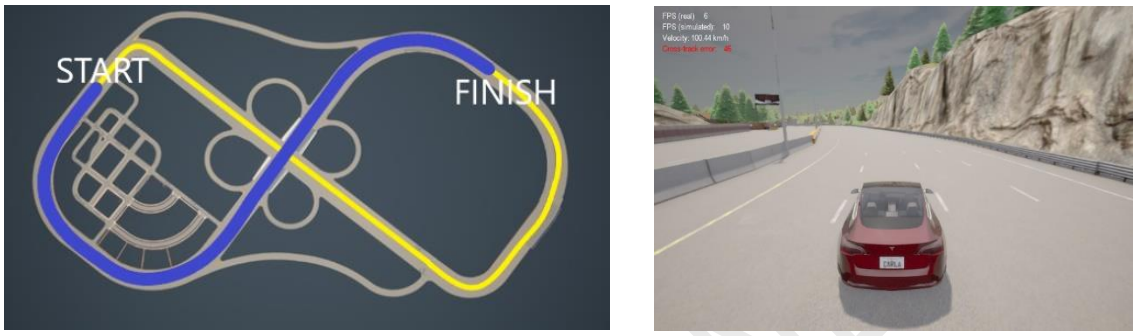
The Lane images 1 and 2 are of a simple road structure with both presenting the most ideal scenario for the model to test in, in which the ground plane is entirely flat allowing inverse perspective mapping techniques to be applied befittingly. The model result together with inverse perspective mapping produces a fairly accurate bird's eye perspective of the road when comparing the lane boundaries to the model predictions.

Lane images 3 and 4 depict a downhill and uphill scenario respectively to observe how the lane detection system will behave in the more likely situations where the ground plane is not entirely flat. Lane image 3 it can be seen the lane is accurately detected to a region of around 20m where there begins to be a drop off in accuracy due to the downhill trajectory with an error increasing to around 0.5m at the end of the 50m limit. In the last lane image where the trajectory is uphill, the system does well in detecting the lane boundaries with

minimal error, however, with a steeper slope, system may observe errors similar to the downhill trajectory as the detection distance approaches the limit.

CARLA Simulation

Figure 27. *Autonomous Vehicle Simulated onto CARLA*



The vehicle was simulated in the CARLA environment with different velocity targets of 60km/h, 80km/h, and 100 km/h whilst attempting to finish the track layout. The cross-track error from both the Pure Pursuit method and the adaptation of the Pure Pursuit and Stanley controller can be seen below in Figure 27. The noticeable observation is the reduction in the cross-track error during the 3 turns. The custom controller does spike during the first seconds due to the vehicle spawning incorrectly but recovers quickly which proves to be a negligible transient.

The rest of the simulation shows both lateral control method simulations performing well during straights with cross-track error increasing during the turns with minimal stability issues although the custom controller does seem to indicate signs of more instability than the Pure Pursuit method. Adjusting the lookaway distance of both controllers (Figure 28) also has been shown to reduce the error at certain velocities which showcase the need for proper calibration. Another observation was the variation of velocity which both methods show increasing cross-track error as the velocity of the vehicle increases. Thus, an improved development can therefore be achieved by varying the velocity of the vehicle during turns.

Figure 28. *Lateral Control Method Comparison*

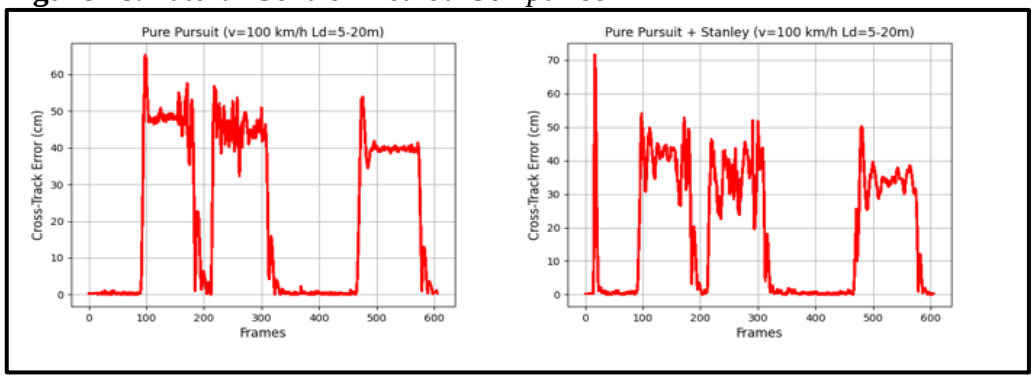
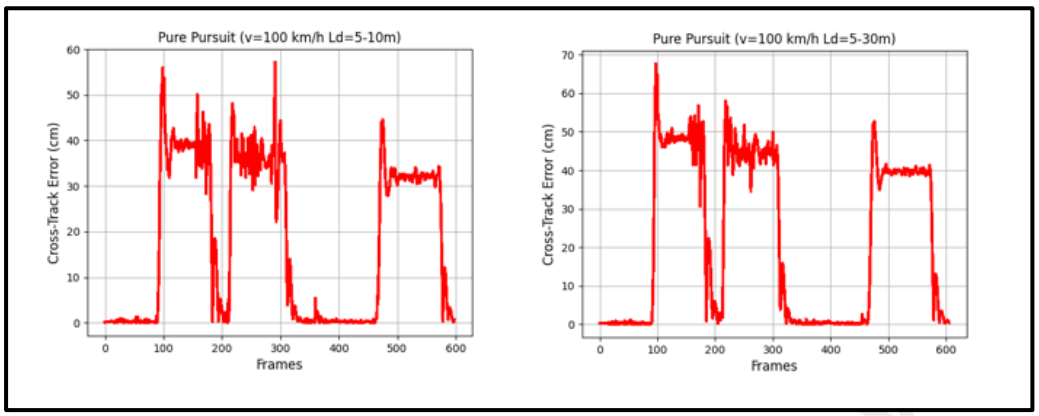


Figure 29. *Adjusting of Lookaway Distance Comparison*



Conclusion

Autonomous vehicles still require improvement with regards to safety and cost. With the report being aimed at how autonomous technology can be used to convert regular vehicles with pre-installed actuation, it became apparent that the platform exists to be improved upon. The results from the simulations showcase that implementing a lane detection system provides a viable option for implementation with opportunities to improve upon in autonomous vehicles. By implementing only a camera attached to the vehicle, the system is capable of detecting lane boundaries fairly accurately under constrained conditions. The CARLA simulations also highlight an improvement with the Pure Pursuit control method with the simulations demonstrating a slight reduction in the cross-track error however, it does decrease the stability of the system which can be alleviated by reducing the speed at corners. Overall, the combination of applying deep learning techniques in the field of computer vision proves to be an exciting platform in the field of autonomous vehicles, with the opportunity of converting regular vehicles into autonomous systems. This integration will also assist in reducing the cost of autonomous vehicles which will prove to improve the overall perception by the public of being exclusive to high-class individuals.

References

Agarap, A. F. M., 2019. Deep Learning using Rectified Linear Units (ReLU).

Campbell, S., Mahony, N. O. & Krpalcova, L., 2018. Sensor Technology in Autonomous Vehicles. *2018 29th Irish Signals and Systems Conference (ISSC)*, pp. 1-4.

CARLA, n.d. *CARLA.* [Online] Available at: https://carla.org/

Dawkins, P., n.d. *Equations of Planes*, s.l.: s.n.

Desborough, H., 2000. *PID Control*, s.l.: s.n.

Ding, Y., 2020. *Three Methods of Vehicle Lateral Control: Pure Pursuit, Stanley and MPC.*

Fei-Fei, Johnson, J. & Yeung, S., 2017. *Detection and Segmentation*.

Google, 2021. *ML Practicum: Image Classification.* [Online] Available at: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

He, K., Zhang, X., Ren, S. & Sun, J., 2016. *Deep Residual Learning for Image Recognition.* s.l., 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Jazar, R. N., n.d. *Vehicle Dynamics: Theory and Applications*.

Jordan, J., 2018. *An overview of semantic image segmentation*.

Kim, P., 2017. *MATLAB Deep Learning*, s.l.: Apress.

Kingma, D. P. & Ba, J. L., 2014. *Adam: A Method for Stochastic Optimization.* s.l., 3rd International Conference for Learning Representations.

Kyriakidis, M., Happee, R. & Winter, J. d., 2015. Public opinion on automated driving: Results of an international questionnaire among 5000 respondents. *Transportation Research Part F: Traffic Psychology and Behaviour*, p. 127–140.

Law, C. K., Dalal, D. & Shearrow, S., 2018. *Robust Model Predictive Control for Autonomous Vehicle/Self-Driving Cars*.

Lee, J.-C., 2010. *Generic Obstacle Detection for Backing-Up Maneuver Safety Based on*, s.l.: Submitted to Institute of Biomedical Engineering College of Computer Science.

Moujahid, A., Hina, M. D., Soukane, A. & Ortalda, A., 2018. *Machine Learning Techniques in ADAS: A Review.* Paris, 2018 International Conference on Advances in Computing and Communication Engineering.

Ronneberger, O., Fischer, P. & Brox, T., 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* s.l., International Conference on Medical Image Computing and Computer-Assisted Intervention.

SAE, 2018. *SAE J3016 "Levels of Driving Automation"*.

Schoettle, B. & Sivak, M., 2014. *A Survey of Public Opinion about Autonomous and Self-Driving Vehicles in the U.S, the U.K and Australia*, Michigan: The University of Michigan Transportation Research Institute.

Sturm, P., n.d. *Geometric Computer Vision.* [Online] Available at: https://hal.inria.fr/cel-02129241/file/poly_3D_eng.pdf

Theers, M. & Singh, M., 2020. *Algorithms for Automated Driving*.

Thrun, S. et al., 2006. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics.*

Vishnukumar, H. J., Müller, D. C., Butting, D. B. & Sax, P. D.-I. E., 2017. *Machine Learning and Deep Neural Network – Artificial Intelligence Core for Lab and Real-World Test and Validation for ADAS and Autonomous Vehicles.* s.l., IEEE.

Yakubovskiy, P., 2019. *Segmentation_models_pytorch.* [Online] Available at: https://segmentation-modelspytorch.readthedocs.io/en/latest/#

Yu, D., Wang, H., Chen, P. & Wei, Z., 2014. *Mixed Pooling for Convolutional Neural Networks.* s.l., The 9th International Conference on Rough Sets and Knowledge Technology.