Chaitanya Bharathi Institute of Technology

Major Project

Virtual Autonomous Vehicle

Guided By Smt.T.Satyakirnamai
Assistant Professor

Project Batch Number: CBIT/IT/2021-22/217
Fawaz Arshed Khan (160118737092)
Saad Mohammed Khaled (160118737100)

TABLE OF CONTENTS

INTRODUCTION & MOTIVATION

With the rapid developments in technology, computers are leaving behind the old notion of 'Computers are dumb machines.' Among all the achievements obtained in various fields of computer science, an autonomous vehicle has been one of the biggest and most important invention. It has been a topic of research for many years and numerous algorithms involving advanced concepts of artificial intelligence. Popularity of these cars are increasing tremendously. Recent studies show that around 10 Million autonomous cars would be on-road in near future.

LITERATURE SURVEY

TITLE	AUTHOR	DESCRIPTION
Simulation of Self-driving Car using Deep Learning (ICISS - 2020)	Aman Bhalla Munipalle Sai Nikhila Pradeep Singh	This paper focuses on development of Artificial Intelligence has revolutionized the area of autonomous vehicles by incorporating complex models and algorithms.
Self-Parking Car Simulation using Reinforcement Learning Approach for Moderate Complexity Parking Scenario (ECTI-CON - 2020)	Baramee Thunyapoo, Chatree Ratchadakorntham,Punnarai Siricharoen and Wittawin Susutti	The auto-parking car simulation framework using proximal policy optimization (PPO) for deep reinforcement learning in a less complex parking scenario which comprises basic and challenging zones for parking
Convolutional Neural Network based Working Model of Self Driving Car (ICESC - 2020)	Chaitra P G Gautami S Deepthi V Suraj H M Prof.Naveen Kumar	This paper discusses hardware and software components of a self driving car that includes usage of technologies such as Deep learning techniques namely Convolution Neural Networks

LITERATURE SURVEY

TITLE	AUTHOR	DESCRIPTION
Simulation of Self-driving System by implementing Digital Twin with GTA 5 (ICEIC - 2021)	Heuijee Yun and Daejin Park	This paper has used OpenCV to capture the GTA 5 game screen and analyzing images with YOLO and TensorFlow based on Python, to build quite an accurate object recognition system and also provide lane departure prevention.
Artificial Intelligence Based Decision Making of Autonomous Vehicles Before Entering Roundabout (IEEE - 2019)	Dávid Tollner Hang Cao Máté Zöldy	This paper focuses on creating standards for autonomous vehicle artificial intelligence based decisions at roundabout entering situation needs deeper understanding of vehicle and traffic behaviour, parametrizing, modelling and simulation

SOFTWARE & HARDWARE REQUIREMENTS

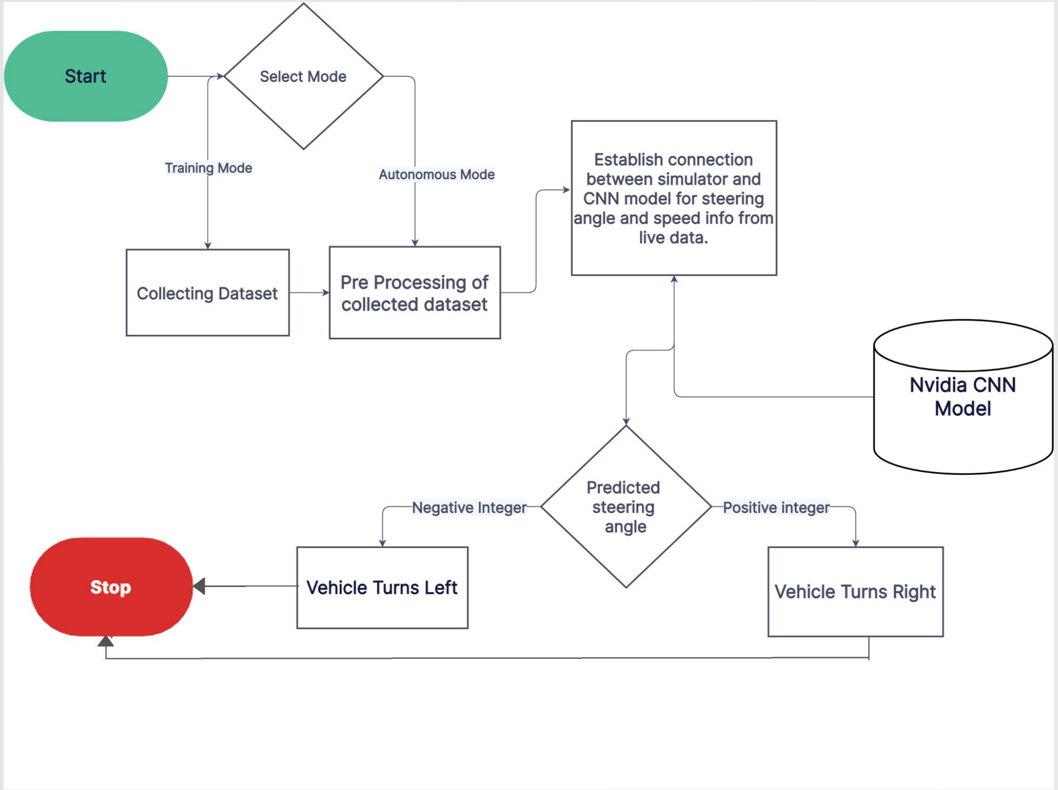
Software Requirements:

- Python Installed
- Google Colab
- Atom
- Udacity Simulator
- Windows/MacOS

Hardware Requirements:

- Processor: i7/Ryzen 7
- RAM: 8Gb and above
- Disk Space: 5Gb and above
- Any basic graphics card

SYSTEM DESIGN & METHODOLOGY



ALGORITHMS

- Lane lines Detection:

    1.Gaussian Blur

    2.Canny Edge Detection

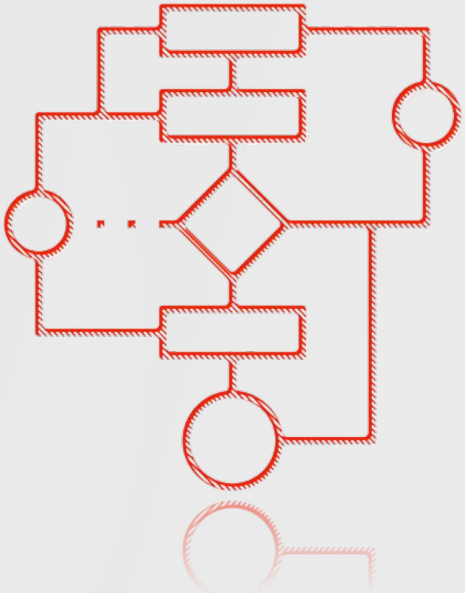
    3.Hough Transformation

- Nvidia Convolutional Neural Network

    4.Elu Activation function

    5.Adam Optimizer

    6.Augmentation Algorithms



Gaussian Blur

In Gaussian Blur operation, the image is convolved with a Gaussian filter instead of the box filter. The Gaussian filter is a low-pass filter that removes the high-frequency components are reduced.

Before Gaussian Blur

After Gaussian Blur

Canny Edge Detection

The Canny edge detection is an edge detection operator that uses multi-stage algorithm to detect wide range of edges in images. It is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed.

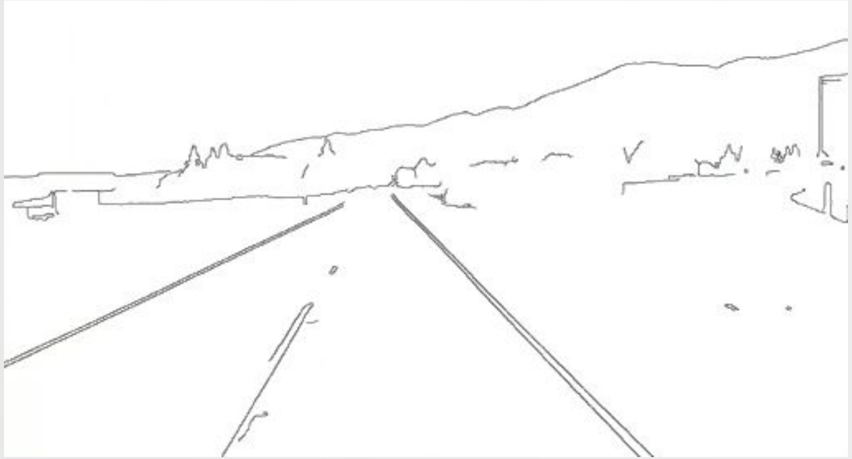
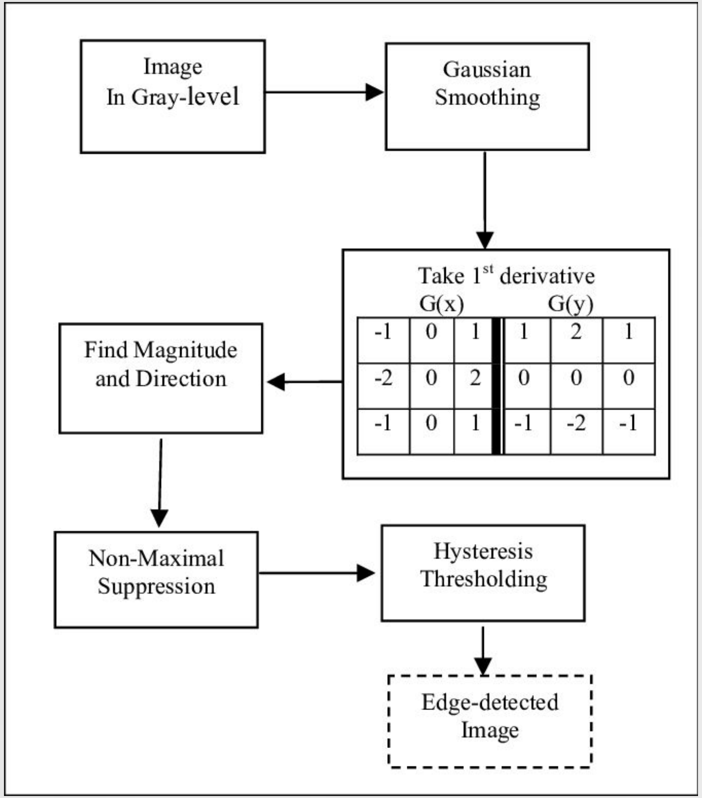


Image Edges

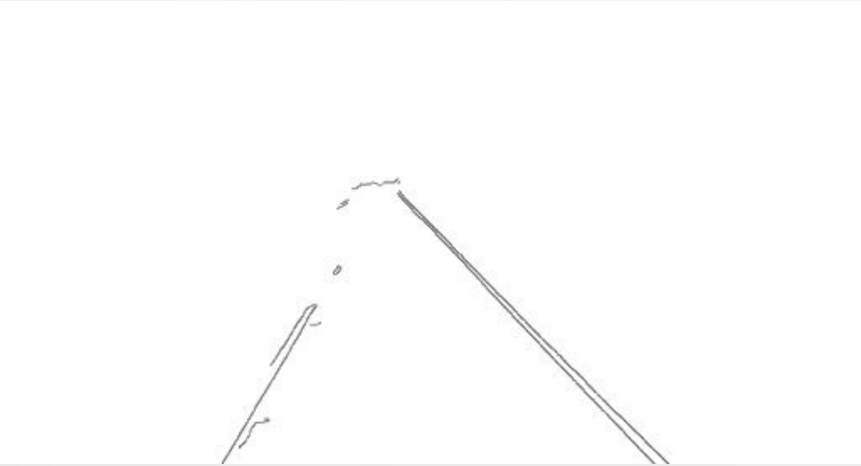


Derivate f(x,y)

Hough Transformation

The point of intersection of two lines in parametric space of Hough Space gives us the coordinates of the line which precisely connects to the two points.

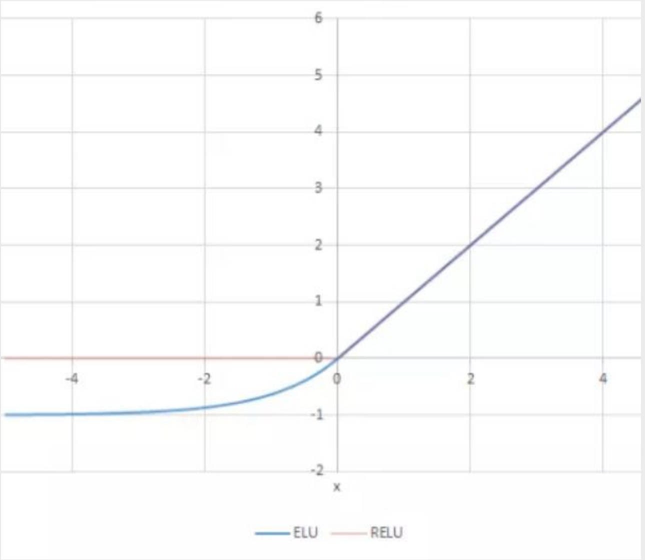
$y = mx + b$



Road Edges

ELU ACTIVATION FUNCTION

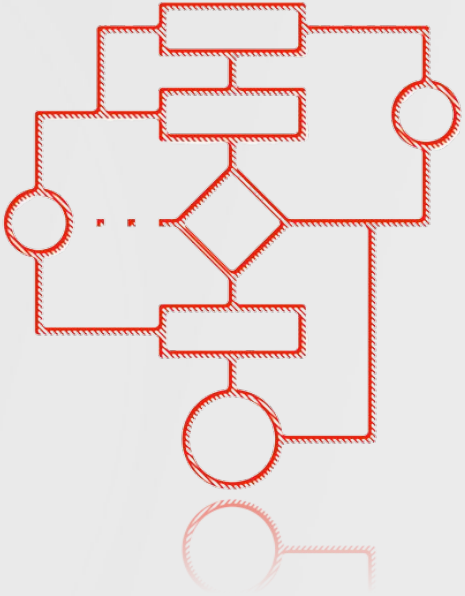
ELU. Exponential Linear Unit or its widely known name ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number. ELU is very similiar to RELU except negative inputs.



$R(z)=\{z0z>0z<=0\}$

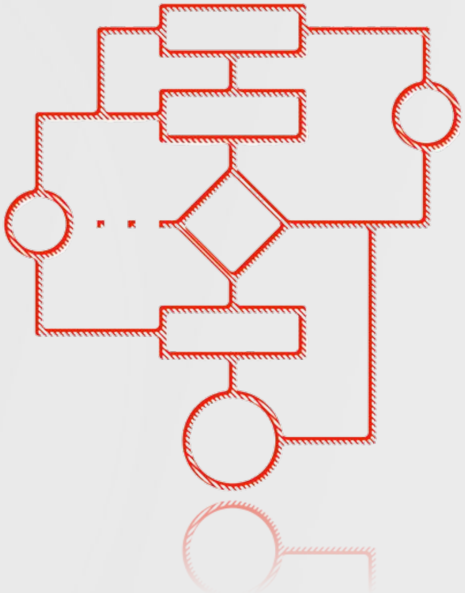
ADAM OPTIMIZER

Adam is an alternative optimization algorithm that provides more efficient neural network weights by running repeated cycles of adaptive moment estimation.



IMGAUG AUGMENTATION

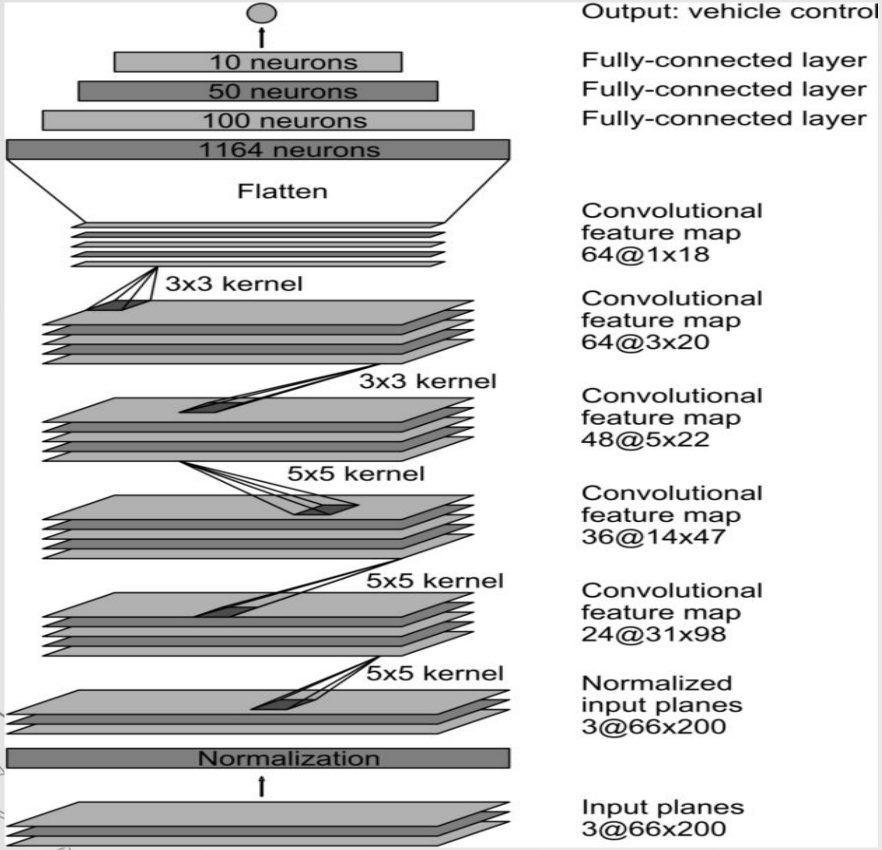
- **ZOOM**
- **PAN**
- **IMAGE RANDOM FLIP**
- **IMAGE RANDOM BRIGHTNESS**



IMPLEMENTATION OF LANE LINE DETECTION

```python
import cv2
import numpy as np

def make_points(image, line):
    slope, intercept = line
    y1 = int(image.shape[0])
    y2 = int(y1*3/5)
    x1 = int((y1 - intercept)/slope)
    x2 = int((y2 - intercept)/slope)
    return [[x1, y1, x2, y2]]

def average_slope_intercept(image, lines):
    left_fit    = []
    right_fit   = []
    if lines is None:
        return None
    for line in lines:
        for x1, y1, x2, y2 in line:
            fit = np.polyfit((x1,x2), (y1,y2), 1)
            slope = fit[0]
            intercept = fit[1]
            if slope < 0:
                left_fit.append((slope, intercept))
            else:
                right_fit.append((slope, intercept))

    left_fit_average  = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)
    left_line  = make_points(image, left_fit_average)
    right_line = make_points(image, right_fit_average)
    averaged_lines = [left_line, right_line]
    return averaged_lines
```

IMPLEMENTATION OF CNN



Nvidia CNN architecture

IMPLEMENTATION OF CNN

```
def nvidia_model():
  model = Sequential()
  model.add(Convolution2D(24, (5,5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
  model.add(Convolution2D(36, (5,5), strides=(2, 2), activation='elu'))
  model.add(Convolution2D(48, (5,5), strides=(2, 2), activation='elu'))
  model.add(Convolution2D(64, (3,3), activation='elu'))

  model.add(Convolution2D(64, 3, 3, activation='elu'))
#   model.add(Dropout(0.5))


  model.add(Flatten())

  model.add(Dense(100, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(50, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(10, activation = 'elu'))
#   model.add(Dropout(0.5))

  model.add(Dense(1))

  optimizer = Adam(lr=1e-3)
  model.compile(loss='mse', optimizer=optimizer)
  return model
model = nvidia_model()
print(model.summary())
history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),
                              steps_per_epoch=300
```

IMPLEMENTATION OF CNN

```
                                    steps_per_epoch=300,
                                    epochs=10,
                                    validation_data=batch_generator(X_valid, y_valid, 100, 0),
                                    validation_steps=200,
                                    verbose=1,
                                    shuffle = 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training', 'validation'])
plt.title('Loss')
plt.xlabel('Epoch')
```

IMPLEMENTATION OF CLIENT-SERVER CONNECTION USING FLASK AND SOCKET.IO

drive.py

```python
import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2

sio = socketio.Server()

app = Flask(__name__) #'__main__'
speed_limit = 10
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img


@sio.on('telemetry')
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = img_preprocess(image)
    image = np.array([image])
    steering_angle = float(model.predict(image))
    throttle = 1.0 - speed/speed_limit
    print('{} {} {}'.format(steering_angle, throttle, speed))
    send_control(steering_angle, throttle)

```

IMPLEMENTATION OF CLIENT-SERVER CONNECTION USING FLASK AND SOCKET.IO

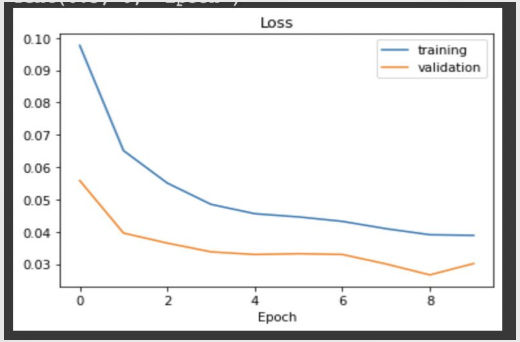
```python


@sio.on('connect')
def connect(sid, environ):
    print('Connected')
    send_control(0, 0)

def send_control(steering_angle, throttle):
    sio.emit('steer', data = {
        'steering_angle': steering_angle.__str__(),
        'throttle': throttle.__str__()
    })


if __name__ == '__main__':
    model = load_model('model.h5')
    app = socketio.Middleware(sio, app)
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)

```

TESTING OF CNN

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 31, 98, 24)        1824

 conv2d_1 (Conv2D)           (None, 14, 47, 36)        21636

 conv2d_2 (Conv2D)           (None, 5, 22, 48)         43248

 conv2d_3 (Conv2D)           (None, 3, 20, 64)         27712

 conv2d_4 (Conv2D)           (None, 1, 6, 64)          36928

 flatten (Flatten)           (None, 384)               0

 dense (Dense)               (None, 100)               38500

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 10)                510

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 175,419
Trainable params: 175,419
Non-trainable params: 0
_________________________________________________________________
```

VALIDATION OF CNN

```
Epoch 1/10
300/300 [==============================] - 225s 708ms/step - loss: 0.0976 - val_loss: 0.0558
Epoch 2/10
300/300 [==============================] - 213s 713ms/step - loss: 0.0651 - val_loss: 0.0396
Epoch 3/10
300/300 [==============================] - 211s 704ms/step - loss: 0.0551 - val_loss: 0.0365
Epoch 4/10
300/300 [==============================] - 209s 698ms/step - loss: 0.0485 - val_loss: 0.0338
Epoch 5/10
300/300 [==============================] - 212s 710ms/step - loss: 0.0456 - val_loss: 0.0330
Epoch 6/10
300/300 [==============================] - 213s 713ms/step - loss: 0.0446 - val_loss: 0.0332
Epoch 7/10
300/300 [==============================] - 212s 707ms/step - loss: 0.0432 - val_loss: 0.0330
Epoch 8/10
300/300 [==============================] - 213s 711ms/step - loss: 0.0410 - val_loss: 0.0300
Epoch 9/10
300/300 [==============================] - 212s 707ms/step - loss: 0.0391 - val_loss: 0.0267
Epoch 10/10
300/300 [==============================] - 216s 723ms/step - loss: 0.0389 - val_loss: 0.0302
Text(0.5, 0, 'Epoch')
```

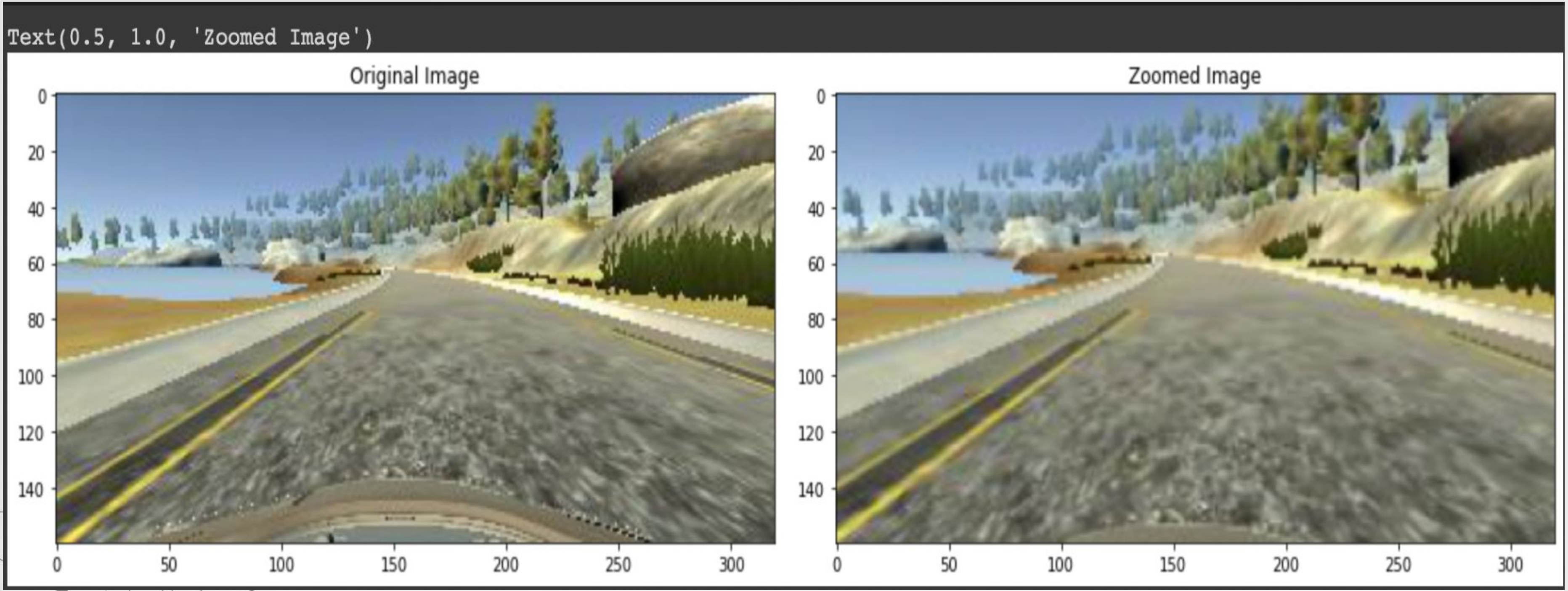


TESTING OF ZOOM AUGMENTATION

```
[16] def zoom(image):
         zoom = iaa.Affine(scale=(1, 1.3))
         image = zoom.augment_image(image)
         return image
```

```
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
zoomed_image = zoom(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(zoomed_image)
axs[1].set_title('Zoomed Image')
```

RESULTS OF ZOOM AUGMENTATION

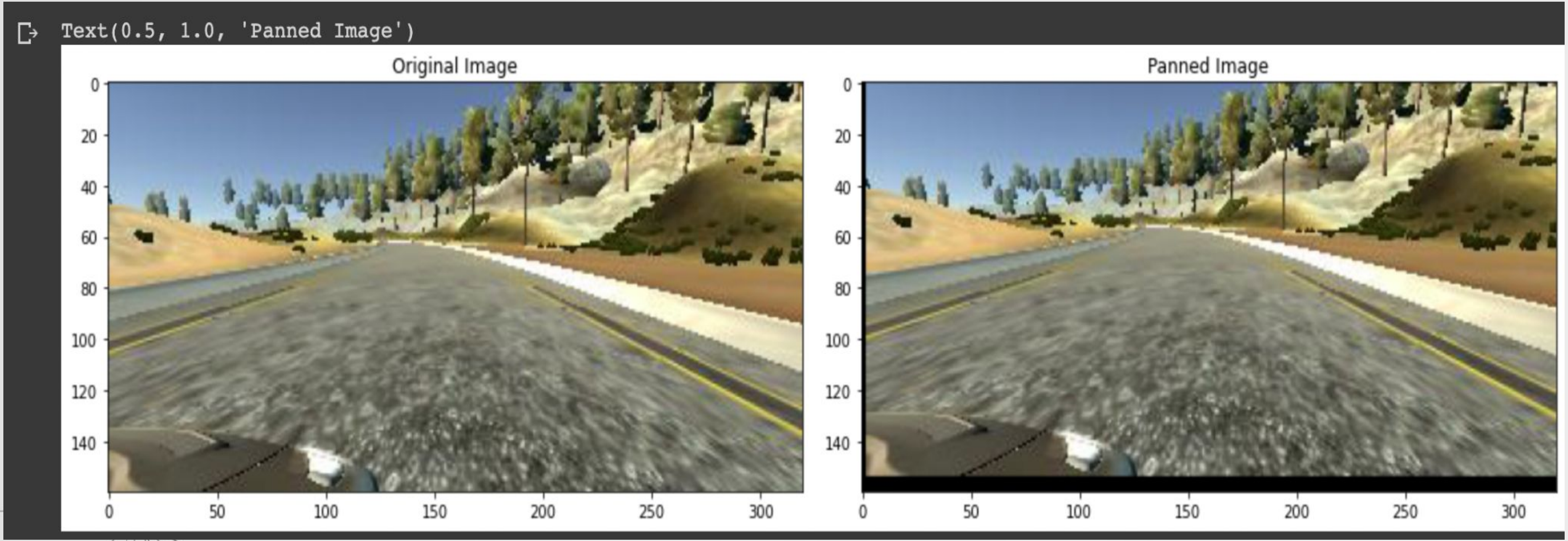


TESTING OF PAN AUGMENTATION

```
[18] def pan(image):
       pan = iaa.Affine(translate_percent= {"x" : (-0.1, 0.1), "y": (-0.1, 0.1)})
       image = pan.augment_image(image)
       return image
```

```
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
panned_image = pan(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(panned_image)
axs[1].set_title('Panned Image')
```

RESULTS OF PAN AUGMENTATION

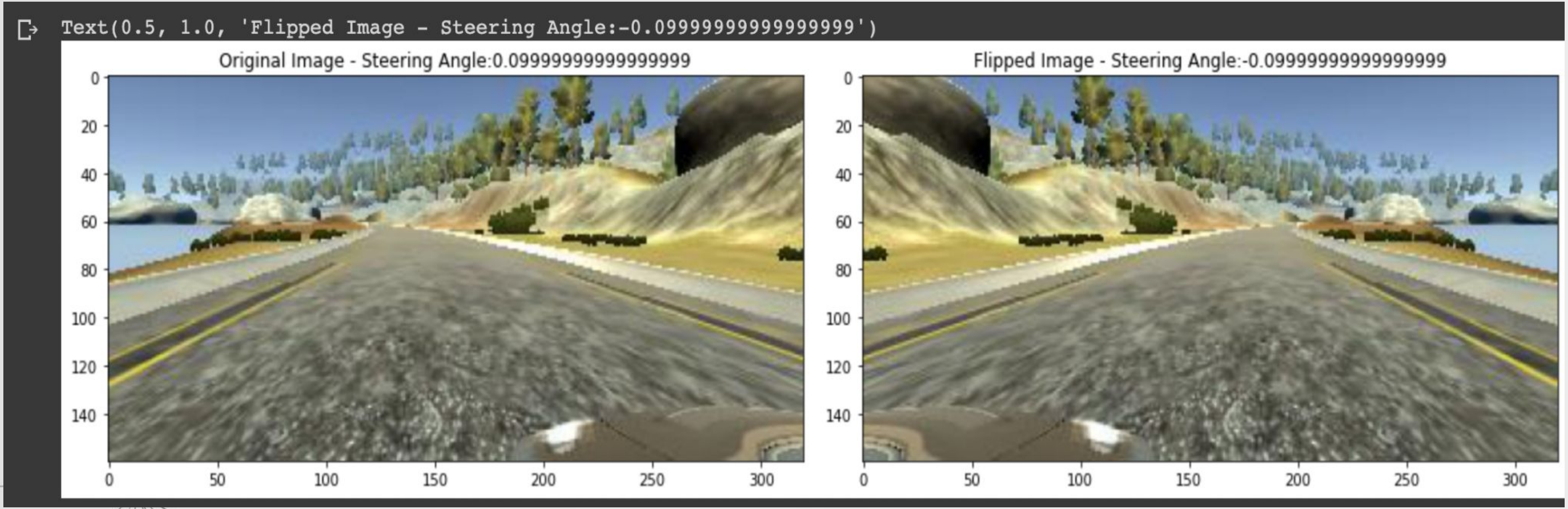


TESTING OF IMG FLIP AUGMENTATION

```
[20] def img_random_flip(image, steering_angle):
         image = cv2.flip(image,1)
         steering_angle = -steering_angle
         return image, steering_angle
```

```
random_index = random.randint(0, 1000)
image = image_paths[random_index]
steering_angle = steerings[random_index]


original_image = mpimg.imread(image)
flipped_image, flipped_steering_angle = img_random_flip(original_image, steering_angle)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image - ' + 'Steering Angle:' + str(steering_angle))

axs[1].imshow(flipped_image)
axs[1].set_title('Flipped Image - ' + 'Steering Angle:' + str(flipped_steering_angle))
```

RESULTS OF FLIP AUGMENTATION

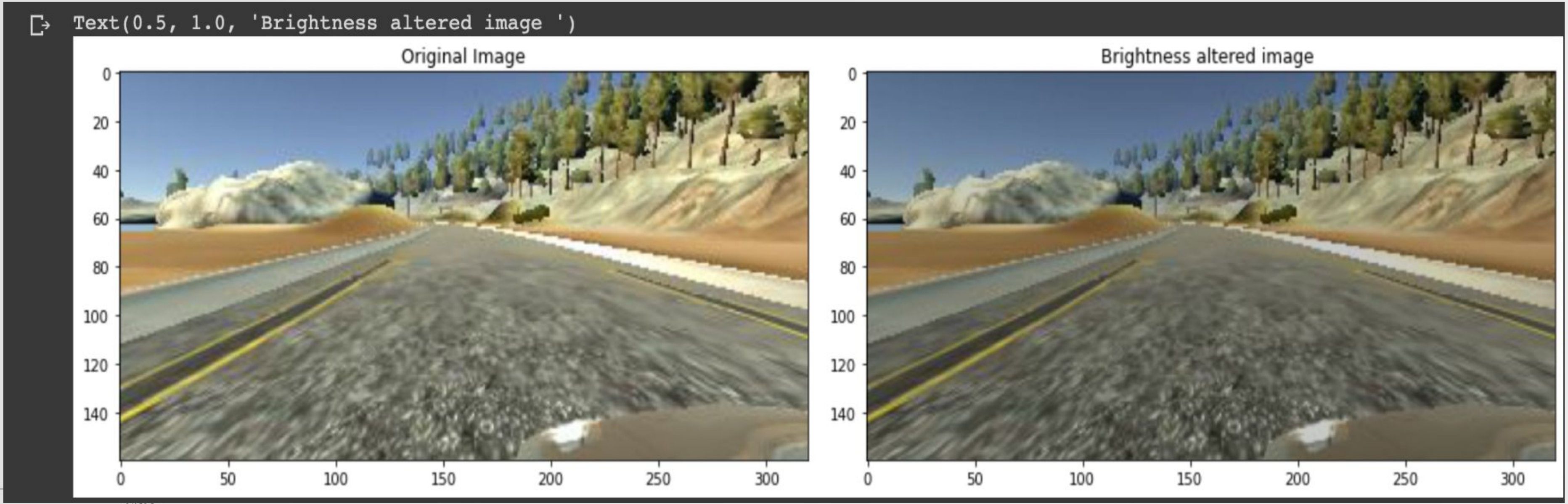


TESTING OF RANDOM BRIGHTNESS AUGMENTATION

```
[ ] def img_random_brightness(image):
        brightness = iaa.Multiply((0.2, 1.2))
        image = brightness.augment_image(image)
        return image
```

```
image = image_paths[random.randint(0, 1000)]
original_image = mpimg.imread(image)
brightness_altered_image = img_random_brightness(original_image)

fig, axs = plt.subplots(1, 2, figsize=(15, 10))
fig.tight_layout()

axs[0].imshow(original_image)
axs[0].set_title('Original Image')

axs[1].imshow(brightness_altered_image)
axs[1].set_title('Brightness altered image ')
```

RESULTS OF RANDOM BRIGHTNESS AUGMENTATION



RESULT

FUTURE SCOPE & CONCLUSION

The technology underlying semi- and fully autonomous vehicles is well-developed and poised for commercial deployment. Major automotive companies and software developers have made considerable progress in navigation, collision avoidance, and street mapping. Addressing relevant issues and making sure regulatory rules are clear should be high priorities in all the countries considering autonomous vehicles.

BIBLIOGRAPHY

1. K. Sama et al., "Extracting Human-Like Driving Behaviors From Expert Driver Data Using Deep Learning," in IEEE Transactions on Vehicular Technology, vol. 69, no. 9, pp. 9315-9329, Sept. 2020, doi: 10.1109/TVT.2020.2980197.

2. G. Prabhakar, B. Kailath, S. Natarajan and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," 2017 IEEE Region 10 Symposium (TENSYMP), 2017, pp. 1-6, doi: 10.1109/TENCONSpring.2017.8069972.

3. D. Li, D. Zhao, Q. Zhang and Y. Chen, "Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving [Application Notes]," in IEEE Computational Intelligence Magazine, vol. 14, no. 2, pp. 83-98, May 2019, doi: 10.1109/MCI.2019.2901089.

4. Hironobu Fujiyoshi, Tsubasa Hirakawa, Takayoshi Yamashita, Deep learning-based image recognition for autonomous driving, IATSS Research, Volume 43, Issue 4, 2019, Pages 244-252, ISSN 0386-1112, https://doi.org/10.1016/j.iatssr.2019.11.008.

5. Q. Rao and J. Frtunikj, "Deep Learning for Self-Driving Cars: Chances and Challenges," 2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS), 2018, pp. 35-38.