TITLE: SELF DRIVING CAR

ABSTRACT:

In this project a simple self driving car is designed and implemented. The concept of the project was inspired by the recent surge in automated car industry. The designed car was capable of detecting the road signals and taking the right turn accordingly. To implement the whole system, the body of the car was connected to the analyzer computer via wifi where the computer can analyze the feed video frame by frame. In a real car the analyzer computer can be simply mounted on board. The whole system was capable of taking right decision with excellent accuracy.

OBJECTIVE:

The objective is not to free the driver from control of the car for whatever marketing reason sells the idea.
The objective is to monitor the driver through an internet of things that produce data about the individual driver and all drivers collectively.That data is input to monitoring systems that generate information about the driver and allow drivers that becomes valuable. Money is to be made on the value of that information. This objective is what will make driving safer.

INTRODUCTION:

A self-driving car can analyze surrounding without any human interactions and take decisions accordingly without any human interactions. A number of sensors are combined and are used to identify the pathway and road signal from the surroundings. An autonomous car is has reduced costs due to less wastage of fuel, increased safety, increased mobility, increased customer satisfaction and that's why it has more advantage than traditional cars. The biggest benefit of using a self-driving car is significantly fewer traffic accidents. More than 90% of all accidents are caused by some degree of human error, including distraction, impaired driving, or poor decision making. With self-driving cars making

decisions and communicating with one another, the number of accidents should reduce.

METHODOLOGY

Hybrid navigation is the simultaneous use of more than one navigation system for location data determination, needed for navigation.

Sensing

To reliably and safely operate an autonomous vehicle, usually a mixture of sensors is utilized. Typical sensors include lidar (Light Detection and Ranging), stereo vision, GPS and IMU. Modern self-driving cars generally use Bayesian simultaneous localization and mapping (SLAM) algorithms, which fuse data from multiple sensors and an off-line map into current location estimates and map updates. Waymo has developed a variant of SLAM with detection and tracking of other moving objects (DATMO), which also handles obstacles such as cars and pedestrians. Simpler

systems may use roadside real-time locating system (RTLS) technologies to aid localization.

Maps

Self-driving cars require a new class of high-definition maps (HD maps) that represent the world at up to two orders of magnitude more detail. In May 2018, researchers from the Massachusetts Institute of Technology (MIT) announced that they had built an automated car that can navigate unmapped roads. Researchers at their Computer Science and Artificial Intelligence Laboratory (CSAIL) have developed a new system, called MapLite, which allows self-driving cars to drive on roads that they have never been on before, without using 3D maps. The system combines the GPS position of the vehicle, a "sparse topological map" such as OpenStreetMap (i.e. having 2D features of the roads only), and a series of sensors that observe the road conditions.

Sensorfusion

Control systems on automated cars may use sensor fusion, which is an approach that integrates information from a variety of sensors on the car to produce a more consistent, accurate, and useful view of the environment. Self-driving cars tend to use a combination of cameras, LiDAR sensors, and radar sensors in order to enhance performance and ensure the safety of the passenger and other drivers on the road. An increased consistency in self-driving performance prevents accidents that may occur because of one faulty sensor.

Pathplanning

Path planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. Self-driving cars rely on path planning technology in order to follow the rules of traffic and prevent accidents from occurring. The large scale path of the vehicle can be determined by using a voronoi diagram, an occupancy grid mapping, or with a driving corridors algorithm. A driving corridors algorithm allows the vehicle to locate and drive within open free space that is bounded by lanes or barriers. While these algorithms work in a simple situation, path planning has not been proven to be effective in a complex scenario. Two techniques used for path planning are

graph-based search and variational-based optimization techniques. Graph-based techniques can make harder decisions such as how to pass another vehicle/obstacle. Variational-based optimization techniques require a higher level of planning in setting restrictions on the vehicle's driving corridor to prevent collisions.

Drive by wire

Drive by wire technology in the automotive industry is the use of electrical or electro-mechanical systems for performing vehicle functions traditionally achieved by mechanical linkages.

Driver monitoring system

Driver monitoring system is a vehicle safety system to assess the driver's alertness and warn the driver if needed. It is recognized in developer side that the role of the systems will increase as SAE Level 2 systems become more common-place, and becomes more challenging at Level 3 and above to predict the driver's readiness for handover.

Vehicular communication

Vehicular communications is a growing area of communications between vehicles and including roadside communication infrastructure. Vehicular communication systems use vehicles and roadside units as the communicating nodes in a peer-to-peer network, providing each other with information. This connectivity enables autonomous vehicles to interact with non-autonomous traffic and pedestrians to increase safety. And autonomous vehicles will need to connect to the cloud to update their software and maps, and feedback information to improve the used maps and software of their manufacturer.

Re-programmable

Autonomous vehicles have software systems that drive the vehicle, meaning that updates through reprogramming or editing the software can enhance the benefits of the owner (e.g. update in better distinguishing blind person vs. non-blind person so that the vehicle will take extra caution when approaching a blind person). A characteristic of this re-programmable part of autonomous vehicles is that the updates need not only to come from the supplier, because through machine learning, smart autonomous vehicles can generate certain updates and install them accordingly (e.g. new navigation maps or new intersection computer systems). These reprogrammable characteristics of the digital technology and the possibility of smart machine learning give manufacturers of autonomous vehicles the opportunity to differentiate themselves on software.

In March 2021, UNECE regulation on software update and software update management system was published.

Modularity

Autonomous vehicles are more modular since they are made up out of several modules which will be explained hereafter through a Layered Modular Architecture. The Layered Modular Architecture extends the architecture of purely physical vehicles by incorporating four loosely coupled layers of devices, networks, services and contents into Autonomous Vehicles. These loosely coupled layers can interact through certain standardized interfaces.

1. The first layer of this architecture consists of the device layer. This layer consists of the following two parts: logical capability and physical machinery. The physical machinery refers to the actual vehicle itself (e.g. chassis and carrosserie). When it comes to digital technologies, the physical machinery is accompanied by a logical capability layer in the form of operating systems that helps to guide the vehicles itself and make it autonomous. The logical capability provides control over the vehicle and connects it with the other layers;
2. On top of the device layer comes the network layer. This layer also consists of two different parts: physical transport and logical transmission. The physical transport layer refers to the radars, sensors and cables of the autonomous vehicles which enable the transmission of digital information. Next to that, the network layer of autonomous vehicles also has a logical transmission which contains communication protocols and network standard to communicate the digital information with other networks and platforms or between layers. This increases the accessibility of the autonomous vehicles and enables the computational power of a network or platform;
3. The service layer contains the applications and their functionalities that serves the autonomous vehicle (and its owners) as they extract, create, store and consume content with regards to their own driving history, traffic congestion, roads or parking abilities for example.;
4. The final layer of the model is the contents layer. This layer contains the sounds, images and videos. The autonomous vehicles store, extract and use to act upon and improve their driving and understanding of the environment. The contents layer also provides metadata and directory information about the content's origin, ownership, copyright, encoding methods, content tags, Geo-time stamps, and so on (Yoo et al., 2010).

Homogenization

In order for autonomous vehicles to perceive their surroundings, they have to use different techniques each with their own accompanying digital information (e.g. radar, GPS, motion sensors and computer vision). Homogenization requires that the digital information from these different sources is transmitted and stored in the same form. This means their differences are decoupled, and digital information can be transmitted, stored, and computed in a way that the vehicles and their operating system can better understand and act upon it. In international standardization field, ISO/TC 22 is in charge of in-vehicle transport information and control systems, and ISO/TC 204 is in charge of information, communication and control systems in the field of urban and rural surface transportation. International standards have been actively developed in the domains of AD/ADAS functions, connectivity, human interaction, in-vehicle systems, management/engineering, dynamic map and positioning, privacy and security.

Mathematical safety model

In 2017, Mobileye published a mathematical model for automated vehicle safety which is called "Responsibility-Sensitive Safety (RSS)". It is under standardization at IEEE Standards Association as "IEEE P2846: A Formal Model for Safety Considerations in Automated Vehicle Decision Making".[

In 2022, a research group of National Institute of Informatics (NII, Japan) expanded RSS and developed "Goal-Aware RSS" to make RSS rules possible to deal with complex scenarios via program logic.

CODE:

#driving_data

```
import cv2
import random
import numpy as np
xs = []
ys = []
#points to the end of the last batch
train_batch_pointer = 0
val_batch_pointer = 0
#read data.txt
```

```
with open("driving_dataset/data.txt") as f:
    for line in f:
        xs.append("driving_dataset/" + line.split()[0])
        #the paper by Nvidia uses the inverse of the turning radius,
        #but steering wheel angle is proportional to the inverse of
turning radius
        #so the steering wheel angle in radians is used as the output
        ys.append(float(line.split()[1]) * 3.14159265 / 180)
#get number of images
num_images = len(xs)
#shuffle list of images
c = list(zip(xs, ys))
random.shuffle(c)
xs, ys = zip(*c)
train_xs = xs[:int(len(xs) * 0.8)]
train_ys = ys[:int(len(xs) * 0.8)]
val_xs = xs[-int(len(xs) * 0.2):]
val_ys = ys[-int(len(xs) * 0.2):]
num_train_images = len(train_xs)
num_val_images = len(val_xs)
def LoadTrainBatch(batch_size):
    global train_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):

x_out.append(cv2.resize(cv2.imread(train_xs[(train_batch_pointer + i) %
num_train_images])[-150:], (200, 66)) / 255.0)
        y_out.append([train_ys[(train_batch_pointer     +     i)     %
num_train_images]])
    train_batch_pointer += batch_size
    return x_out, y_out
def LoadValBatch(batch_size):
    global val_batch_pointer
    x_out = []
    y_out = []
    for i in range(0, batch_size):
        x_out.append(cv2.resize(cv2.imread(val_xs[(val_batch_pointer  +
i) % num_val_images])[-150:], (200, 66)) / 255.0)
        y_out.append([val_ys[(val_batch_pointer + i) % num_val_images]])
    val_batch_pointer += batch_size
    return x_out, y_out
```

#model

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import scipy
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)
def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
def conv2d(x, W, stride):
  return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1],
padding='VALID')
x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])
y_ = tf.placeholder(tf.float32, shape=[None, 1])
```

```
x_image = x
#first convolutional layer
W_conv1 = weight_variable([5, 5, 3, 24])
b_conv1 = bias_variable([24])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 2) + b_conv1)
#second convolutional layer
W_conv2 = weight_variable([5, 5, 24, 36])
b_conv2 = bias_variable([36])
h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, 2) + b_conv2)
#third convolutional layer
W_conv3 = weight_variable([5, 5, 36, 48])
b_conv3 = bias_variable([48])
h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 2) + b_conv3)
#fourth convolutional layer
W_conv4 = weight_variable([3, 3, 48, 64])
b_conv4 = bias_variable([64])
h_conv4 = tf.nn.relu(conv2d(h_conv3, W_conv4, 1) + b_conv4)
#fifth convolutional layer
W_conv5 = weight_variable([3, 3, 64, 64])
b_conv5 = bias_variable([64])
h_conv5 = tf.nn.relu(conv2d(h_conv4, W_conv5, 1) + b_conv5)
#FCL 1
W_fc1 = weight_variable([1152, 1164])
b_fc1 = bias_variable([1164])
h_conv5_flat = tf.reshape(h_conv5, [-1, 1152])
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
#FCL 2
W_fc2 = weight_variable([1164, 100])
b_fc2 = bias_variable([100])
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)
#FCL 3
W_fc3 = weight_variable([100, 50])
b_fc3 = bias_variable([50])
h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)
h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)
#FCL 3
W_fc4 = weight_variable([50, 10])
b_fc4 = bias_variable([10])
h_fc4 = tf.nn.relu(tf.matmul(h_fc3_drop, W_fc4) + b_fc4)
h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)
#Output
W_fc5 = weight_variable([10, 1])
b_fc5 = bias_variable([1])
y = tf.multiply(tf.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2)
#scale the atan output
```

#run

```
import tensorflow.compat.v1 as tf

tf.disable_v2_behavior()
import model
import cv2
from subprocess import call
import os
#check if on windows OS
```

```
windows = False
if os.name == 'nt':
    windows = True
sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")
img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape
smoothed_angle = 0
cap = cv2.VideoCapture(0)
while(cv2.waitKey(10) != ord('q')):
    ret, frame = cap.read()
    image = cv2.resize(frame, (200, 66)) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image],
model.keep_prob: 1.0})[0][0] * 180 / 3.14159265
    if not windows:
        call("clear")
    print("Predicted steering angle: " + str(degrees) + " degrees")
    cv2.imshow('frame', frame)
    #make smooth angle transitions by turning the steering wheel based
on the difference of the current angle
    #and the predicted angle
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 /
3.0) * (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)
cap.release()
cv2.destroyAllWindows()
```

#run_dataset

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import model
import cv2
from subprocess import call
import os
#check if on windows OS
windows = False
if os.name == 'nt':
    windows = True
sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")
img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape
smoothed_angle = 0
i = 0
while(cv2.waitKey(10) != ord('q')):
    full_image = cv2.imread("driving_dataset/" + str(i) + ".jpg")
    image = cv2.resize(full_image[-150:], (200, 66)) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob:
1.0})[0][0] * 180.0 / 3.14159265
    if not windows:
        call("clear")
    print("Predicted steering angle: " + str(degrees) + " degrees")
    cv2.imshow("frame", full_image)
```

```
    #make smooth angle transitions by turning the steering wheel based
on the difference of the current angle
    #and the predicted angle
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 /
3.0) * (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)
    i += 1
cv2.destroyAllWindows()
```

#train

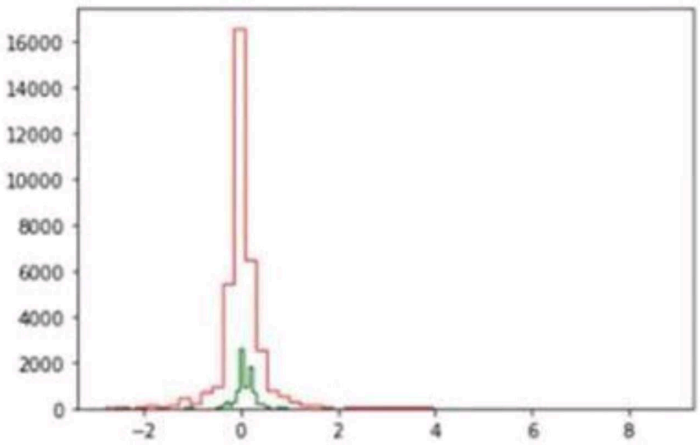
```
import os

import tensorflow.compat.v1 as tf

tf.disable_v2_behavior()
from tensorflow.core.protobuf import saver_pb2
import driving_data
import model
LOGDIR = './save'
sess = tf.InteractiveSession()
L2NormConst = 0.001
train_vars = tf.trainable_variables()
loss  =  tf.reduce_mean(tf.square(tf.subtract(model.y_,  model.y)))  +
tf.add_n([tf.nn.l2_loss(v) for v in train_vars]) * L2NormConst
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.global_variables_initializer())
# create a summary to monitor cost tensor
tf.summary.scalar("loss", loss)
# merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
saver = tf.train.Saver(write_version = saver_pb2.SaverDef.V2)
# op to write logs to Tensorboard
logs_path = './logs'
summary_writer            =            tf.summary.FileWriter(logs_path,
graph=tf.get_default_graph())
epochs = 30
batch_size = 100
# train over the dataset about 30 times
for epoch in range(epochs):
  for i in range(int(driving_data.num_images/batch_size)):
    xs, ys = driving_data.LoadTrainBatch(batch_size)
    train_step.run(feed_dict={model.x:      xs,      model.y_:      ys,
model.keep_prob: 0.8})
    if i % 10 == 0:
      xs, ys = driving_data.LoadValBatch(batch_size)
      loss_value  =  loss.eval(feed_dict={model.x:xs,  model.y_:  ys,
model.keep_prob: 1.0})
      print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_size
+ i, loss_value))
    # write logs at every iteration
    summary  =  merged_summary_op.eval(feed_dict={model.x:xs,  model.y_:
ys, model.keep_prob: 1.0})
    summary_writer.add_summary(summary,             epoch             *
driving_data.num_images/batch_size + i)
```

```
    if i % batch_size == 0:
      if not os.path.exists(LOGDIR):
        os.makedirs(LOGDIR)
      checkpoint_path = os.path.join(LOGDIR, "model.ckpt")
      filename = saver.save(sess, checkpoint_path)
  print("Model saved in file: %s" % filename)
print("Run the command line:\n" \
      "--> tensorboard --logdir=./logs " \
       "\nThen open http://0.0.0.0:6006/ into your web browser")
```

```
Out[18]: (array([ 26.,  12.,  23.,  23.,  16.,  25.,   7.,   4.,   7.,
                   3.,   3.,   5.,   4.,   4.,   4.,   5.,   5.,  17.,
                  18.,  21.,  67.,  41.,  17.,  19.,  15.,  15.,  16.,
                  24.,  25., 152., 280., 126., 325., 759., 2609.,  944.,
                1806., 727., 245., 258., 132.,  30.,  20.,  53.,  75.,
                  46.,   6.,   4.,   5.,   9.]),
          array([-2.78989   , -2.70848027, -2.62707053, -2.5456608 , -2.46425107,
                 -2.38284133, -2.3014316 , -2.22002187, -2.13861213, -2.0572024 ,
                 -1.97579267, -1.89438293, -1.8129732 , -1.73156347, -1.65015373,
                 -1.568744  , -1.48733427, -1.40592453, -1.3245148 , -1.24310507,
                 -1.16169533, -1.0802856 , -0.99887587, -0.91746613, -0.8360564 ,
                 -0.75464667, -0.67323693, -0.5918272 , -0.51041747, -0.42900773,
                 -0.347598  , -0.26618827, -0.18477853, -0.1033688 , -0.02195907,
                  0.05945067,  0.1408604 ,  0.22227013,  0.30367987,  0.3850896 ,
                  0.46649933,  0.54790907,  0.6293188 ,  0.71072853,  0.79213827,
                  0.873548  ,  0.95495773,  1.03636747,  1.1177772 , -1.19918693,
                  1.28059667]),
          [<matplotlib.patches.Polygon at 0x1d692c02f10>])
```



CONCLUSION:

Overall, autonomous cars are good for society because it will give us more time, less accidents, and continue to advance in technology. Self-driving cars have the possibility of preventing 90% of car accidents that occur from human errors. With less and less accidents, we can save money on car parts, there will be less

waste in the environment, and injuries and deaths will decrease. However, automation isn't completely positive. Jobs will be lost, feuds will occur if an accident happens between a self-driving car and a regular car (liability issues), and it will be difficult to fully transition to autonomous cars. Automation in general is shifting jobs from labor to technologically-based. This will be bad for some people, but overall is good because it promotes education and innovation. In conclusion, it is necessary to state that self-driving car is the very perspective technology studied and developed by prominent companies in this sphere, such as Google, Tesla Motors, and others. The implementation of this solution will allow people to feel safer on the roads, to have more time for personal needs and interests, and to reduce the transportation costs. Hopefully, the self-driving cars will be developed enough to be ready for common usage all over the world, the legislation system regarding the autonomous cars will be improved, and common people will soon be able to evaluate the benefits of a new technology on their own.

BY ML_INT_KARISMA.