~~Safe~~ Self Driving Discount

By Michael Gonzales

CH EN 6960/593R – Dynamic Optimization for Engineers

University of Utah – Brigham Young University

Table of Contents

1. Introduction

This project is based on the premise that many car insurance companies give their customers a discount on their premiums when they drive safely. They even go as far as fitting the customer's car with a device that tracks their driving habits and records things like mileage, speed, aggressive driving, time of day, etc.[1] This project will be based on similar principles but with a twist. The scenario for this project is a self-driving car fitted with an object detection camera that has competing incentives: reach the destination quickly while driving safely.

The goal of this project will be to implement a control strategy for a self-driving car. The project will have both a software and hardware component. The hardware will be a camera that views passing images (e.g., traffic signs/lights, pedestrians, distractions, etc.). The software program will need to detect the image, classify it, and then export the object's class as an input into the control strategy. The controller will need to make decisions based on the object. See Figure 1 for an example.

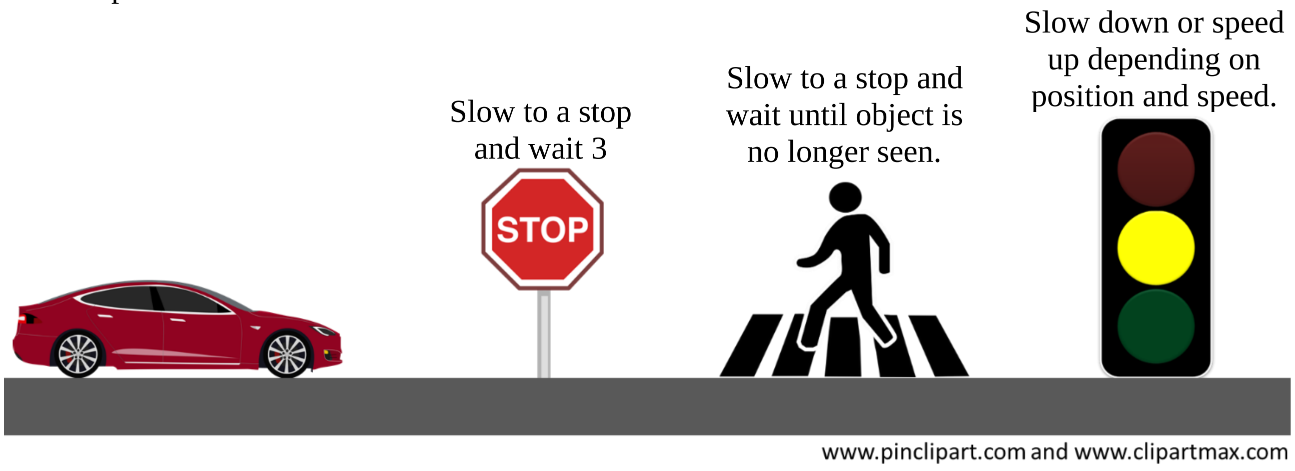

Slow to a stop and wait 3
STOP
Slow to a stop and wait until object is no longer seen.
Slow down or speed up depending on position and speed.
www.pinclipart.com and www.clipartmax.com

Figure 1. *Example scenario.*

The objective of the self-driving car will be to both reduce overall travel time and drive safe. Emphasis will be placed on limiting the excessive gas and brake pedal movement.

2. Literature review

The machine learning aspect of this project will be image classification using common python libraries such as Keras and Tensorflow. The external camera will be an ESP32-CAM[2] board that transmits images/video via wifi.

The model for the self-driving car is derived based on first principles using vehicle dynamics. There has been much work published in articles and textbooks that provide models for vehicles.

[1] https://www.progressive.com/auto/discounts/snapshot/snapshot-faq/
[2] https://www.espressif.com/en

A good starting point is a work done by Pushkin Kachroo[3], who proposes a model that relates both wheel and vehicle dynamics into a combined system. This model is adapted for this project.

To help identify optimal deceleration and acceleration, I have found some parallel work done by Zhaohui Wu[4] et al., who has published work that relates position, velocity, and acceleration with rider comfort. There work was referenced to determine if the deceleration would be comfortable for an actual occupant.

3. Model Description

Sections 3.1-3.3 show the vehicle model used to create both simulation and control. Sections 3.4 and 3.5 detail the simulation and control models.

3.1 Wheel Dynamics:

Angular motion:

$$\dot{\omega}_w = \frac{[T_e - T_b - R_w F_t - R_w F_w]}{J_w}$$

ω_w	Angular velocity of the wheel
J_w	Moment of inertia of the wheel
R_w	Radius of the wheel
T_e	Shaft torque from the engine
T_b	Brake torque
F_t	Tire tractive force
F_w	Wheel viscous friction

Tire tractive force:

$$F_t = \mu(\lambda) N_v$$

$\mu(\lambda)$	Tire adhesion coefficient
N_v	Normal reaction force from the ground

[3] Kachroo, P., & Tomizuka, M. (1994). Vehicle Traction Control And its Applications. UC Berkeley: California Partners for Advanced Transportation Technology. Retrieved from https://escholarship.org/uc/item/6293p1rh.
[4] Z. Wu, Y. Liu and G. Pan, "A Smart Car Control Model for Brake Comfort Based on Car Following," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 1, pp. 42-46, March 2009, doi: 10.1109/TITS.2008.2006777.

Tire adhesion coefficient:

$$\mu(\lambda) = \frac{2\mu_p \lambda_p \lambda}{\lambda_p^2 + \lambda^2}$$

λ	Wheel slip
λ_p	Peak wheel slip
μ_p	Peak tire adhesion coefficient

Wheel slip:

$$\lambda = \frac{(\omega_w - \omega_v)}{\omega}, \qquad \omega \neq 0$$

ω_v	Vehicle angular velocity of the wheel
ω_w	Angular velocity of the wheel
ω	Max angular velocity, i.e. $\omega = \max(\omega_w, \omega_v)$

Vehicle angular velocity of the wheel:

$$\omega_v = \frac{V}{R_w}$$

V	Vehicle linear velocity
R_w	Radius of the wheel

3.2 Vehicle Dynamics:

Acceleration:

$$\dot{V} = \frac{[N_w F_t - F_v]}{M_v}$$

N_w	Number of driving/braking wheels
F_t	Tire tractive force
F_v	Wind drag force
M_v	Vehicle mass

3.3 Combined System (Wheel + Vehicle dynamics):

$$x_1 = \frac{V}{R_w}$$
$$x_2 = \omega_w$$
$$\dot{x}_1 = -f_1(x_1) + b_{1N}\mu(\lambda)$$
$$\dot{x}_2 = -f_2(x_2) - b_{2N}\mu(\lambda) + b_3 T$$

where,

$$T = T_e - T_b$$
$$\lambda = \frac{(x_2 - x_1)}{x}$$
$$x = \max(x_1, x_2)$$
$$f_1(x_1) = \frac{[F_v(R_w x_1)]}{(M_v R_w)}$$
$$f_2(x_2) = \frac{F_w(x_2)}{J_w}$$
$$b_{1N} = \frac{N_v N_w}{(M_v R_w)}$$
$$b_{2N} = \frac{R_w N_v}{J_w}$$
$$b_3 = \frac{1}{J_w}$$

3.4 Simulation Detail:

To simulate the vehicle, the above model was implemented with the Python optimization suite, Gekko[5]. In Figure 2 it shows the vehicle response, velocity, V, for various gas pedal, T_e, step increases.

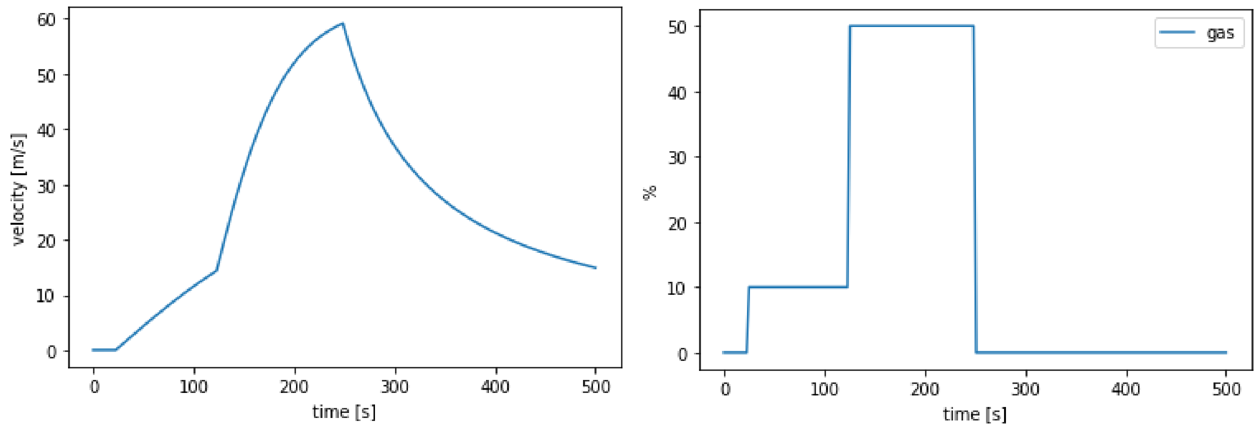


Figure 2. *Simulation response.*

3.5 Control Detail:

The method for controlling the velocity of the vehicle was the use of a model predictive control (MPC). This MPC controller was created using a separate Gekko model. The variables of interest in the MPC are:

Control Variable (CV):
- Velocity, V

Manipulated Variables (MV):
- Engine Torque, T_e (i.e. gas pedal)
- Brake Torque, T_b

Some of the preliminary results of the MPC are shown in Figure 3. As seen, the results are not optimal for some of the following reasons:
- Excessive overshoot of velocity setpoint.
- Not meeting setpoints.
- Simultaneous use of both gas and brake pedal.
- Excessive moment/increase/decrease of gas and brake pedal.

[5] https://gekko.readthedocs.io/en/latest/#

Section 5 will detail how the MPC controller was tuned to provide more optimal results which allowed for achievement of the objective, reduce travel time and drive safe.

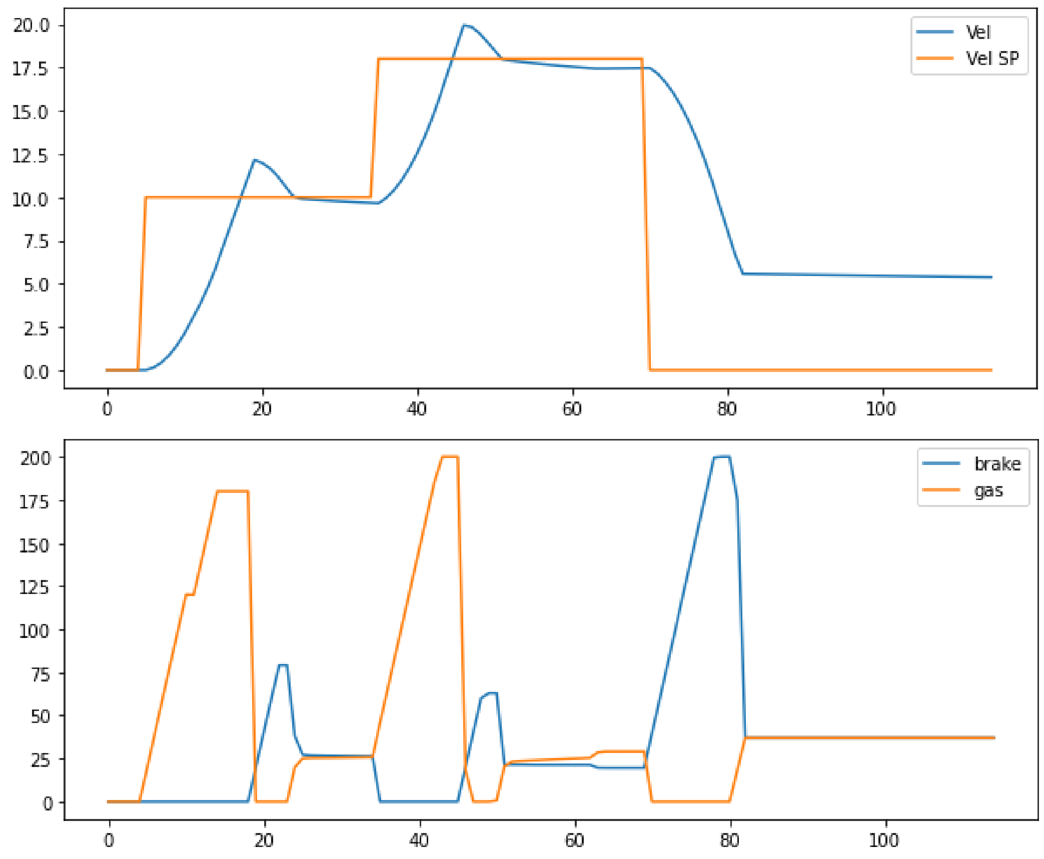
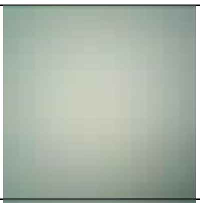
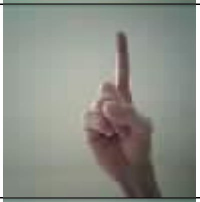
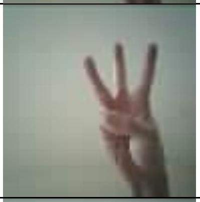
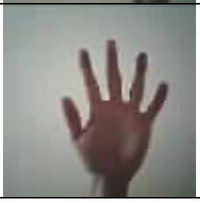


Figure 3. *Preliminary MPC results.*

4. Machine Learning

Part of the project is also implementing a machine learning aspect. This is done through an external camera and Python image classifying code. Once a known image is detected that will feed in as an input into the MPC, adjusting the MVs. The images used for the classification areg hand gestures. This approach allows for simple data (image) collection and is more dynamic than showing static photos of traffic signs to the camera. Table outlines which hand gestures are used and how they are then fed into the MPC for model control.

Table 1. Image classification.

Camera Sees:	**MPC Reacts:**
	No action taken.
	Speed change – Increase speed by 20 mph
	Pedestrian – Reduce velocity to zero until camera no longer sees image.
	Stop sign – Reduce velocity to zero. Wait for new speed increase command.

The image classification code (see Appendix) has been successful in classify the hand gestures shown in Table 1. Figure 4 shows the results of training and validating the classification model.

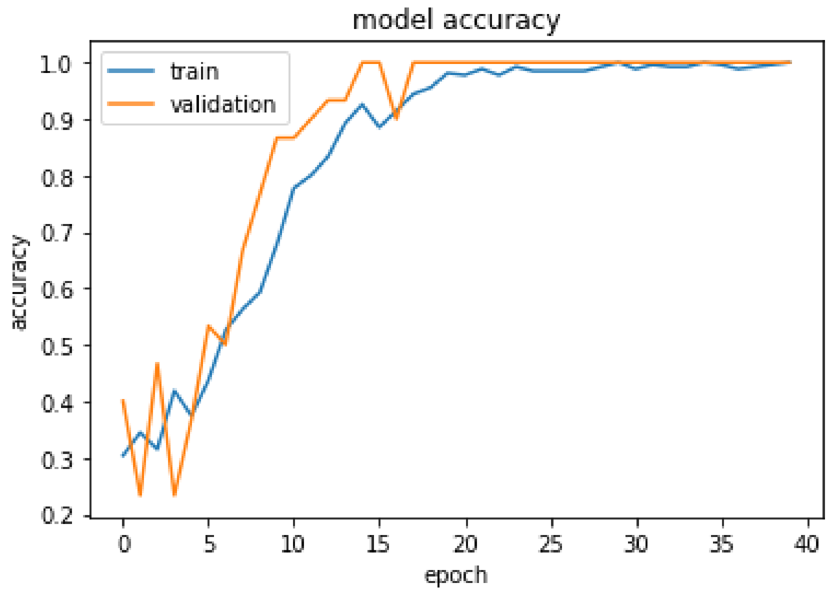


Figure 4. *Validation of classification model.*

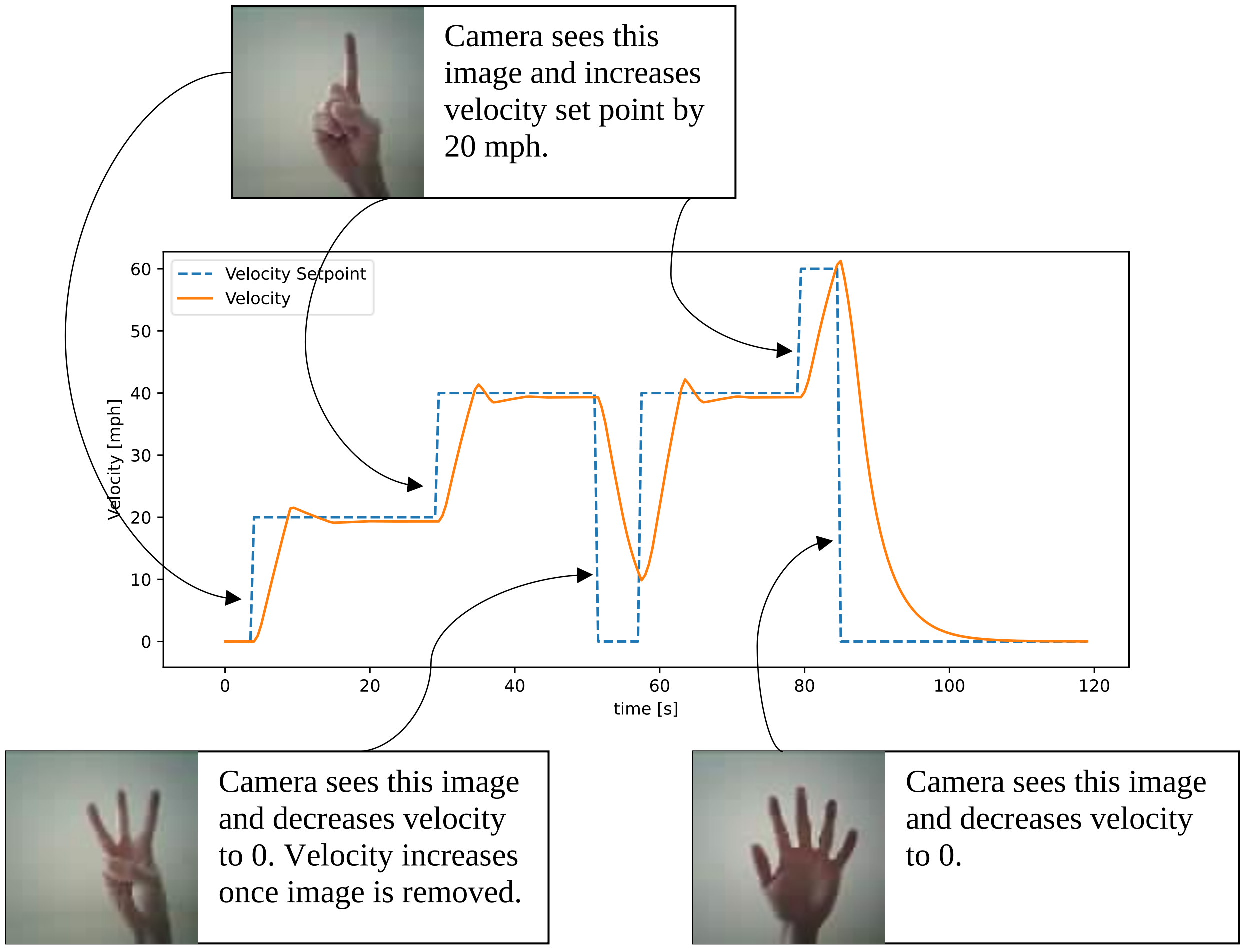
5. Results: Combined MPC and Machine Learning

As stated in Section 3.5 model tuning was needed to achieve the objectives. The focus of the tuning was on decreasing the time to reach the velocity setpoint and creating a smooth transition during braking. The optimal tuning was achieved by adjusting the Gekko model settings for the MVs, gas (T_e) and brake (T_b) pedal. Table 2 shows what settings were used and how they affected the vehicle response.

Table 2. MV settings.

Gekko Setting	**Value**	**Description**
Te – COST	0.22	This value added a penalty for using the gas pedal. The value was set lower than that of the brake. This allowed for less restrictive moment of the gas compared to brake. The COST setting was also needed so that the controller did not attempt to activate both gas and brake simultaneously.
Tb – COST	0.3	The value used was chosen to provide a smooth deceleration.
Te – DCOST	0.08	This value added a penalty for movement of the gas pedal. This helped to reduce jerky vehicle motion.
Tb – DCOST	0.1	This value added a penalty for movement of the gas pedal. This helped to reduce jerky vehicle motion.
Te – DMAXHI	16.7%	This value only allows movement/increase of gas pedal by 16.7% each time step (0.5 sec). This allows for smooth increase of using the gas pedal without rapid increase and decrease of velocity.
Tb – DMAXHI	16.7%	This value only allows movement/increase of brake pedal by 16.7% each time step (0.5 sec).

After tuning the MPC controller and adding in the image classification code into the loop the script checks the camera through each iteration. The camera image classifiers are now the inputs into the velocity setpoints of the MPC. Figure 5 shows the results of the combined process.



Here is the corresponding gas and break pedal movement to achieve the velocity profile above:

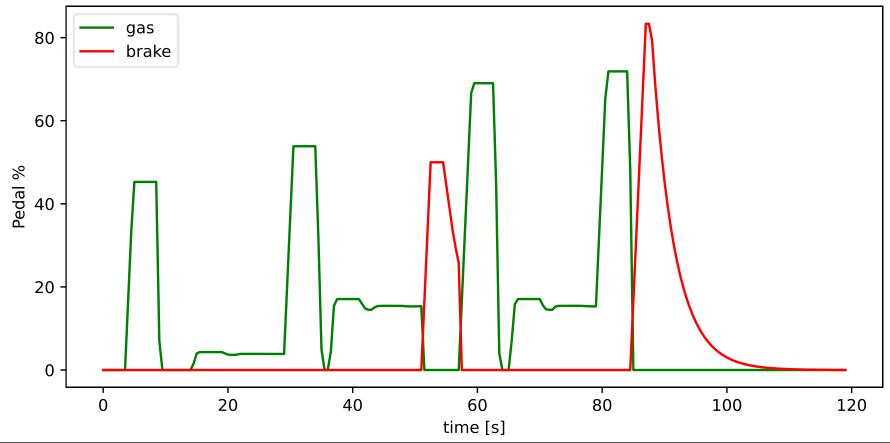


Figure 5. *Combine MPC with Image Detection.*

6. Discussion and Conclusion

The results in Section 5 show a successful implementation of MPC with Machine Learning. Figure 5 shows that there is still overshoot when reaching a velocity setpoint. This is intentional as it is a tradeoff for a shorter vehicle travel time. The deceleration of the vehicle is very smooth. This is a tradeoff that increases travel time but provides occupant comfort and a safe control of the vehicle.

Future work for this project would be to add specific objective functions to the Gekko model rather than manually set MVs options (Table 2) by trial and error. This would quantify the objectives rather than base then solely on visual evaluation of the resulting data. Also. Additional images could be added to the classification model to increase the availability of vehicle actions.

Appendix

SAVE IMAGE FROM URL

In [ ]:

```
#Code used to take and save images used to create classification model
#Code adapted from Stack Overflow user KSs
(https://stackoverflow.com/questions/50948061/how-to-save-or-download-an-
image-that-i-get-in-a-request-python)

import requests
import time

for i in range(100,200):
    image_path = "C:/Users/asd/OneDrive - University of Utah/Spring
2021/Dynamic Opt/Image_classification/images/blank/"+"limage"+str(i)+".jpg"

    Picture_request = requests.get('http://192.168.1.184/capture')
    if Picture_request.status_code == 200:
        with open(image_path, 'wb') as f:
            f.write(Picture_request.content)

    #print('picture taken',i)
    #time.sleep(0.1)
```

CLASSIFICATION CODE

In [ ]:

```
#Code modified from Toward Data Science article authored by Arthur Arnx
(https://towardsdatascience.com/all-the-steps-to-build-your-first-image-
classifier-with-code-cf244b015799)

import numpy as np
import os
import cv2
import random
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
Conv2D, MaxPooling2D
from keras.models import model_from_json
from keras.models import load_model
import matplotlib.pyplot as plt
import urllib.request

###### Prepare Image Data ######
################################
file_list = []
class_list = []

DATADIR = "images"
```

```
# All the categories you want your neural network to detect
CATEGORIES = ["blank","five","one", "three"]

# The size of the images that your neural network will use
IMG_SIZE = 50

# Checking or all images in the data folder
for category in CATEGORIES :
        path = os.path.join(DATADIR, category)
        for img in os.listdir(path):
                img_array = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)

training_data = []

def create_training_data():
        for category in CATEGORIES :
                path = os.path.join(DATADIR, category)
                class_num = CATEGORIES.index(category)
                for img in os.listdir(path):
                        try :
                                img_array = cv2.imread(os.path.join(path, img),
cv2.IMREAD_GRAYSCALE)
                                new_array = cv2.resize(img_array, (IMG_SIZE,
IMG_SIZE))
                                training_data.append([new_array, class_num])
                        except Exception as e:
                                pass

create_training_data()

random.shuffle(training_data)

X = [] #features
y = [] #labels

for features, label in training_data:
        X.append(features)
        y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

# Creating the files containing all the information about your model
pickle_out = open("X.pickle", "wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle", "wb")
pickle.dump(y, pickle_out)
pickle_out.close()

pickle_in = open("X.pickle", "rb")
X = pickle.load(pickle_in)
```

In [ ]:

```
import numpy as np
import os
import cv2
import random
import pickle
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
Conv2D, MaxPooling2D
from keras.models import model_from_json
from keras.models import load_model
import matplotlib.pyplot as plt
import urllib.request

###### Building convolutional neural network ######
###################################################
# Opening the files about data
X = pickle.load(open("X.pickle", "rb"))
y = pickle.load(open("y.pickle", "rb"))

# normalizing data (a pixel goes from 0 to 255)
X = X/255.0

# Building the model
model = Sequential()

act_fun = "relu"
# 4 convolutional layers
model.add(Conv2D(32, (3, 3), input_shape = X.shape[1:]))
model.add(Activation(act_fun))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation(act_fun))
model.add(MaxPooling2D(pool_size=(2,2)))

#model.add(Conv2D(64, (3, 3)))
#model.add(Activation(act_fun))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation(act_fun))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 2 hidden layers
model.add(Flatten())
model.add(Dense(128))
model.add(Activation(act_fun))

model.add(Dense(128))
model.add(Activation(act_fun))

# The output layer with 5 neurons, for 5 classes
model.add(Dense(4))
model.add(Activation("softmax"))

#callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# Compiling the model using some basic parameters
model.compile(loss="sparse_categorical_crossentropy",
                              optimizer="adam",
                              metrics=["accuracy"])

# Training the model, with 40 iterations
# validation_split corresponds to the percentage of images used for the
validation phase compared to all the images
history = model.fit(X, np.array(y), batch_size=32, epochs=25,
validation_split=0.1)

# Saving the model
model_json = model.to_json()
with open("model.json", "w") as json_file :
        json_file.write(model_json)

model.save_weights("model.h5")
print("Saved model to disk")

model.save('CNN.model')

# Printing a graph showing the accuracy changes during the training phase
print(history.history.keys())
plt.figure(1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

In [2]:

```
###### Predict an image ######
##############################
import tensorflow as tf
import urllib.request
import cv2
import time
import matplotlib.pyplot as plt
%matplotlib tk

def prepare(file):
    IMG_SIZE = 50
    img_array = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

CATEGORIES = ["blank","five","one", "three"]

model = tf.keras.models.load_model("CNN.model")

 # to run GUI event loop
plt.ion()
 # here we are creating sub plots
figure1, ax1 = plt.subplots(figsize=(10, 8))

ax1.plot()


for i in range(60):

    urllib.request.urlretrieve("http://192.168.1.184/capture", "latest.jpg")
    image = prepare("latest.jpg")
    prediction = model([image])
    prediction = list(prediction[0])
    status = CATEGORIES[prediction.index(max(prediction))]
    ax1.clear()

    ax1.text(0.3,0.3,status, fontsize=50)
    ax1.text(0,0,i, fontsize=20)


        # drawing updated values
    figure1.canvas.draw()

    figure1.canvas.flush_events()
    time.sleep(1)
```

MPC Control

In [ ]:

```
from gekko import GEKKO
import numpy as np
import matplotlib.pyplot as plt
%matplotlib tk
import time
import tensorflow as tf
import urllib.request
import cv2
import copy


divisions = 1/2 #divisions per second

#### SIMULATION MODEL ####
##########################
s = GEKKO(remote=False)
```

```
s.time = [0,divisions]

#constants
s.Mw = s.Const(value = 14) #wheel_mass = 14 #kg
s.Rw = s.Const(value = 0.334) #Radius of the wheel #tesla 3
s.Mv = s.Const(1700) #mass of vehicle in kg
s.lam_p = s.Const(value = 0.8) #Peak wheel slip
s.mu_p = s.Const(value = 0.8) #asphalt=0.9, Peak tire adhesion coefficient
s.Nw = s.Const(value = 4)  #number of driving/breaking wheels

#variables
s.Te = s.MV() #engine torque
s.Tb = s.MV() #Brake torque

s.ome_w = s.CV(value = 0) #Angular velocity of the wheel
s.vel = s.CV(0, lb=0)
s.x1 = s.Var(0) #state var 1 [1/s]
s.x2 = s.Var(0) #state var 1 [1/s]
s.lam = s.Var(0)
s.mu = s.Var(0, lb=-1, ub =1)

#intermediates
s.Jw = s.Intermediate(1/2*s.Mw*s.Rw**2) #Moment of inertia of the wheel
s.Nv = s.Intermediate(s.Mv*9.81) #Normal reaction force from the ground
s.Fw = s.Intermediate(0.08*s.ome_w) #Wheel viscous friction
s.Fv = s.Intermediate(4*s.vel**2) #drag force
s.x = s.Intermediate(max(s.x1.value,s.x2.value))
s.lam_y = s.Intermediate((s.x2-s.x1))
s.mu_y = s.Intermediate((2*s.mu_p*s.lam_p*s.lam))

s.T = s.Intermediate(s.Te-s.Tb)
s.f1 = s.Intermediate(s.Fv/(s.Mv*s.Rw))
s.f2 = s.Intermediate(s.Fw/s.Jw)
s.b1 = s.Intermediate(s.Nv*s.Nw/(s.Mv*s.Rw))
s.b2 = s.Intermediate(s.Rw*s.Nv/s.Jw)
s.b3 = s.Intermediate(1/s.Jw)


#equations
s.Equation(s.x1==s.vel/s.Rw)
s.Equation(s.x2==s.ome_w)
s.Equation(s.x1.dt()==-s.f1+s.b1*s.mu)
s.Equation(s.x2.dt()==-s.f2-s.b2*s.mu+s.b3*s.T)
s.Equation(s.lam*s.x==s.lam_y)
s.Equation(s.mu*(s.lam_p**2 + s.lam**2)==s.mu_y)

#solver options
s.options.imode = 4


#### MPC MODEL ####
###################
m = GEKKO(remote=False)
```

```
#m.time = np.arange(0,15,divisions)
#m.time = [0,divisions,1,2,4,6,8,12,15,20,25,30]
m.time = [0,divisions,1,2,4,6,8]

#max torques
Te_max = 700
Tb_max = 700

#constants
m.Mw = m.Const(value = 14) #wheel_mass = 14 #kg
m.Rw = m.Const(value = 0.334) #Radius of the wheel #tesla 3
m.Mv = m.Const(1700) #mass of vehicle in kg
m.lam_p = m.Const(value = 0.8) #Peak wheel slip
m.mu_p = m.Const(value = 0.8) #asphalt=0.9, Peak tire adhesion coefficient
m.Nw = m.Const(value = 4)  #number of driving/breaking wheels

#parameters
m.Te = m.MV() #engine torque
m.Te.STATUS = 1
m.Te.lower = 0
m.Te.upper = Te_max
m.Te.DCOST = 0.08
m.Te.COST = 0.22
m.Te.DMAXHI = Te_max/6
m.Te.FSTATUS = 0

m.Tb = m.MV() #Brake torque
m.Tb.STATUS = 1
m.Tb.lower = 0
m.Tb.DMAXHI = Tb_max/6
m.Tb.upper = Tb_max
m.Tb.DCOST = 0.1
m.Tb.COST = 0.3
m.Tb.FSTATUS = 0

#variables
m.ome_w = m.CV(value = 0) #Angular velocity of the wheel

m.vel = m.CV(0)
m.vel.lower = 0
m.vel.status= 1
m.vel.fstatus = 1

m.x1 = m.Var(0) #state var 1 [1/s]
m.x2 = m.Var(0) #state var 1 [1/s]
m.lam = m.Var(0)
m.mu = m.Var(0, lb=-1, ub =1)


#intermediates
m.Jw = m.Intermediate(1/2*m.Mw*m.Rw**2) #Moment of inertia of the wheel
m.Nv = m.Intermediate(m.Mv*9.81) #Normal reaction force from the ground
m.Fw = m.Intermediate(0.08*m.ome_w) #Wheel viscous friction
```

```
m.Fv = m.Intermediate(4*m.vel**2) #drag force
m.x = m.Intermediate(max(m.x1.value,m.x2.value))
m.lam_y = m.Intermediate((m.x2-m.x1))
m.mu_y = m.Intermediate((2*m.mu_p*m.lam_p*m.lam))

m.T = m.Intermediate(m.Te-m.Tb)
m.f1 = m.Intermediate(m.Fv/(m.Mv*m.Rw))
m.f2 = m.Intermediate(m.Fw/m.Jw)
m.b1 = m.Intermediate(m.Nv*m.Nw/(m.Mv*m.Rw))
m.b2 = m.Intermediate(m.Rw*m.Nv/m.Jw)
m.b3 = m.Intermediate(1/m.Jw)


#equations
m.Equation(m.x1==m.vel/m.Rw)
m.Equation(m.x2==m.ome_w)
m.Equation(m.x1.dt()==-m.f1+m.b1*m.mu)
m.Equation(m.x2.dt()==-m.f2-m.b2*m.mu+m.b3*m.T)
m.Equation(m.lam*m.x==m.lam_y)
m.Equation(m.mu*(m.lam_p**2 + m.lam**2)==m.mu_y)
#m.Equation(m.y==m.mu)

#objective
#m.Minimize(abs(m.acc))

#solver options
m.options.imode = 6
m.options.cv_type = 1
m.options.max_iter = 500
m.options.coldstart = 1


#image classification
def prepare(file):
    IMG_SIZE = 50
    img_array = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

CATEGORIES = ["blank","five","one","three"]
#
model = tf.keras.models.load_model("CNN.model")

#loop
sec = 200
steps = int(sec/divisions)

#setpoints
 #using camera
V_sp = np.zeros(steps)
V_sp_incr = np.zeros(steps)

 #not using camera
```

```
#V_sp = np.zeros(steps)
#V_sp[2:] = 8.9408
#V_sp[int(steps/4):] = 3*8.9408
#V_sp[int(2*steps/4):] = 5*8.9408
#V_sp[int(3*steps/4):] = 0


#wait times
speed_wait = 0
pre_status = "blank"

#Arrays for values
V = np.zeros(steps)
gas = np.array([None]*steps)
brake = np.array([None]*steps)
Time = np.arange(0,sec,divisions)

##plots##

 # to run GUI event loop
plt.ion()
 # here we are creating sub plots
figure1, ax1 = plt.subplots(figsize=(10, 8))
figure2, ax2 = plt.subplots(figsize=(10, 8))
figure3, ax3 = plt.subplots(figsize=(10, 8))

ax1.plot()
ax2.plot()
ax3.plot()

for i in range(steps):

    #if using camera uncomment next section
    #check camera
    urllib.request.urlretrieve("http://192.168.1.184/capture", "latest.jpg")
    image = prepare("latest.jpg")
    prediction = model([image])
    prediction = list(prediction[0])
    status = CATEGORIES[prediction.index(max(prediction))]
    pre_status = copy.copy(status)


    #increase speed
    if status=="one" and speed_wait<i:
        V_sp[i:]= V_sp[i-1]+8.9408
        V_sp_incr[i:] = V_sp_incr[i-1]+8.9408
        speed_wait = i+5/divisions

    #stop sign
    if status =="five":
        V_sp[i:] = 0
        speed_wait = 1000
        V_sp_incr[i:]=0
```

```
if status=="five" and V[i]<0.01:
    speed_wait = i+3/divisions

#if status=="five" and V[i]<0.01 and speed_wait<i:
    #V_sp[i:] = V_sp_incr[i:]

#pedestrian
if status == "three":
    V_sp[i:] = 0

if status=="blank" and pre_status=="three":
    V_sp[i:] = V_sp_incr[i:]

# input setpoint with deadband +/- DT
if V_sp[i] == 0:
    band = 0.004
else:
    band = 0.3

m.vel.SPHI = V_sp[i] + band
m.vel.SPLO = V_sp[i] - band

# solve MPC
m.solve(disp=False, debug=0)

#enter values from mpc to simulation
s.Tb.value = m.Tb.NEWVAL
s.Te.value = m.Te.NEWVAL

gas[i] = m.Te.NEWVAL
brake[i] = m.Tb.NEWVAL

#simulate
s.solve(disp=False, debug=0)

#take values from simualation into mpc
m.vel.MEAS = s.vel.model

#record values from sim
V[i] = s.vel.model


if i%1==0:

    if i<30/divisions:
        start=0
    else:
        start=int(i-30/divisions)
    #print("status = ",status)

    ax1.clear()
    ax2.clear()
```

```
        ax3.clear()

        ax3.text(0.3,0.3,status, fontsize=50)
        ax3.text(0,0,speed_wait, fontsize=20)
        ax3.text(0.2,0,i, fontsize=20)

        ax1.set_xlabel("time [s]")
        ax1.set_ylabel("Velocity [mph]")
        line2, = ax1.plot(Time[start:i], V_sp[start:i]*2.237,'--',
label="Velocity Setpoint")
        line1, = ax1.plot(Time[start:i], V[start:i]*2.237, label="Velocity")
        ax1.set_ylim((0, None))
        ax1.legend(loc='upper left')

        ax2.set_xlabel("time [s]")
        ax2.set_ylabel("Pedal %")
        line1, =ax2.plot(Time[start:i], gas[start:i]/Te_max*100, 'g-
',label='gas')
        line2, =ax2.plot(Time[start:i], brake[start:i]/Tb_max*100,'r-',
label='brake')
        ax2.set_ylim((0, None))
        ax2.legend(loc='upper left')

        # drawing updated values
        figure1.canvas.draw()
        figure2.canvas.draw()
        figure3.canvas.draw()

        # This will run the GUI event
        # loop until all UI events
        # currently waiting have been processed
        figure1.canvas.flush_events()
        figure2.canvas.flush_events()
        figure3.canvas.flush_events()
```