Appendices

Appendix A: Modifications to Simulation Code

```
## “simulation.py”
## GLOBAL VARIABLES
elapsed_time = 0
iteration = 0
i = 0
time = 0;
c_fs_data = [0]
c_rs_data = [0]
time_vector = [0]
fs_accel_vector = [0]
rs_accel_vector = [0]
start_time = datetime.now()
accel_vector = [0,0,0,0]
c_rs_vector = [0]
c_fs_vector = [0]
fs_pos_vector = [0]
rs_pos_vector = [0]
## PASSING VARIABLES TO “update_state”
i += 1
car.update_state(time_step, i, accel_vector, c_fs_vector, c_rs_vector, start_time, time, time_vector, fs_accel_vector, rs_accel_vector, c_fs_data, c_rs_data, fs_pos_vector, rs_pos_vector)
## “car.py”
def update_state(self, time_step, i, accel_vector, c_fs_vector, c_rs_vector, start_time, time, time_vector, fs_accel_vector, rs_accel_vector, c_fs_data, c_rs_data, fs_pos_vector, rs_pos_vector):
"""
TODO
"""
print(i)
if i == 88000:
print(datetime.now() - start_time)
# Active Damping. The front and rear damping constants and damping matrix are updated every time step.if i >= 2:
# Mass and Inertia Properties.
m_c = 1600
m_f = 23
m_r = m_f
I_zz = 2500
m = m_c + m_f + m_r
mass_vector = np.array([m_c, I_zz, m_f, m_r])
# Constant Damping Properties
c_ft = 20
c_rt = 20
# length properties
wheelbase = 2.74
l_f = 0.4 * wheelbase
l_r = 0.6 * wheelbase
# Active Damping Feedback and Desired Output
```

```
feedback_accel = accel_vector[i+2]
feedback_accel_front = feedback_accel[2]
feedback_accel_rear = feedback_accel[3]
# Control the sensitivity of the active damping
desired_accel = 0
lower = desired_accel - 0.08
upper = desired_accel + 0.08
#desired_accel_lower = desired_accel - 0.01
#desired_accel_upper = desired_accel + 0.01
## ACTIVE DAMPING LOGIC
c_fs = c_fs_vector[i-1]
c_rs = c_rs_vector[i-1]
## FRONT SUSPENSION
if feedback_accel_front > upper:
if c_fs == 2000:
c_fs = c_fs
else:
c_fs += 100
elif feedback_accel_front < lower:
if c_fs == 2000:
c_fs = c_fs
else:
c_fs += 100
if feedback_accel_front >= lower and feedback_accel_front <= upper:if c_fs == 1000:
c_fs = c_fs
else:
c_fs -= 100
## REAR SUSPENSION
if feedback_accel_rear > upper:
if c_rs == 2000:
c_rs == c_rs
else:
c_rs += 100
elif feedback_accel_rear < lower:
if c_rs == 2000:
c_rs = c_rs
else:
c_rs += 100
if feedback_accel_rear >= lower and feedback_accel_rear <= upper:
if c_rs == 1000:
c_rs = c_rs
else:
c_rs -= 100
## CAPTURING DATA
fs_pos_vector.append(self.state["position"][2,0])
rs_pos_vector.append(self.state["position"][3,0])
time_vector.append(time)
fs_accel_vector.append(feedback_accel_front)
rs_accel_vector.append(feedback_accel_rear)
```

```
c_fs_data.append(c_fs)
c_rs_data.append(c_rs)
if i == 88000:
with open('active_fs_accel_data.csv', 'w') as csvfile:
fieldnames = ['time', 'accel']
writer_fs = csv.DictWriter(csvfile, fieldnames=fieldnames)
for j in range(1,len(time_vector)):
writer_fs.writerow({'time': time_vector[j], 'accel': float(fs_accel_vector[j])})
with open('active_fs_pos_data.csv', 'w') as csvfile:
fieldnames = ['time', 'ypos']
writer_fs = csv.DictWriter(csvfile, fieldnames=fieldnames)
for j in range(1,len(time_vector)):
writer_fs.writerow({'time': time_vector[j], 'ypos': float(fs_pos_vector[j])})
## UPDATING CAR.STATE WITH NEW DAMPING CONSTANTSself.state["c_fs"] = c_fs
self.state["c_rs"] = c_rs
c_fs_vector.append(c_fs)
c_rs_vector.append(c_rs)
#print(c_fs_vector)
#print(accel_vector[i-1])
damping_matrix = np.array([
[-(c_fs + c_rs), l_r * c_rs - l_f * c_fs, c_fs, c_rs],
[-(l_f * c_fs - l_r * c_rs), -(l_f**2 * c_fs + l_r**2 * c_rs),
l_f * c_fs, -l_r * c_rs],
[c_fs, l_f * c_fs, -(c_fs + c_ft), 0],
[c_rs, -l_r * c_rs, 0, -(c_rs + c_rt)] ])
damping_matrix = damping_matrix / mass_vector[:, None]
#acceleration for remaining time-steps
accel = (
(stiffness_matrix @ position)
+ (damping_matrix @ velocity)
+ (road_stiffness_matrix @ road_position)
+ (road_damping_matrix @ road_velocity)
+ (self.normal_force_vector)
)
#print(accel_vector[i+2])
c_fs_vector.append(c_fs)
c_rs_vector.append(c_rs)
accel_vector.append(accel)
#print(accel_vector[i])
##”plot_sim.py”
## POSITIONING C VALUES IN VIDEO OUTPUT
annotations["C_FrontSus"] = ax.annotate(
"", xy=(0.5, 0.06), xycoords="axes fraction"
)
annotations["C_RearSus"] = ax.annotate(
"", xy=(0.5, 0.02), xycoords="axes fraction"
)
## UPDATING ANNOTATIONS
self.annotations["C_FrontSus"].set_text(
"C_FrontSus = {:.1f} Ns/m".format(car.state["c_fs"])
)
self.annotations["C_RearSus"].set_text(
"C_RearSus = {:.2f} Ns/m".format(car.state["c_rs"])
)##”road.py”
## CHANGING ROAD PROPERTIES FOR BRAKING SIMULATION
class Road:
def __init__(
self, length, resolution=300, mode="bump", amplitude=0,
frequency=0.05, x_min=None
):
```