Using Adaptive and Cooperative Adaptive Cruise Control to Maximize Throughput of Signalized Arterials

Daniel Albarnaz Farias

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-109
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-109.html

May 19, 2017

Acknowledgement

I wish to thank Alex Kurzhanskiy for the extensive guidance in the project. I wish to also thank for Armin Askari for his involvement in the greater project and for debugging a lot of the code provided in the Appendix.

Using Adaptive and Cooperative Adaptive Cruise Control to Maximize Throughput of Signalized Arterials

Daniel Farias

Electrical Engineering and Computer Sciences
University of California, Berkeley

May 19, 2017

Table of Contents

Abstract

This report discusses how the maximum acceleration and proportion of vehicles using ACC and CACC technology affect the throughput of a given intersection. In most cases, two scenarios are simulated and discussed: (1) free flow after an intersection, and (2) a second intersection 300 meters after the first intersection. Lastly, a microscopic-level simulation of a four-mile length arterial network in Arcadia is used to evaluate the performance of ACC and CACC vehicles. These simulations use the mean travel time and standard deviation as measures of performance. Platoon performance is able to achieve near optimal results when compared to best-case theoretical models. The report concludes the possibility for a very high improvement in urban road capacity by utilizing ACC and CACC technologies at little cost to infrastructure.

1 Introduction

The flow of a freeway is simply the product of speed and density. The headway is the inverse of the density, so the capacity of a given freeway for vehicles traveling at the speed limit increases proportionally with a shorter headway. Normal highway driving conditions constitute a minimum of a two-second headway, translating to about 55 meters between vehicles at 60 mph [1, 2]. Two levels of longitudinal control technologies permit headway reductions by factors of two to three relative to manual driving; adaptive cruise control (ACC) and cooperative adaptive cruise control (CACC). A platoon is a group of such vehicles travelling with a very short headway. Several demonstrations have been made of such technology, with one of the earliest being on the I-15 freeway in San Diego, 1997, with an 8-car platoon traveling roughly 5 meters apart at 60 mph [3, 4]. These demonstrations show a headway reduction even beyond the expected factor of two to three relative to manual vehicles.

These results hold for optimal road conditions: steady flow at the speed limit. However, many roads have bottlenecks at signaled intersections, in which case the possible improvement from platooning is not as clear. For example, consider a four-approach intersection with two lanes in each direction; one for through traffic and one for left turns. Suppose a capacity of 1.8 seconds between vehicles, translating to a total capacity of 2,000 vehicles per hour (vph) per lane. The total capacity leading into the intersection is then 16,000 vph, but since the intersection can only allow two movements at a time, the effective capacity is only 4,000 vph. Thus, increasing the effective capacity by using platooning will not increase the capacity of the full network.

A study by Lioris et al. [5] delves into this observation, investigating the possibility of vehicles crossing an intersection in a platoon using ACC or CACC technology. It concludes that if one increases the saturation flow rates at all intersections in an urban network by a factor Γ, "the network can support an increase in demand by the same factor Γ, with no increase in queuing delay or travel time, and using the same signal control. However, the queues will also grow by the same factor Γ, so if this leads to a saturation of the links, the improvement in throughput will be sub-linear in Γ. On the other hand, if the cycle time is reduced, the queues will also be reduced, and this may restore the linear growth in demand."

However, the study only addresses the case where 100 percent of vehicles use ACC or CACC technology, i.e. a penetration rate of 100 percent. The scenarios in this report investigate an arbitrary proportion of vehicles that use manual, ACC, or CACC technology. Additionally, as stated in the study [5], a "second limitation is that in short urban links vehicles will slow down quickly as queues build up. As a result the saturation flow rate at the upstream intersection will be reduced, thereby depriving the system of the full productivity benefit. It is important to investigate this reduction," which is also addressed in this report.

These results utilize SUMO, an open source microscopic simulator of vehicle traffic; each vehicle is simulated individually. The vehicles are set to use the Intelligent Driver Model (IIDM) [7], which improves upon the default SUMO Intelligent Driver Model [8]. The model was implemented for use in SUMO and the code is available in the Appendix, with further details on the model in Section 2. Section 3 discusses the default intersection throughput when using manually driven vehicles (manual vehicles). Section 4 discusses how the throughput changes when introducing ACC vehicles, and then CACC vehicles. Section 5 discusses the CACC model implemented. It additionally evaluates the ACC and CACC models using travel time and network throughput. For this task, a four-mile section of the Colorado Boulevard and Huntington Drive arterial network in Arcadia, California is used. The network has thirteen signaled intersections. Section 6 concludes the presents the conclusions.

2 Car Following Model

Table 1 includes the description of all values used in the equations in this section, along with the default values used when appropriate:

Symbol	Description	Default Value
t	Time	
Δt	Model time step	0.05 seconds
l	Vehicle length	5 meters
g_{min}	Minimal allowed gap	4 meters
$g(t)$	Actual distance, or gap, from front of given vehicle to tail of leading vehicle	
$g_d(t)$	Desired distance from front of given vehicle to tail of leading vehicle	
τ	"reaction time", or time gap between vehicles	2.05 seconds
$\theta(t)$	Headway of given vehicle	
$f(t)$	Flow, or inverse of headway	
$x(t)$	Vehicle position	
$x_l(t)$	Position of lead vehicle	
v_{max}	Speed limit	20 m/s = 44.7 mph
$v(t)$	Speed of given vehicle	
$v_l(t)$	Speed of leading vehicle	
a_{max}	Maximal acceleration of given vehicle	1.5 m/s^2

a(t)	Acceleration of given vehicle	
b	Desired acceleration for given vehicle	2 m/s^2

Table 1: Notation summary

The following are the state equations for the IIDM car-following model:

$$v(t+\Delta t) = v(t) + a(t)\,\Delta t \quad (1)$$

$$x(t+\Delta t) = x(t) + v(t)\,\Delta t + \frac{a(t)\Delta t^2}{2} \quad (2)$$

$$a(t) = \begin{cases} a_{max}\left(1 - \left(\frac{g_d(t)}{g(t)}\right)^{\delta_1}\right), & if\ g_d(t) > g(t) \\ a^*(t)\left(1 - \left(\frac{g_d(t)}{g(t)}\right)^{\delta_1 a_{max}/a^*(t)}\right), & else \end{cases} \quad (3)$$

Where

$$a^*(t) = a_{max}\left(1 - \left(\frac{v(t)}{v_{max}}\right)^{\delta_2}\right) \quad (4)$$

$$g_d(t) = g_{min} + max\left\{0, v(t)\,\tau + \frac{v(t)(v(t)-v_l(t))}{2\sqrt{a_{max}b}}\right\} \quad (5)$$

Here, the critical variable is $a(t)$, the acceleration. For these simulations, we used $\delta_1 = 4$ and $\delta_2 = 8$. The IIDM model can be tuned to accelerate more aggressively by increasing δ_1 and δ_2. The equilibrium headway is achieved when $a(t) = 0$, $v(t) = v_{max} = v_l(t)$ and $g(t) = g_{min} = v(t)\,\tau$. It can then be calculated to be:

$$\theta_{equilibrium} = \tau + \frac{g_{min}+l}{v_{max}} \quad (6)$$

Using the default values from Table 1, $\theta_e = 2.5$ seconds for manual vehicles, which corresponds to the time period between vehicles from tail to front. This is equivalent to a flow of 0.4 vehicles per second, or 1440 vph. This aligns generally with empirical estimates of throughput, which vary between 1200 to 1900 vph.

3 Intersection Flow

Consider the example in Fig. 1; there is an infinite number of vehicles queued in an arterial with the minimum gap from Table 1 between them. The light turns green at time t = 0, at which time the vehicles begin accelerating. Two sets of experiments are shown; first with a free roadway ahead of the intersection, then with a second signaled intersection 300 meters down the road with a fixed red light. The segment can only accommodate 33 vehicles between the intersections, which exceeds the number of vehicles that can cross the signal in one minute with a default separation of over 2 seconds.

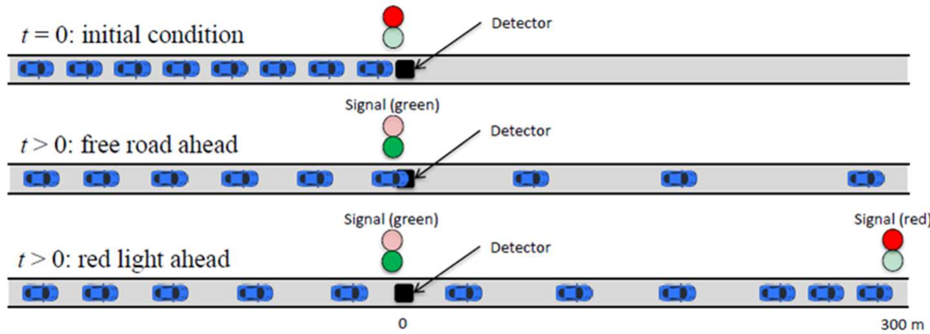

Figure 1: All vehicles are initially still with the minimum gap between them. The signal turns green at time t = 0, and the vehicles start to accelerate. In the second experiment, there is an additional intersection, after 300 meters, at which the vehicles must stop.

The trajectories, speeds, and accelerations of the first ten vehicles are shown in Fig. 2. The x-axis shows the time after the signal turns green. The y-axis shows the given vehicle's position, velocity, or acceleration along the road segment. The top two plots have a black horizontal line at x=0 corresponding to the position of the signal/intersection. The first vehicle is infinitely far from any leader, and so in either scenario it begins to accelerate at the maximal parameter. In the first scenario, the acceleration curve of the first vehicle follows equation (4), corresponding to free acceleration, asymptotically reaching 0 acceleration and the maximum velocity. Other vehicles must wait momentarily until

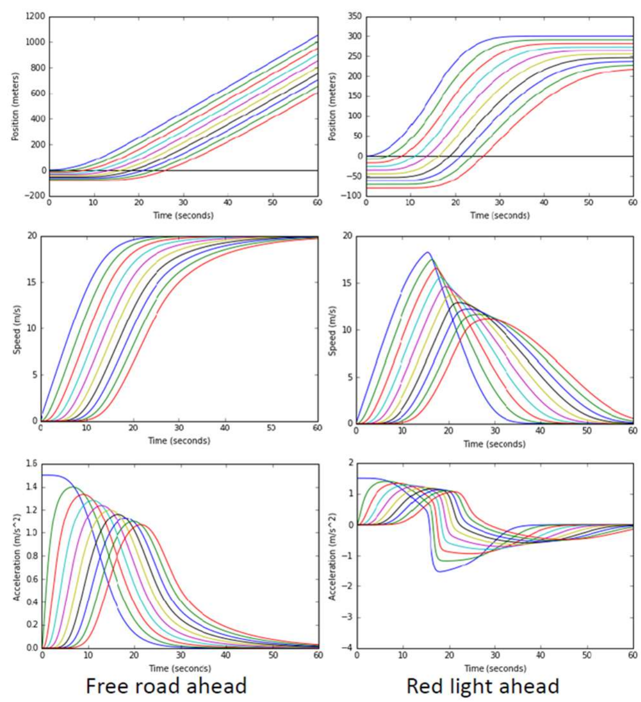

Figure 2: Vehicle trajectories, speeds and accelerations: first additional intersection with no intersection on left, and second experiment with red light on right.

the increase in gap propagates to their position in the queue. In the second scenario, the second intersection is located at x=300. Vehicles slow down as they approach the intersection, and as the vehicles stop and the queue grows, the flow through the first intersection begins to slow down until it is completely blocked. This reflects the second limitation cited in [5], where vehicles in a short link will slow quickly as a queue grows, leading to reduction in the saturation flow rate at upstream intersections.

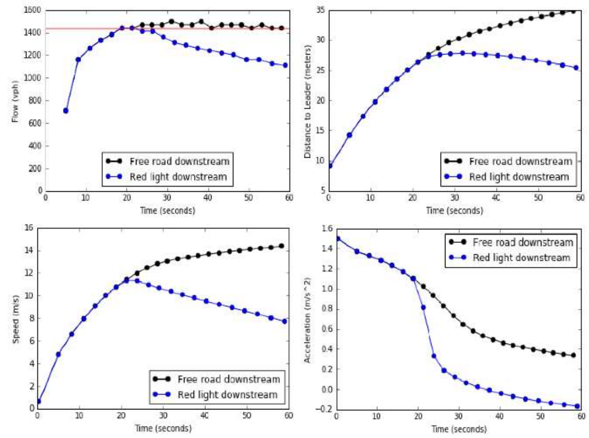

Figure 3: In order, measurements of flow, distance to leader, speed, and acceleration at the detector location shown in Fig. 1.

We can then consider measurements made for vehicles by a detector as they pass the intersection (shown in Fig. 1), shown in Fig. 3. Each dot in Fig. 3 represents a vehicle passing through the detector. The instantaneous flow for each vehicle is calculated by using the reciprocal of the time elapsed since the previous vehicle, which corresponds to the headway. The equilibrium flow for manual vehicles of 1440 vph, as discussed in section 1, is shown as a red line in the top left graph. For the first scenario, with no obstruction of flow, the gaps and speeds both monotonically increase, whereas the acceleration monotonically decreases. Additionally, the number of vehicles that cross the first intersection differs greatly between the two scenarios. At equilibrium flow, 24 vehicles would cross in the first minute. In the first scenario, 23 vehicles cross, and in the second, only 21 vehicles cross. Thus, there is a roughly ten percent loss in flow in the first minute due to the backflow when introducing the second intersection.

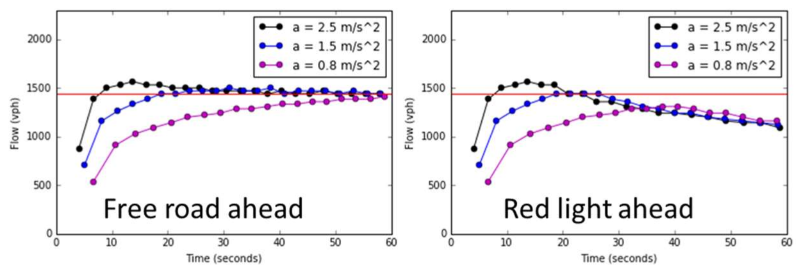

Figure 4: The total throughput result for three different values of acceleration for scenario one (on left) and scenario two (on right).

a_{max} (m/s^2)	Scenario	IIDM
0.8	Free flow	20
	Second intersection	19
1.5	Free flow	23
	Second intersection	21
2.5	Free flow	24
	Second intersection	22

Table 2: Simulation summary: intersection flow in first minute after t=0.

The experiment is then run again with three different values for maximum acceleration: 0.8, 1.5, and 2.5 m/s^2. The effect on the throughput is seen in Fig. 4 and summarized in Table 2. This simulation establishes the flow for the manual case, which is compared to ACC and CACC results in section 4.

4 Impact of ACC and CACC

The same experiments are now repeated but with various different levels of ACC and CACC penetration, corresponding to the fraction of all vehicles that have ACC or CACC capability. Manually driven, ACC, and CACC vehicles all have different values for "reaction time", which corresponds to the minimal time gap between vehicles, and spatial gap. The values used in simulation are given in Table 3. The assumption is that ACC and CACC vehicles require a smaller headway in both seconds and meters. The same car following model, IIDM, is used by all vehicle classes. The only difference between a manual vehicle and an ACC vehicle are the two parameters specified in Table 3. CACC vehicles, however, form "platoons", or groups, of connected vehicles once multiple CACC vehicles become adjacent within a lane. Within this platoon, all followers show the further reduced parameters given in Table 3. CACC vehicles that follow manual vehicles, however, act in the same way as ACC vehicles. Such vehicles include two cases: (1) lone CACC vehicles surrounded by manual vehicles, (2) the leader of any given CACC platoon. We call this the CACC car-following model.

Vehicle class	τ (seconds)	Eq Flow (vph)	g_{min} (m)
Manual	2.05	1,440	4
ACC	1.1	2,400	3
CACC	0.8	3,000	3

Table 3: Values for the reaction time and minimal gap for all three vehicle classes used in simulations.

Take the acceleration function for $a(t)$ defined by equation (3). The CACC car following model is given by [7]:

$$a_{CACC}(t) = \begin{cases} a(t), & if\ a_{CAH}(t) \leq a(t) \\ a_{CAH}(t) + b\tanh\left(\frac{a(t)-a_{CAH}(t)}{b}\right), & else \end{cases} \tag{7}$$

Where

$$a_{CAH}(t) = \begin{cases} \frac{v^2(t)\overline{a_l}(t)}{v_l^2(t)-2(x_l(t)-x(t)-l)\,\overline{a_l}(t)} \\ \overline{a_l}(t) - \frac{\left(v(t)-v_l(t)\right)^2 \Theta(\mathrm{v(t)-v_l(t)})}{2(x_l(t)-x(t)-l)}, & else \end{cases} \tag{8}$$

And

$$\bar{a}_l(t) = \min\{\dot{v}_l(t), a_{max}\} \qquad\qquad \Theta(z) = \begin{cases} 1, if\ z \geq 0 \\ 0, else \end{cases}$$

The same two scenarios as before are simulated with different penetration rates; 10, 25, 50, 75, 90, and 100 percent. Fig. 5 shows the flow, gap, speed and acceleration for the vehicles at three penetration rates: 0, 50, and 100 percent, with CACC active and inactive. The equilibrium flow rates from Table 3 are represented by three red lines, equivalent to 3600/θ, where θ is given by equation (6).

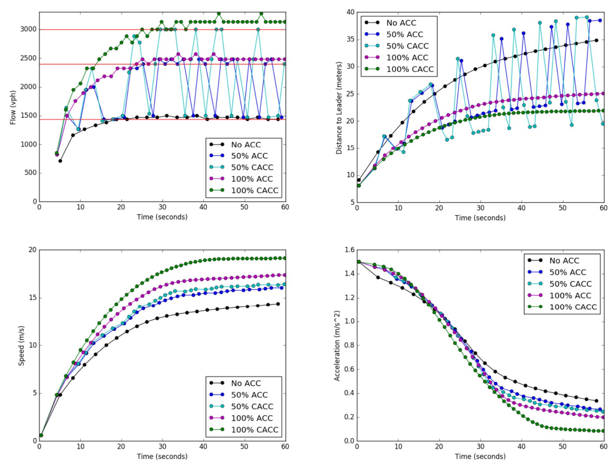

Figure 5: Measurements of flow, distance to leading vehicle, speed and acceleration, speed, and acceleration at the detector location for free flow scenario.

For the blue and teal lines representing 50 percent penetration, the plots switch between two separate lines. The switches correspond to when the vehicle going over the detector switches between being manual and ACC/CACC. In each case, the vehicle roughly follows the curve of the 0 percent or 100 percent scenarios, alternating between the two. For the CACC example, it alternates between three lines since, as discussed previously, CACC vehicles behave as ACC vehicles when behind a manual vehicle. Thus, three behaviors and sets of gap parameters are possible.

The same behavior can be seen in Figure 6 in the scenario that utilizes an additional intersection.

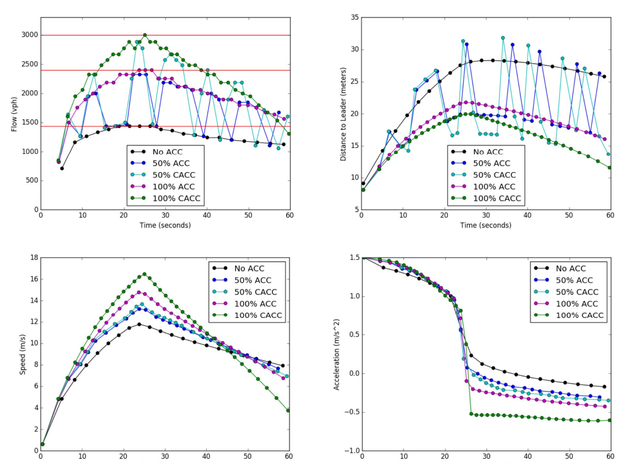

Figure 6: Measurements of flow, distance to leading vehicle, speed and acceleration, speed, and acceleration at the detector location for extra intersection scenario.

For penetration rates between 0 and 100 percent, the ordering of the vehicles can cause high variance in results. For example, if there is a disproportionately high number of ACC vehicles at the front of the queue, it will distort the detected throughput at the intersection under short periods such as one minute. Additionally, the distribution of CACC vehicles among manual vehicles can greatly alter their ability to form platoons, also affecting throughput measurements for small periods. Thus, for mixed-class simulations, 100 one-minute simulations are used and their median vehicle count is extracted. Fig. 7 demonstrates these simulation results, including results for full manual and full ACC/CACC simulations.

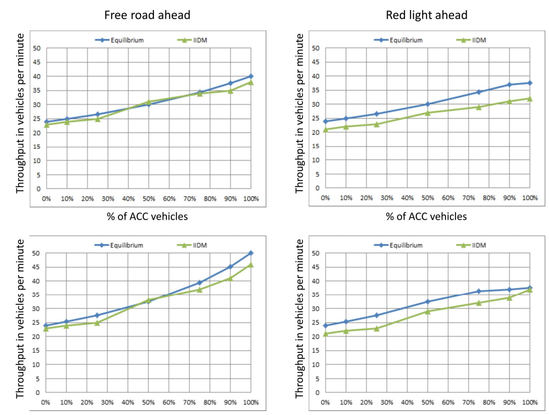

Figure 7: Throughput at intersection as a function of penetration rate. ACC (top) vs CACC (bottom) and scenario one (left) vs scenario two (right).

The blue line in each of the plots corresponds to the equilibrium flow. Take a penetration rate p ∈ [0, 1], corresponding to the fraction of ACC vehicles in the queue. Define τ^{ACC} and g_{min}^{ACC} as the reaction time and minimal gap given in Table 3. The average headway is given by using equation (6):

$$\theta(\lambda) = \lambda\,\tau^{ACC} + (1-\lambda)\tau + \frac{\lambda g_{min}^{ACC} + (1-\lambda) g_{min} + l}{v_{max}} \qquad (9)$$

And the equilibrium flow will correspond to:

$$f(\lambda) = \frac{60}{\theta(\lambda)} \qquad (10)$$

For the scenario with an additional intersection, the flow is further restricted by the capacity of the road segment between the two signals. This results in the following equation:

$$f(\lambda) = \min\{\frac{60}{\theta(\lambda)}, \frac{k\Delta}{\lambda g_{min}^{ACC} + (1-\lambda) g_{min} + l}\} \qquad (11)$$

Where Δ is the length of the road segment and k is the number of lanes in that segment. As previously discussed, our scenario utilizes Δ = 300 and k = 1.

5 Platoons

Vehicles equipped with CACC can form platoons. With 50% CACC penetration rate, platoons provide between 24 and 44% increase in intersection throughput on average, depending on the proximity of intersections.

In simulation, platoon management and formation is divided into three phases: 1) Identifying vehicles that can be grouped into platoons; 2) Adjusting parameters of leaders and followers in platoons; 3) Performing maintenance on the platoon. This behavior is modeled by the state machine in Fig. 8. Leader \ Normal Behavior Follower within range of ACC Vehicle split from platoon Accelerate Decelerate leader accelerates leader decelerates no new instruction.

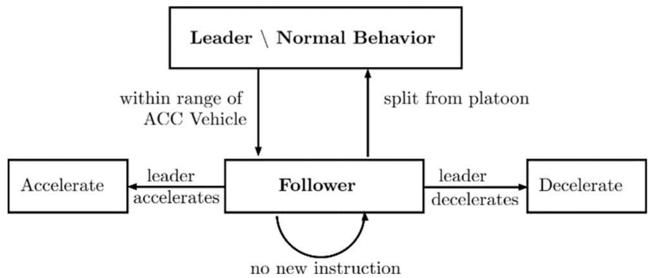

Figure 8: State machine describing behavior of platooned vehicle.

Figure 9: The Huntington-Colorado network (top) and its model in SUMO (bottom).

To form a platoon, vehicles must be in sequence with one another on a given lane. However, vehicles need not share the same final destination and are free to switch lanes or leave the platoon if necessary. If an intermediate vehicle in the platoon changes its route by making a turn or changing lanes, the platoon splits into two: one platoon for the vehicles ahead of the intermediate vehicle and another for all the vehicles behind.

A platoon's lead vehicle has the same properties as ACC vehicles. An isolated CACC vehicle is a leader of a platoon of size 1. When a platoon leader comes into range of another CACC vehicle in front, it joins the platoon becoming a follower. Followers have reduced headway and travel much closer to one another than standalone vehicles. In addition, followers are able to receive information from the leader, such as to accelerate after a green light at an intersection or to decelerate approaching an obstacle, e.g. red light, downstream.

Since followers are not bound to the same route as the platoon leader, they are free to separate. After leaving the platoon, the headway and acceleration parameters are restored to their original values. This can happen for example when the follower changes its route or becomes separated from the rest of platoon, e.g., due to switching traffic signal as it crosses the intersection.

To first study the theoretical potential impact of platooning, we looked at an infinite geometric sequence with value p corresponding to the penetration rate. Given any ACC vehicle, the probability distribution for its platoon size is a negative binomial distribution with n=2, starting at k=1. The sum of two geometric distributions has a distribution given by:

$$f(k\,;p) = k\ *p^{k-1} * (1-p)^2 \tag{9}$$

Fig. 10 shows the distribution of vehicles by size of the platoon they would be a part of in such an infinite train for the 50 percent penetration case. Thus, 25 percent of vehicles would be alone, 25 percent of vehicles would have one other adjacent CACC vehicle, and so on. We can then calculate the percent of vehicles who are followers by excluding the lone CACC vehicles and excluding platoon leaders:

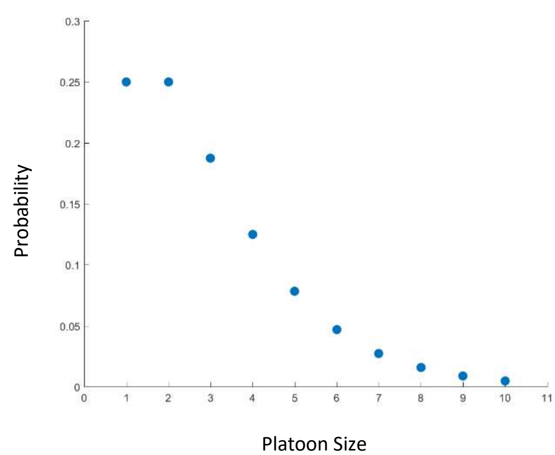

Figure 10: The distribution of platoon size for the 50 percent penetration case.

Lone CACC vehicles: $f(1;p) = \mathbf{(1-p)^2}$

Platoon leaders: $\sum_{k=2}^{\infty}\frac{1}{k} * k\ * p^{k-1} * (1-p)^2 = (1-p)^2 * \sum_{k=1}^{\infty}\ * p^k = \boldsymbol{p * (1-p)}$

And so followers are given by:

$$1 - (1-p)^2 - p(1-p) = \boldsymbol{p} \quad (10)$$

Indicating that followers grow linearly with the penetration rate. In other words, suppose 60 percent of vehicles have CACC technology. Then 36 percent of vehicles will act as CACC vehicles, the remaining 24 percent will act as ACC vehicles, and the other 40 percent will act as manual vehicles. The relationship between flow and penetration rate is thus calculated similarly as in equation (9) through:

$$2.5 * F(p) * (1-p) + 1.5 * F(p) * p * (1-p) + 0.75 * F(p) * p^2 = 3600$$

Which simplifies to:

$$F(p) = \frac{3600}{2.5 - p - 0.75 * p^2} \quad (12)$$

The resulting plot is shown in Fig. 11 in black. The blue line corresponds to the ACC-only case, in which the flow is simply:

$$F(p) = \frac{3600}{2.5 - p} \quad (13)$$

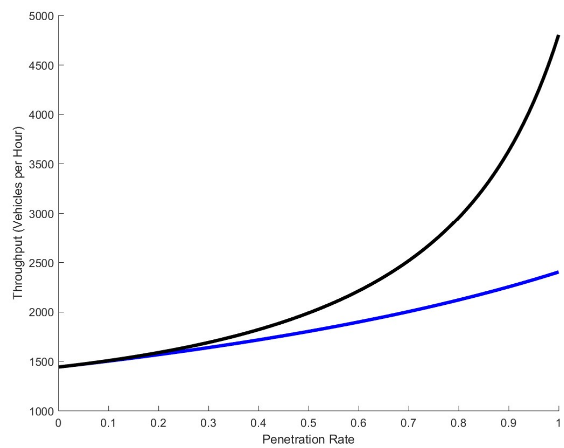

Figure 11: Theoretical throughput as a function of penetration for CACC (black) and ACC-only (blue).

Fig. 11 demonstrates that below 30 percent penetrations, CACC shows very little improvement over ACC since CACC vehicles are not adjacent often enough to form platoons. At roughly 50 percent, there is moderate improvement, but very high levels of penetrations are required for large improvements.

It is worth noting that for the 50 percent penetration case, the simulation performed slightly better than theoretically expected in terms of throughput (24 to 44 percent improvement). This is primarily because regular ACC vehicles underperformed during simulations relative to the expected curve in Fig. 11, whereas the simulations that utilized CACC vehicles were closer to its theoretical curve.

To simulate the practical impact of platooning, we used a SUMO model of the 4-mile stretch of Colorado Boulevard / Huntington Drive arterial with 13 signalized intersections in Arcadia, Southern California, shown in Fig. 9. IIDM and CACC models were implemented in SUMO, and platoon management and formation were handled via SUMO/TraCI API. Using real world ow measurements and estimated turn ratios at intersections, we generated 1 hour of origin-destination (OD) travel demand data. Then, we ran a series of simulation varying the fraction of ACC/CACC vehicles from 0 to 75%. In each simulation two vehicle classes were modeled: ordinary vehicles and ACC (or CACC) vehicles. In simulations with CACC vehicles platoons were formed. The total number of OD pairs in this network is 399. The same number of vehicles was processed in each simulation. The rates and locations at which cars were generated were identical in all scenarios to eliminate the variance in randomly generated routes. For cases of 0, 25, 50 and 75 percent ACC (CACC) penetration rate, we computed average travel time for the route O→D, where O and D identify origin and destination of the selected west-east route in Fig. 9. Table 4 lists the mean travel time (MTT) and its standard deviation (STD), in seconds. As expected, the mean travel time reduces as the fraction of ACC/CACC vehicles increases. Surprisingly the standard deviation also decreases. Furthermore, the travel time of ordinary vehicles is also reduced, although that of ACC/CACC vehicles is reduced more.

ACC/CACC	**Vehicle Class**	**ACC**		**CACC**	
		Median TT	**STD**	**Median TT**	**STD**
0	Manual	653	102	653	102
25 %	Manual	640	96	638	96
	ACC/CACC	605	82	600	76
	All	631	94	629	94
50 %	Manual	583	66	579	60
	ACC/CACC	583	61	570	64
	All	583	64	575	62
75 %	Manual	595	45	583	41
	ACC/CACC	558	58	540	52
	All	567	57	550	48

Table 4: Mean travel time (MTT) and standard deviation (STD) in seconds for varying percentage of ACC vehicles on the main arterial of Fig. 9.

6 Conclusion

Increased penetration rate of ACC vehicles in traffic increased the throughput at all main road segments and reduced travel time for all vehicles, including those that did not utilize the technology. At higher penetrations, CACC vehicles are able to form platoons which further increased the throughput at intersections. However, at lower penetration rates, CACC vehicles become intertwined between manual vehicles, in which case they perform just as effectively as ACC vehicles.

Queues are a significant obstacles in the urban networks simulated, reducing the flow of upstream intersections through backflow. ACC and CACC vehicles reduced the queue sizes at all observed intersections, translating to more efficient flow through intersections. Additionally, simulations show that platoon sizes and improvement matches closely to expected theoretical results. The results on this report corroborate the results in [5], showing that ACC and CACC technology can significantly increase urban road mobility at little cost to infrastructure.

References

[1] NAHSRC, “Automated Highway Demo 97,” https://www.youtube.com/watch?v= C9G6JRUmg_A.

[2] S. Shladover, “Why automated vehicles need to be connected vehicles,” 2013, http://www.ewh.ieee.org/tc/its/VNC13/IEEE_VNC_BostonKeynote_Shladover.pdf.

[3] J. Ploeg, B. T. M. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer, “Design and experimental evaluation of cooperative adaptive cruise control,” Proc. 14th ITSC IEEE Conf, pp. 260–265, 2011.

[4] V. Milanes, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura, “Cooperative adaptive cruise control in real traffic situations,” IEEE Trans. Intelligent Transportation Systems, vol. 15, no. 1, pp. 296–305, Feb 2014.

[5] J. Lioris, R. Pedarsani, F. Tascikaraoglu, and P. Varaiya, “Platoons of connected vehicles can double throughput in urban roads,” Transportation Research, Part C, to appear.

[6] H. S. Mahmassani, “50th anniversary invited articleâĂŤautonomous vehicles and connected vehicle systems: Flow and operations considerations,” Transportation Science, 2016, published online.

[7] M. Treiber and A. Kesting, Traffic Flow Dynamics: Data, Models and Simulation. Springer, 2013.

[8] M. Treiber, A. Hennecke, and D. Helbing, “Congested traffic states in empirical observations and microscopic simulations,” Physical Review E, vol. 62, no. 2, pp. 1805–1824, 2000.

[9] D. Krajzewicz, “Traffic simulation with SUMOsimulation of urban mobility,” in Fundamentals of Traffic Simulation. Springer, 2010, pp. 269– 293, http://www.dlr.de/ts/en/desktopdefault. aspx/tabid-9883/16931_read-41000, accessed 04/28/2016.

Appendix

This section includes relevant code used to implement various functions discussed in this paper. It only includes files that were written by me, though in some cases edited or modified to run in a modular environment.

Below is the code implementation of the IIDM model used in this report:

MSCFModel_IIDM.cpp

```
// ===========================================================================
// included modules
// ===========================================================================
#ifdef _MSC_VER
#include <windows_config.h>
#else
#include <config.h>
#endif

#include <iostream>
using namespace std;

#include "MSCFModel_IIDM.h"
#include <microsim/MSVehicle.h>
#include <microsim/MSLane.h>
#include <utils/common/RandHelper.h>
#include <utils/common/SUMOTime.h>


// ===========================================================================
// method definitions
// ===========================================================================
MSCFModel_IIDM::MSCFModel_IIDM(const MSVehicleType* vtype,
                             SUMOReal accel, SUMOReal decel,
                             SUMOReal headwayTime, SUMOReal delta,
                             SUMOReal internalStepping)
    : MSCFModel(vtype, accel, decel, headwayTime), delta2(delta),
      myAdaptationFactor(1.), myAdaptationTime(0.),
      myIterations(MAX2(1, int(TS / internalStepping + .5))),
      myTwoSqrtAccelDecel(SUMOReal(2 * sqrt(accel* decel))) {
}


MSCFModel_IIDM::MSCFModel_IIDM(const MSVehicleType* vtype,
                             SUMOReal accel, SUMOReal decel,
                             SUMOReal headwayTime,
                             SUMOReal adaptationFactor, SUMOReal adaptationTime,
                             SUMOReal internalStepping)
    : MSCFModel(vtype, accel, decel, headwayTime), delta2(4.),
      myAdaptationFactor(adaptationFactor), myAdaptationTime(adaptationTime),
      myIterations(MAX2(1, int(TS / internalStepping + .5))),
      myTwoSqrtAccelDecel(SUMOReal(2 * sqrt(accel* decel))) {
}


MSCFModel_IIDM::~MSCFModel_IIDM() {}


SUMOReal
MSCFModel_IIDM::moveHelper(MSVehicle* const veh, SUMOReal vPos) const {
    const SUMOReal vNext = MSCFModel::moveHelper(veh, vPos);
    if (myAdaptationFactor != 1.) {
        VehicleVariables* vars = (VehicleVariables*)veh->getCarFollowVariables();
        vars->levelOfService += (vNext / veh->getLane()->getVehicleMaxSpeed(veh) - vars-
>levelOfService) / myAdaptationTime * TS;
    }
    return vNext;
```

```
}


SUMOReal
MSCFModel_IIDM::followSpeed(const MSVehicle* const veh, SUMOReal speed, SUMOReal gap2pred,
SUMOReal predSpeed, SUMOReal /*predMaxDecel*/) const {
    //return _v(veh, gap2pred, speed, predSpeed, veh->getLane()->getVehicleMaxSpeed(veh));
          return _v(veh, gap2pred, speed, predSpeed, MIN2(veh->getLane()->getSpeedLimit(), veh-
>getMaxSpeed()));
}


SUMOReal
MSCFModel_IIDM::stopSpeed(const MSVehicle* const veh, const SUMOReal speed, SUMOReal gap2pred)
const {
    if (gap2pred < 1) {
        return 0;
    }
    //return _v(veh, gap2pred, speed, 0, veh->getLane()->getVehicleMaxSpeed(veh), false);
          return _v(veh, gap2pred, speed, 0, MIN2(veh->getLane()->getSpeedLimit(), veh-
>getMaxSpeed()), false);
}


/// @todo update interactionGap logic to IIDM
SUMOReal
MSCFModel_IIDM::interactionGap(const MSVehicle* const veh, SUMOReal vL) const {
    // Resolve the IIDM equation to gap. Assume predecessor has
    // speed != 0 and that vsafe will be the current speed plus acceleration,
    // i.e that with this gap there will be no interaction.
    const SUMOReal acc = myAccel * (1. - pow(veh->getSpeed() / veh->getLane()-
>getVehicleMaxSpeed(veh), delta2));
    const SUMOReal vNext = veh->getSpeed() + acc;
    const SUMOReal gap = (vNext - vL) * (veh->getSpeed() + vL) / (2 * myDecel) + vL;

    // Don't allow timeHeadWay < deltaT situations.
    return MAX2(gap, SPEED2DIST(vNext));
}


SUMOReal
MSCFModel_IIDM::_v(const MSVehicle* const veh, const SUMOReal gap2pred, const SUMOReal egoSpeed,
                  const SUMOReal predSpeed, const SUMOReal desSpeed, const bool respectMinGap)
const {
// IIDM speed update
    SUMOReal headwayTime = myHeadwayTime;
    if (myAdaptationFactor != 1.) {
        const VehicleVariables* vars = (VehicleVariables*)veh->getCarFollowVariables();
        headwayTime *= myAdaptationFactor + vars->levelOfService * (1. - myAdaptationFactor);
    }
    SUMOReal newSpeed = egoSpeed;
    SUMOReal gap = gap2pred;

    for (int i = 0; i < myIterations; i++) {
        const SUMOReal delta_v = newSpeed - predSpeed;
        // s is S* in IIDM equation
        SUMOReal s = MAX2(SUMOReal(0), newSpeed * headwayTime + newSpeed * delta_v /
myTwoSqrtAccelDecel);

        if (respectMinGap)
            s += myType->getMinGap();

        // This is equation for IDM:
        //const SUMOReal acc = myAccel * (1. - pow(newSpeed / desSpeed, delta2) - pow(s/gap,
delta1));

        ////////////// For IIDM:
        SUMOReal afree;
                   SUMOReal acc = myAccel * (1. - pow(s / gap, delta1));

        if (newSpeed <= desSpeed) { // if we want to speed up or remain (V <= V0)
```

```
                            afree = myAccel * (1 - pow(newSpeed / desSpeed, delta2)); // free
acceleration function

                            if ((s / gap) < 1) { // we are too close to leader
                   acc = afree * (1 - pow(s / gap, delta1 * myAccel / afree));
          }
        }
        else { // if we want to slow down (V > V0)
                            afree = -myDecel * (1 - pow(desSpeed / newSpeed, myAccel * delta2 /
myDecel)); // free acceleration function

                            if ((s / gap) >= 1) {
                   acc += afree;
          }
          else {
                   acc = afree;
          }
        }

        ////////////// End IIDM

                   SUMOReal oldSpeed = newSpeed;
        newSpeed += ACCEL2SPEED(acc) / myIterations;
        //TODO use more realistic position update which takes accelerated motion into account
                   gap -= MAX2(SUMOReal(0), SPEED2DIST((newSpeed - predSpeed) / myIterations));
    }
//    return MAX2(getSpeedAfterMaxDecel(egoSpeed), newSpeed);
    return MAX2(SUMOReal(0), newSpeed);
}


MSCFModel*
MSCFModel_IIDM::duplicate(const MSVehicleType* vtype) const {
    return new MSCFModel_IIDM(vtype, myAccel, myDecel, myHeadwayTime, delta2, TS / myIterations);
}
```

MSCFModel_IIDM.h

```
#ifndef MSCFMODEL_IIDM_H
#define MSCFMODEL_IIDM_H

// ===========================================================================
// included modules
// ===========================================================================
#ifdef _MSC_VER
#include <windows_config.h>
#else
#include <config.h>
#endif

#include "MSCFModel.h"
#include <microsim/MSLane.h>
#include <microsim/MSVehicle.h>
#include <microsim/MSVehicleType.h>
#include <utils/xml/SUMOXMLDefinitions.h>


// ===========================================================================
// class definitions
// ===========================================================================
/** @class MSCFModel_IIDM
 * @brief The Improved Intelligent Driver Model (IIDM) car-following model
 * @see MSCFModel
 */
class MSCFModel_IIDM : public MSCFModel {
public:
    /** @brief Constructor
     * @param[in] accel The maximum acceleration
     * @param[in] decel The maximum deceleration
```

```
     * @param[in] headwayTime the headway gap
     * @param[in] delta a model constant
     * @param[in] internalStepping internal time step size
     */
    MSCFModel_IIDM(const MSVehicleType* vtype, SUMOReal accel, SUMOReal decel,
                   SUMOReal headwayTime, SUMOReal delta, SUMOReal internalStepping);


    /** @brief Constructor
     * @param[in] accel The maximum acceleration
     * @param[in] decel The maximum deceleration
     * @param[in] headwayTime the headway gap
     * @param[in] adaptationFactor a model constant
     * @param[in] adaptationTime a model constant
     * @param[in] internalStepping internal time step size
     */
    MSCFModel_IIDM(const MSVehicleType* vtype, SUMOReal accel, SUMOReal decel,
                   SUMOReal headwayTime, SUMOReal adaptationFactor, SUMOReal adaptationTime,
                   SUMOReal internalStepping);


    /// @brief Destructor
    ~MSCFModel_IIDM();


    /// @name Implementations of the MSCFModel interface
    /// @{

    /** @brief Applies interaction with stops and lane changing model influences
     * @param[in] veh The ego vehicle
     * @param[in] vPos The possible velocity
     * @return The velocity after applying interactions with stops and lane change model
influences
     */
    SUMOReal moveHelper(MSVehicle* const veh, SUMOReal vPos) const;


    /** @brief Computes the vehicle's safe speed (no dawdling)
     * @param[in] veh The vehicle (EGO)
     * @param[in] speed The vehicle's speed
     * @param[in] gap2pred The (netto) distance to the LEADER
     * @param[in] predSpeed The speed of LEADER
     * @return EGO's safe speed
     * @see MSCFModel::ffeV
     */
    SUMOReal followSpeed(const MSVehicle* const veh, SUMOReal speed, SUMOReal gap2pred, SUMOReal
predSpeed, SUMOReal predMaxDecel) const;


    /** @brief Computes the vehicle's safe speed for approaching a non-moving obstacle (no
dawdling)
     * @param[in] veh The vehicle (EGO)
     * @param[in] gap2pred The (netto) distance to the the obstacle
     * @return EGO's safe speed for approaching a non-moving obstacle
     * @see MSCFModel::ffeS
     * @todo generic Interface, models can call for the values they need
     */
    SUMOReal stopSpeed(const MSVehicle* const veh, const SUMOReal speed, SUMOReal gap2pred)
const;


    /** @brief Returns the maximum gap at which an interaction between both vehicles occurs
     *
     * "interaction" means that the LEADER influences EGO's speed.
     * @param[in] veh The EGO vehicle
     * @param[in] vL LEADER's speed
     * @return The interaction gap
     * @todo evaluate signature
     * @see MSCFModel::interactionGap
     */
    SUMOReal interactionGap(const MSVehicle* const , SUMOReal vL) const;
```

```
    /** @brief Returns the model's name
     * @return The model's name
     * @see MSCFModel::getModelName
     */
    int getModelID() const {
        return myAdaptationFactor == 1. ? SUMO_TAG_CF_IDM : SUMO_TAG_CF_IIDM;
    }
    /// @}



    /** @brief Duplicates the car-following model
     * @param[in] vtype The vehicle type this model belongs to (1:1)
     * @return A duplicate of this car-following model
     */
    MSCFModel* duplicate(const MSVehicleType* vtype) const;


    VehicleVariables* createVehicleVariables() const {
        if (myAdaptationFactor != 1.) {
            return new VehicleVariables();
        }
        return 0;
    }


private:
    class VehicleVariables : public MSCFModel::VehicleVariables {
    public:
        VehicleVariables() : levelOfService(1.) {}
        /// @brief state variable for remembering speed deviation history (lambda)
        SUMOReal levelOfService;
    };


private:
    SUMOReal _v(const MSVehicle* const veh, const SUMOReal gap2pred, const SUMOReal mySpeed,
                const SUMOReal predSpeed, const SUMOReal desSpeed, const bool respectMinGap =
true) const;


private:
    /// @brief The IDM delta exponent
          const SUMOReal delta1 = 2;
          const SUMOReal delta2;

    /// @brief The IDMM adaptation factor beta
    const SUMOReal myAdaptationFactor;

    /// @brief The IDMM adaptation time tau
    const SUMOReal myAdaptationTime;

    /// @brief The number of iterations in speed calculations
    const int myIterations;

    /// @brief A computational shortcut
    const SUMOReal myTwoSqrtAccelDecel;

private:
    /// @brief Invalidated assignment operator
    MSCFModel_IIDM& operator=(const MSCFModel_IIDM& s);
};

#endif /* MSCFMODEL_IIDM_H */
```

Below are the core parts of the platoon implementation of the CACC model. They have been modified to run in a modular format:

platoon_functions.py

```
import os
import sys
import optparse
import subprocess
import random
import traci
import settings
import pdb

########## Global variables used in runner file ################################
# settings.platoonedvehicles = []
# settings.platoons = []
# settings.platoonleaderspeed = []
# Note - whenever trying to modify the global variables, they must be referenced
#       as settings.platoonedvehicles or settings.platoons, etc...
################################################################################



################################################################################
# Platoon Control function
#   This function controls the platoons and performs inter-vehicle communication
#   to prevent crashes
################################################################################
def platoon_control(accTau, accMinGap, targetTau, targetMinGap, platoon_comm,time):
	allvehicles = traci.vehicle.getIDList();


	# Go through and make sure all vehicles are still in simulation
	for veh in settings.platoonedvehicles:
		if not (veh in allvehicles):
			settings.platoonedvehicles.remove(veh)

	index = -1

	merge_platoons(targetTau,targetMinGap)


	for platoon in settings.platoons:
		index += 1


		if platoon_maintenance(platoon, accTau, accMinGap, 
allvehicles,targetTau,targetMinGap,time) == -1:
			continue

		# Communication step
		leader = platoon[2]
		try:
			leader_accel = traci.vehicle.getAccel(leader)
			leader_speed = traci.vehicle.getSpeed(leader)
			if len(settings.platoonleaderspeed) > index: # if we are not in the first 
time step for this platoon
				leader_accel = (leader_speed - settings.platoonleaderspeed[index]) 
/ (settings.step_length*platoon_comm)
			else:
				leader_accel = 0
			target_speed = traci.lane.getMaxSpeed(traci.vehicle.getLaneID(leader))

			if (leader_accel < -1.0) or (leader_speed < target_speed):
				for car in platoon[3:]: # go through all followers and have them 
slow down accordingly
					try:
						leading_temp = traci.vehicle.getLeader(car, 100)
```

```
                                        if leading_temp:
                                                dist = leading_temp[1]
                                        else:
                                                dist = 100
                                        if dist < leader_speed * targetTau: # if we're too
close
                                                        traci.vehicle.slowDown(car,
leader_speed, settings.step_length*platoon_comm) # slows down the vehicle for the appropriate
period
                                        continue
                                except:
                                        print("no leader")
                                        continue
                        continue
                except:
                        print("no leader anymore")
                        continue
        del settings.platoonleaderspeed[:] # clears the list
        for platoon in settings.platoons: # records the speed of all platoon leaders to calculate
acceleration # records the speed of all platoon leaders to calculate acceleration
                try:
                        settings.platoonleaderspeed.append(traci.vehicle.getSpeed(platoon[2]))
                        continue
                except:
                        print("platoon leader left simulation")
                        continue


################################################################################
# Platoon Maintenance function
#   This function performs maintenance on platoons by removing vehicles from
#   them for various reasons
################################################################################
def platoon_maintenance(platoon, accTau, accMinGap, allvehicles,targetTau,targetMinGap,time):
        # Remove vehicles that reached destination

        for car in platoon[2:]:
                if not (car in allvehicles): # car not in simulation anymore
                        platoon.remove(car)
                        if car in settings.platoonedvehicles:        # shouldn't  be necessary,
read below
                                settings.platoonedvehicles.remove(car)       # this is
causing issues and it should not. Only started after I moved code to a function, come back to it

        if len(platoon) < 3: # no vehicles in platoon
                settings.platoons.remove(platoon)
                return -1

        if len(platoon) < 4: # only one vehicle in platoon
                try:
                        make_unplatooned(platoon[2], accTau, accMinGap)
                        settings.platoons.remove(platoon)
                except:
                        print("one vehicle in platoon left simulation")
                return -1

        # Check to see lane divergence
        leader = platoon[2]

        try:
                curr_lane = traci.vehicle.getLaneID(leader) #if in middle of intersection, will
give random numbers
                if (curr_lane != platoon[0]) and (curr_lane != platoon[1]) and (curr_lane[:-1] ==
platoon[1][:-1]):
                        # the leader switched lanes within the same road segment, so remove it as
leader
                        platoon.remove(leader)
                        make_unplatooned(leader, accTau, accMinGap)
                        # Configure the new leader
                        leader = platoon[2]
                        curr_lane = traci.vehicle.getLaneID(leader)
```

```
                    make_leader(leader,accTau,accMinGap)


                if (curr_lane != platoon[0]) and (curr_lane != platoon[1]) and (":" not in
curr_lane):
                    # our leader has moved on to a new lane.
                    platoon[0] = platoon[1];
                    platoon[1] = curr_lane
        except:
            print("leader left simulation")
            pdb.set_trace()

        lane1 = platoon[0]; lane2 = platoon[1];

        # Go through follower vehicles
        lane_check = False
        leading_check = True
        flag = False


        # checks whether the leading vehicle is still in the platoon
        if leading_check:
            remove_counter = 0
            index = 2;
            for car in platoon[3:]:
                index += 1;

                try:
                    leading_temp = traci.vehicle.getLeader(car, 100) # gets the car
ahead, up to 100m


                    if leading_temp:
                        curr_leading = leading_temp[0]
                    else:
                        curr_leading = None

                    curr_lane = traci.vehicle.getLaneID(car)
                    # checks leading vehicle but also whether it's this car's lane
which changed -> if it has simply remove it
                    if not (curr_leading in platoon) and (curr_lane != platoon[0]) and
(curr_lane != platoon[1]):
                        remove_counter += 1
                        platoon.remove(car)
                        make_unplatooned(car, accTau, accMinGap)

                        # make_leader(car,accTau,accMinGap)

                        # new_platoon = platoon[1:2] #should be just platoon[1],
but platoon[1:2] makes it an array
                        # new_route = traci.vehicle.getRoute(car) # gets the route
for the leading car
                        # road,lane = get_RoadLane(traci.vehicle.getLaneID(car))
                        # new_platoon.append(get_next_segment(new_route, road)) #
gets the leading car's next segment
                        # new_platoon.append(car)
                        for car2 in platoon[index+1-remove_counter:]: #add +1 to
index,  move cars behind to this platoon to be processed after
                            #new_platoon.append(car2)
                            #traci.vehicle.setColor(car2, (255,255,255,0)) #
Here we can use 255,255,255 to mark platoon splits
                            make_unplatooned(car2,accTau,accMinGap)
                            platoon.remove(car2)
                        #settings.platoons.append(new_platoon)
                        break

                    # if the lane has not changed, it's the leader that has moved, so
make this car the new leader of a new platoon if there are
                    # more vehicles behind it
                    if not (curr_leading in platoon) and ((curr_lane == platoon[0]) or
(curr_lane == platoon[1])):
```

```
                                        remove_counter += 1
                                        platoon.remove(car)

                                        make_leader(car,accTau,accMinGap)

                                        if index == 3 and curr_leading == None: #the leader changed 
route, so remove it from platoon
                                                make_unplatooned(platoon[2],accTau,accMinGap)
                                                flag = True


                                        if index+1 >= len(platoon) + remove_counter: # this is the 
last vehicle in platoon, so don't make a  new platoon
                                                traci.vehicle.setColor(car, (0,255,0,0))
                                                if car in settings.platoonedvehicles:
                                                        settings.platoonedvehicles.remove(car)
                                                if len(platoon) == 4: #last vehicle in platoon, so 
make the leader normal

        make_unplatooned(platoon[2],accTau,accMinGap)
                                                break

                                        new_platoon = platoon[0:1]
                                        new_route = traci.vehicle.getRoute(car) # gets the route 
for the leading car
                                        road,lane = get_RoadLane(traci.vehicle.getLaneID(car))
                                        new_platoon.append(get_next_segment(new_route, road)) # 
gets the leading car's next segment
                                        new_platoon.append(car)
                                        for car2 in platoon[index+1-remove_counter:]: #add +1 to 
index,  move cars behind to this platoon to be processed after
                                                new_platoon.append(car2)
                                                #traci.vehicle.setColor(car2, (255,255,255,0)) # 
Here we can use 255,255,255 to mark platoon splits
                                                make_platooned(car2,targetTau,targetMinGap)
                                                platoon.remove(car2)
                                        settings.platoons.append(new_platoon)
                                        break
                                continue #everything normal
                        except:
                                print("car not in simulation anymore")
                                pdb.set_trace()
                                continue

                if flag:
                        platoon.remove(platoon[2])
                        flag = False

        # uses lane check to filter vehicles
        if lane_check:
                        index = 2;
                        for car in platoon[3:]:
                                        index += 1;
                                        curr_lane = traci.vehicle.getLaneID(car)
                                        if (curr_lane != lane1) and (curr_lane != lane2) and 
(curr_lane[:-1] == platoon[1][:-1]): # vehicle just changed lane
                                        # car has switched lanes or reached a new road
                                                        platoon.remove(car)
                                                        make_unplatooned(car, accTau, accMinGap) # 
remove car and revert it to regular ACC

                                        elif (curr_lane != lane1) and (curr_lane != lane2) and (":" 
not in curr_lane): # vehicles are lagging behind or branched out, split platoon
                                                # car has switched lanes or reached a new road
                                                        platoon.remove(car)
                                                        settings.platoonedvehicles.remove(car)
                                                        traci.vehicle.setMinGap(car, accMinGap)
                                                        traci.vehicle.setTau(car, accTau)
                                                        traci.vehicle.setColor(car, (0,255,255,0))

                                                        leader_route = traci.vehicle.getRoute(car)
```

```
                                                    next_lane = get_next_segment(leader_route,
curr_lane[:-2])
                                                    new_platoon = [curr_lane,
get_next_segment(leader_route, curr_lane[:-2]) + curr_lane[(len(curr_lane)-2):]]
                                                    new_platoon.append(car)
                                                    for car2 in platoon[index+1:]: # move cars
behind to this platoon to be processed after
                                                            new_platoon.append(car2)
                                                            traci.vehicle.setColor(car2,
(255,255,255,0))
                                                            print 'CANT POSSIBLY BE HERE'
                                                            platoon.remove(car2)
                                                    settings.platoons.append(new_platoon)
                                                    #settings.platoonleaderspeed.append() # no
need for this, I believe
                                                    break
        # If platoon is gone, delete it
        if len(platoon) < 4:
                try:
                        make_unplatooned(leader, accTau, accMinGap)
                        settings.platoons.remove(platoon)
                        return -1
                except:
                        return -1

        # make sure leader has correct parameters --> this should not be necessary, check back on
code to see where bug is but it does fix it technically
        try:
                make_leader(platoon[2],accTau,accMinGap)
        except:
                print("leader correct parameters")
        return -1


################################################################################
# Create Platoons function
#   This function creates platoons in a given road segment and cycle time
#   interval
################################################################################
def create_platoons(road, lane, start_range, end_range, accTau, accMinGap, targetTau,
targetMinGap, programPointer):
        road_segment = road + lane;
        if (programPointer >= start_range and programPointer <= end_range):
                first = True # for leader in platoon
                cars = traci.lane.getLastStepVehicleIDs(road_segment)
                platoon = [road_segment]

                # iterate through cars in order of closest to last and check to see if ACC to add
to platoon
                for car in cars[::-1]:

                        # if 'veh2470' == car: #t =1306, platooning creation error somewhere
                        #       pdb.set_trace()

                        # if 'veh282' in car: #veh765' in car:
                        #       aa = ['veh282' in a for a in settings.platoons]
                        #       print (True in aa)
                        #       pdb.set_trace()

                        cartype = traci.vehicle.getTypeID(car)
                        if ("CarA" in cartype) or ("CarIIDM" in cartype):
                                if (car in settings.platoonedvehicles):
                                        # If this vehicle is a leader, first do a check to see if
                                        # car ahead can be the leader instead
                                        if get_platoon(car): #car already in a platoon, don't need
to do anything except check
                                                                                  #if platoon infront
you can join
                                                if car == cars[-1]: #first car in line, nothing you
can join (if not here, itll loop and make a
```

```
                                                                        # a cylical
platoon)
                                                  continue
                                          else:
                                                  if traci.vehicle.getColor(car) ==
(0,255,255,0): #you're a leader
                                                          car_array = cars[::-1]
                                                          front_car =
car_array[car_array.index(car)-1]
                                                          front_pltn = get_platoon(front_car)
                                                          ff = traci.vehicle.getLeader(car)
                                                          dist = ff[1]

                                                          if front_pltn and dist <= 70: #if car
infront is part of a platoon and within 70m, join in
                                                                  behind_pltn =
get_platoon(car)

                                                                  for car_pltnB in
behind_pltn[2:]:

        make_platooned(car_pltnB,targetTau,targetMinGap)

        front_pltn.append(car_pltnB) #add the trailing platoon vehicles to the front one


        settings.platoons.remove(behind_pltn) #get rid of the trailing platoon
                                                                  continue
                                                  else: #you're a follower
                                                          continue

                                  if (traci.vehicle.getColor(car) == (0,255,255,0)):
                                          leading_temp = traci.vehicle.getLeader(car, 100)
                                          # There is a vehicle ahead
                                          if leading_temp:
                                                  type_alt =
traci.vehicle.getTypeID(leading_temp[0])
                                                  platoon_alt = get_platoon(leading_temp[0])
                                                  if (("CarA" in type_alt) or ("CarIIDM" in
type_alt)) and (not platoon_alt) and (leading_temp[1] <= 70): # no, the leading vehicle is not in
a platoon, but it could be and within 70m
                                                          platoon_curr = get_platoon(car)

                                                          if platoon_curr != None: #stupid bug
where cars are technically platooned but not showing up in platoons variable
                                                                  platoon_curr.insert(2,
leading_temp[0])
                                                          make_platooned(car, targetTau,
targetMinGap) # make it a regular follower, instead of a leader


        make_leader(leading_temp[0],accTau,accMinGap)

                                                          first = False
                                                          leader_route =
traci.vehicle.getRoute(leading_temp[0]) # gets the route for the leading car

        settings.platoonedvehicles.append(leading_temp[0])
                                                          continue

                                                  if (("CarA" in type_alt) or ("CarIIDM" in
type_alt)) and (platoon_alt) and (leading_temp[1] <= 70): # yes, the leading vehicle IS in a
platoon, so we can merge and within 70m
                                                          platoon_curr = get_platoon(car) # get
current platoon
                                                          if platoon_curr != None:
                                                                  for veh_alt in
platoon_curr[2::]: # iterate through vehicles in current platoon and add them to the platoon in
front

        platoon_alt.append(veh_alt)
```

```
                                                            make_platooned(car, targetTau,
targetMinGap) # make it a regular follower, instead of a leader

                                                            if platoon_curr != None:

        settings.platoons.remove(platoon_curr) # remove the platoon that merged with the one in
front
                                                            #traci.vehicle.setSpeed(car,
target_speed)

                                                            first = False
                                                            try:
                                                                    leader_route =
traci.vehicle.getRoute(platoon_alt[2]) # gets the route for the leading car
                                                                    continue
                                                            except:
                                                                    print("no leader anymore")
                                                                    pdb.set_trace()
                                                                    continue
                                                            continue
                                                    else:
                                                            continue

                                    if (traci.vehicle.getColor(car) == (255,255,255,0)): #if
already a follower
                                            follower_pltn = get_platoon(car)
                                            leading_temp = traci.vehicle.getLeader(car, 100)
                                            if leading_temp:
                                                    type_alt =
traci.vehicle.getTypeID(leading_temp[0])

                                                    if follower_pltn and (leading_temp[1] <= 70)
and \
                                                    (("CarA" in type_alt) or ("CarIIDM" in
type_alt)): #car belongs to another platoon, but changed lanes so can be part of another one
                                                            platoon.append(car)
                                                            follower_pltn.remove(car)
                                                            if len(platoon) == 3:
                                                                    make_leader(car)
                                            else: #its a follower but not part of a platoon (bug
catcher b/c not possible)
                                                    platoon.append(car)



                                    # Leading car is not a leader, so continue
                                    if len(platoon) == 3: # if there was a single ACC vehicle
                                            make_unplatooned(platoon[2], accTau, accMinGap)
                                    if len(platoon) > 3: # if there were multiple ACC vehicles
                                            settings.platoons.append(platoon) # add the platoon
                                            platoon = [road_segment]
                                    first = True
                                    platoon = [road_segment]
                                    continue

                            if first:
                                    # Checks if the car ahead is in a platoon it can join
                                    leading_temp = traci.vehicle.getLeader(car, 100)

                                    # if car == cars[0] and leading_temp: #
                                    #       if (leading_temp[1] > 70):# if we have a vehicle
which is last in the lane and car infront too far
                                    #                                 # don't make it into a
platoon
                                    #               continue
                                    #       else:


                                    if leading_temp: # and (leading_temp[0] not in
settings.platoonedvehicles): #if there is a platoonable car infront, that becomes the leader and
u become follower
```

```
                                                                    platoon_alt = get_platoon(leading_temp[0])
                                                                    if platoon_alt and (leading_temp[1] <= 70):
# yes, it can join a platoon and within 70m
                                                                            platoon_alt.append(car)
                                                                            make_platooned(car, targetTau,
targetMinGap)
                                                                            #traci.vehicle.setSpeed(car,
target_speed)
                                                                            continue
                                        # elif leading_temp and (leading_temp[0] in
settings.platoonedvehicles): #car infront is in a platoon, giddy up
                                        #       make_platooned(car,targetTau,targetMinGap)
                                        #       platoon_alt = get_platoon()

                                        if car == cars[0]: # and (not leading_temp): #vehicle at
end, with no one infront - don't make platoon
                                                continue

                                        car_array = cars[::-1]
                                        behind_car = car_array[car_array.index(car)+1]

                                        if get_platoon(behind_car): #if the next car is in a
platoon, add that platoon to the front car
                                                first = True
                                                platoon_alt = get_platoon(behind_car)

                                                lead_platoon_alt = platoon_alt[2]
                                                try:
                                                        ff =
traci.vehicle.getLeader(lead_platoon_alt,100)
                                                        dist = ff[1]
                                                except:
                                                        #pdb.set_trace() #IIDM 75, time 240 #DEBUG
HEREEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE##################################################
                                                        continue

                                                if dist <= 70: #platoon is within 70m of the front
vehicle, so mere

                                                        make_leader(car,accTau,accMinGap)
                                                        leader_route = traci.vehicle.getRoute(car) #
gets the route for the leading car

        platoon.append(get_next_segment(leader_route, road) + lane) # gets the leading car's next
segment
                                                        platoon.append(car)


                                                        for cars_pltnB in platoon_alt[2:]:
                                                                try:
                                                                        platoon.append(cars_pltnB)

        make_platooned(cars_pltnB,targetTau,targetMinGap)
                                                                except:
                                                                        print ("follower left
simulation")
                                                        settings.platoons.remove(platoon_alt)

                                                        settings.platoons.append(platoon)
                                                        platoon = [road_segment]
                                                        continue
                                                else: #shouldnt continue platoon formation
                                                        continue

                                        else: #not in platoon, so acc too far or manual - do
regular formation
                                                make_leader(car,accTau,accMinGap)
                                                #traci.vehicle.setSpeed(car, target_speed)
        # set its speed higher to help ease propogation delay
```

```
                                                    leader_route = traci.vehicle.getRoute(car) # gets
the route for the leading car
                                                    platoon.append(get_next_segment(leader_route, road)
+ lane) # gets the leading car's next segment
                                                    platoon.append(car)
                                                    first = False

                                else:
                                            leading_temp = traci.vehicle.getLeader(car, 100)
                                            if leading_temp[1] <= 70 and
(traci.vehicle.getColor(leading_temp[0]) == (255,255,255,0) or\
                                            traci.vehicle.getColor(leading_temp[0]) == (0,255,255,0)):
#if within 70m to make platoon, and the car infront is follower

                                                        #or leader
                                                    make_platooned(car, targetTau, targetMinGap)

                                                    platoon_infront = get_platoon(leading_temp[0])
                                                    if platoon_infront: #this is if a legit platoon
exists infront, if not a platoon is being formed
                                                            platoon_infront.append(car)
                                                            continue
                                                    else: #forming new platoon
                                                            #traci.vehicle.setSpeed(car, target_speed)
               # set its speed higher to help ease propogation delay
                                                            platoon.append(car)
                                                            if car == cars[0]: # this platoon includes
the last car on this segment
                                                                    settings.platoons.append(platoon) #
add the platoon
                                                            else: #theres more cars
                                                                    car_array = cars[::-1]
                                                                    behind_car =
car_array[car_array.index(car)+1]
                                                                    if get_platoon(behind_car): #if the
next car is in a platoon - just end platoon formation here

          #later the merge platoon function will take care of making them 1 platoon
                                                                            first = True

                                                                            if len(platoon) == 3: #no
platoon, just 1 car

        make_unplatooned(platoon[2],accTau,accMinGap)
                                                                                    platoon =
[road_segment]
                                                                                    continue


        settings.platoons.append(platoon)
                                                                            platoon = [road_segment]
                                                                    else: #not in platoon, so regular acc
or manual
                                                                            continue #since if either,
platoon formation continues or gets halted by else case at bottom

                                            else: #the car cannot join the platoon because too far, so
stop the platoon formation here and start another one
                                                    if len(platoon) == 3: # if there was a single ACC
vehicles
                                                            make_unplatooned(platoon[2], accTau,
accMinGap)
                                                    if len(platoon) > 3: # if there were multiple ACC
vehicles
                                                            settings.platoons.append(platoon) # add the
platoon
                                                    first = True

                                                    if car != cars[0]: #its not the last car so we can
still try to make platoons, else we're done
                                                            platoon = [road_segment]
```

```
                                                        make_leader(car,accTau,accMinGap)

                                                        leader_route = traci.vehicle.getRoute(car) #
gets the route for the leading car

       platoon.append(get_next_segment(leader_route, road) + lane) # gets the leading car's next
segment
                                                        platoon.append(car)
                                                        first = False

                      # if it is manual, stop making the platoon, since no cars behind can
accelerate anyways
                      else:
                              if len(platoon) == 3: # if there was a single ACC vehicles
                                     make_unplatooned(platoon[2], accTau, accMinGap)
                              if len(platoon) > 3: # if there were multiple ACC vehicles
                                     settings.platoons.append(platoon) # add the platoon
                              first = True
                              platoon = [road_segment]


################################################################################
# Get next segment function
#   Simply returns the next segment in a vehicles route
################################################################################
def get_next_segment(leader_route, road_segment):
               index = 0
               for segment in leader_route:
                              index += 1
                              if segment == road_segment:
                                             break
               if len(leader_route) > index:
                              return leader_route[index]
               else:
                              return "destination"

################################################################################
# Merge platoons function
#   Combines two platoons if they happen to be on the same road together
#       --covers a bug where you can have two platoons beside each other with
#   --one car that overlaps between the platoons BUT there is no manual
#       --vehicle inbetween preventing the formation of one large platoon
################################################################################
def merge_platoons(targetTau,targetMinGap):
       idxs = []
       for i in range(len(settings.platoons)):
               for j in range(i+1,len(settings.platoons)):
                      platoon1 = settings.platoons[i]
                      platoon2 = settings.platoons[j]
                      # if  (platoon1[0] == platoon2[0]) and (platoon1[1] == platoon2[1]) and \
                      # (i != j):

                      if (i!=j):
                              try:
                                     if (platoon1[-1] == platoon2[2]): #last car of platoon1 ==
first car of platoon2
                                            idxs.append([i,j])
                                     if (platoon1[2] == platoon2[-1]):
                                            idxs.append([j,i])
                              except:
                                     #pdb.set_trace()
                                     print "platoon error somewhere"


       try:
               for k in idxs:
                      idx1 = k[0]
                      idx2 = k[1]
                      platoon2_veh = settings.platoons[idx2][3:]
                      settings.platoons[idx1].extend(platoon2_veh)
                      make_platooned(settings.platoons[idx2][2],targetTau,targetMinGap)
```

```
        except:
                print "middle man car has left simulation"
        try:
                for l in idxs:
                        idx1 = l[0]
                        del settings.platoons[idx1]
        except:
                print "index out of range"

################################################################################
# Get platoon function
#   Returns the platoon the a vehicle belongs to
################################################################################
def get_platoon(veh):
        for platoon in settings.platoons:
                if veh in platoon:
                        return platoon
        return None

################################################################################
# Make Platooned function
#   Sets vehicle parameters to that of a following car in a platoon
################################################################################
def make_platooned(veh, targetTau, targetMinGap):
        traci.vehicle.setType(veh,'CarIIDM')
        traci.vehicle.setMinGap(veh, targetMinGap)   # temporarily set its minimum gap
        traci.vehicle.setTau(veh, targetTau)         # temporarily set its tau
        traci.vehicle.setColor(veh, (255,255,255,0))        # set its color to white, signifying
car follower
        traci.vehicle.setSpeedFactor(veh, 1.5)       # allow it to speed up to close gaps

        if not (veh in settings.platoonedvehicles): # might be leader
                        settings.platoonedvehicles.append(veh)
        #traci.vehicle.setVehicleClass(veh,"IIDM")

################################################################################
# Make Unplatooned function
#   Remove vehicles from being platooned
################################################################################
def make_unplatooned(veh, accTau, accMinGap):
        if veh in settings.platoonedvehicles: # shouldn't be necessary
                settings.platoonedvehicles.remove(veh)

        traci.vehicle.setType(veh,'CarA')
        traci.vehicle.setMinGap(veh, accMinGap)
        traci.vehicle.setTau(veh, accTau)
        traci.vehicle.setColor(veh, (0,255,0,0))
        traci.vehicle.setSpeedFactor(veh, 1.0)

################################################################################
# Make Leader function
#   Make platoon leaders (same parameters as ACC vehicles but cyan color)
################################################################################
def make_leader(veh,accTau,accMinGap):
        traci.vehicle.setType(veh,'CarA')
        traci.vehicle.setMinGap(veh, accMinGap)
        traci.vehicle.setTau(veh, accTau)
        traci.vehicle.setColor(veh, (0,255,255,0))
        traci.vehicle.setSpeedFactor(veh, 1.0)
        if not (veh in settings.platoonedvehicles): # might be leader
                        settings.platoonedvehicles.append(veh)

################################################################################
# get_RoadLane function
#   Get the road and lane that a car is on
################################################################################
def get_RoadLane(path):
        road,lane = path.split('_')
        return road,lane
```