Centralized Automated Traffic Control

Yuyang Guo
yuyangg@andrew.cmu.edu
December 9, 2013

1. Background

Traffic congestion is one of the major limiting factors of the quality of metropolitan areas. It severely impacts the quality of life, economic competitiveness, and largely contributes to excessive pollution. Most current solutions involve limiting the quantity of automobiles on the street, which do not benefit the city traffic in the long term. Therefore, an intelligent centralized traffic control system that could largely increase the capacity of city roads and revolutionize the way people think about trafficing is in demand.

2. Current Situation

We take the ring roads in Beijing as an example since it would be a nice demonstration of why the new centrallized control system would significantly improve the traffic. The ring roads are basically closed circle routes that are designed to serves as an express way to get around the city. However, once the roads are at more than its half capacity, the speed of the cars slow down significantly. (Maybe more research needed here). And the efficiency of road is inversely proportional to the number of automobiles on the route. We propose that there exist a better control system to increase the efficiency of city road usage when it approaches the capacity.

3. Method of Improvement

The most significant reason of congestion for ring roads at its full capacity is the lack of sychronization. That is because of the spring effect when car's velocity and acceleration propagates along the circle when cars start to move. This is fundamentally constrained by the lack of information of the entire system for each individual car. With a fully centrallized system that sychronizes movement along the entire circle, in the ideal case, it is possible to move the automobiles arbitrarily high speed even when it is at its full capacity.

4. Challenges in Synchronization

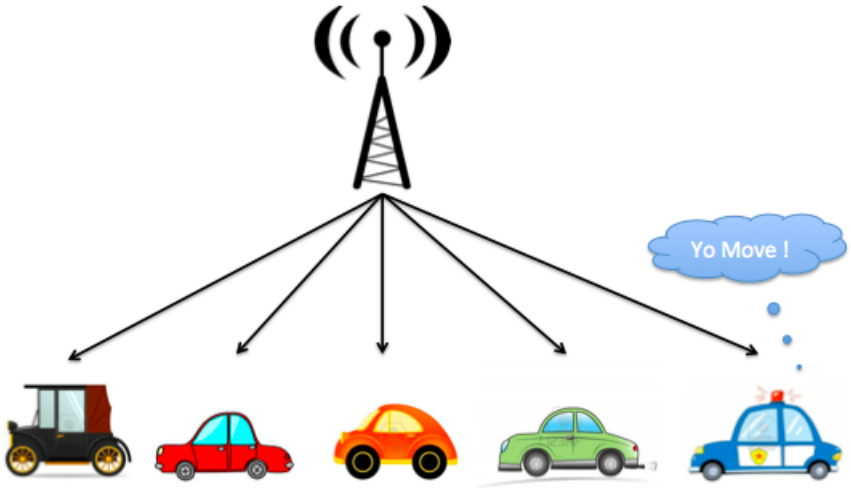
Naturally it is not possible to achieve full synchronization. When the centralized system sends command to each individual automobile involved, each car would receive the signal between some time period $(t - \delta, t + \delta)$, even if we pre-register the time of execution to avoid network latency by sending commands that indicates actions into the future, there would still be execution time skews. Therefore, we would like to prove that even when the cars are sychronized with some limited error in timing, we are still able to maintain safety with some buffer distance between the cars.

5. Control Flow Setup

Consider a system consisting of a Centrallized Controller, and n cars (denoted as a set CARS = $\{Car_i$ where $i \in N \cap [1, n]\}$). The Centralized Controller holds information of where each car should

be at if there were no time delay, as well as where the cars are actually at, whereas individual cars are of ignorant the environment. After the Centralized Controller ensures that some collection of commands $\{Com_i \mid i \in N \cap [1, n]\}$ does not violate system safety, it sends Com_i, which consists of an acceleration command A_i and a timing command T_i indicating how long the acceleration should be executed, to Car_i for all i's. With some time delay between $(0, maxdt)$, each car receives the command and start executing the commands faithfully (meaning, accelerate with rate A_i for exactly T_i).

We risk the professionalism of this paper to include the following picture:



6. Modeling

Instead of modeling the entire system with arbitrarily many cars using distributed hybrid system, we introduce the idea of Reference Car and Buffered Region.

Definition The *Reference Car* is an imaginary car which has the position, velocity and acceleration of a car that moves the same way as if there is no time delay. *Reference Position*, *Reference Velocity* and *Reference Accelaration* refers to the respective attributes of the Reference Car.

Definition A *Buffered Region* is a region around a car's Reference Position. Our safety requirement dictates that each car does not leave its Buffered Region.

With that in mind, we can segment the road into disjoint Buffered Regions. As long as each car stays in its buffered region, the system would ensure no collision. Therefore, a proof for one car that would never leave its Buffered Region suffices to show that the entire system is safe, as long as we make sure that the buffered regions don't coincide. And for our purpose, since we command the cars with same acceleration at the same time, we know that the relative distance between Reference Positions are always the same, so they should never overlap in our system.

7. Proofs

We will prove safety for the following procedure that involves three phases:

Phase I: Acceleration

```
@requiresI: the car starts at its reference position with velocity 0.
@controlI : accelerate the car to a random velocity below its maxv
@ensuresI : the car has some non-zero velocity below its maxv
            and that its position is behind the reference
            and the car's postion error is within a fixed buffer
```

(proof to be found in *final_acc.proof)*

Phase II: Breaking

```
@requiresII: the car has some non-zero velocity below its maxv
             and that its position is behind the reference
             and the car's postion error is within a fixed buffer
@controlII : break till the car comes to a stop
@ensuresII : the car has 0 velocity and its actual position with respect
             to its reference position is within (-buffer, buffer)
```

(proof to be found in final_dec.proof)

Phase III: Adjustment

```
@requiresIII: the car has 0 velocity and its actual position with respect
           to its reference position is within (-buffer, buffer)
@controlIII: car drives from its actual postion to its reference postion
           with some bounded amount of time
@ensuresIII: car is back to its reference postion with velocity 0
```

(proof to be found in final_adj.proof)

We comment on the fact that

$$\text{@ensuresI} = \text{@requiresII}$$
$$\text{@ensuresII} = \text{@requiresIII}$$
$$\text{@ensuresIII} = \text{@requiresI}$$

this means that this full control loop is repeatable.

Since we have the KeYmeara proofs for each individual phase ending in its ensures, we now hand link them together. We also comment on the fact that, even though we did not use double box to prove

that for each phase the entire trace preserves @safety $= (refX - buffer) \leq x \leq (refX + buffer)$, it is actually implied. Because the car and the refCar position are furthest apart at their end position as constrained by the fact that they accelerate/decelerate identically but just by some time difference. Therefore we will assume that in our final hand-linked proof.

(We abbreviate requires as req, control as con, ensures as ens so that the proof might somehow possibly maybe fit into the page. And the proof is attached to the end of file because it needs horizontal formatting.)

8. Analysis and Applications

A general application of this system is to pack cars tightly on a road while keeping the speed very fast. Some examples are ring roads mentioned in the first few sections of this paper, highway, traffic control at busy traffic lights and city traffic in general.

We analyze the improvements from two perspectives: car distance and response time.

Distance

As adviced by California Driver Handbook[1], drivers should follow the three second rule to ensure safety. That is, if the car right in front is driving past a point at time T, then the following car should not pass not that point until $T + 3secs$. To get some concrete data, if the drivers are driving at

$$V = 60mph \approx 27m/s$$

then the safe following distance (D) would be

$$D = 27m/s * 3s = 81m \approx 266feet$$

Now consider the controller we proved, the required distance (RD) between two cars is

$$RD = 2 * buffer = 2 * maxv * maxdt$$

Where maxv is the stable velocity that our car is going at in the end. So here is a chart for RD for different network latencies.

maxdt (secs)	RD (m)	RD (ft)
1	54	177
0.5	27	88.6
0.3	16.2	53.1
0.2	10.8	35.4
0.1	5.4	17.7

Response Time According to an online Reaction Time Study Statistics[2], the human mean reaction time from seeing a sign to action is around 200 miliseconds. Therefore, suppose there are 50 cars in a row, and that it takes 200 miliseonds for the next car to move when it sees the car in front of it moves, then even in the ideal case it would take the last car in row at least 10 seconds to start moving. In comparison, the centralized system would take a few hundred miliseconds to start.

[1] http://apps.dmv.ca.gov/pubs/dl600.pdf, page 47

[2] http://www.humanbenchmark.com/tests/reactiontime/stats.php

9. Limitations

Due to limited time and Computation Resource (yeah, macbook air), our controller is highly simplified and has the following non-exhaustive list of limitations:

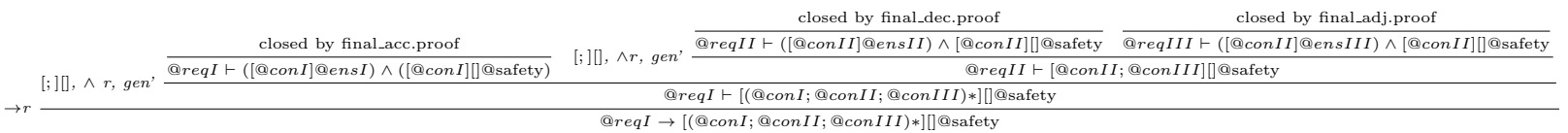
Adjustment Needed Our controller requires that, after accelerating and deceleration cycle, we need an adjustment period where the centralized controller commands each car to drive to its reference position. Even though the adjustment time is bounded by maxdt as proven in final_adj.proof, it is still an annoying property that could be removed if we have a controller that adjust the car to its reference position while driving.

Strict Accelerate Break Sequence Our controller dictates that, after accelerating for some period of time, we cannot directly accelerate again. We must come to a break and go through the adjustment phase until we can re-accelerate. It is also true that for our controller, we cannot start acceleration halfway through breaking. It requires a more complicated proof that we couldn't get to with the limited amount of time.

Failure State Varification we weren't able to prove the failure state is into a safe state. For the controller described above, if one car suddenly breaks down and delerate to a stop during an acceleration. The car should inform the centralized controller and command all the cars to come to a stop. We did not varify that this would not cause too much damage.

10. Acknowledgements

Thanks to Professor André Platzer for so kindly offering this course so that I may be able to take a stab at proving a system that I dreamt of ever since I was 10 years old. Thanks to his son who kindly presented us with little bouncing balls with his angelic smile that inspired us to do great thing in life. Huge thanks to head TA Sarah Loos who devoted to the course so much and graded all the crappy controllers that we came up with initially and providing us infinite amount of support when we are frustrated by formal verification. And thanks to Khalil Ghorbal and Stefan Mitsch for the amazing simulations for the labs and Jan-David Quesel or whoever it was who was one of those course staff that speaks Deutsch (like, everyone) :)


closed by final_acc.proof
closed by final_dec.proof
closed by final_adj.proof
[;]□, ∧ r, gen'
[;]□, ∧r, gen'
→r
@reqI ⊢ ([@conI]@ensI) ∧ ([@conI]□@safety)
@reqII ⊢ ([@conII]@ensII) ∧ [@conII]□@safety
@reqIII ⊢ ([@conII]@ensIII) ∧ [@conII]□@safety
@reqII ⊢ [@conII; @conIII]□@safety
@reqI ⊢ [(@conI; @conII; @conIII)*]□@safety
@reqI → [(@conI; @conII; @conIII)*]□@safety