Adaptive Cruise Control - Proof and Verification

Kashi Vishwanath Kolloju
The Bradley dept. of ECE
Virginia Tech
Blacksburg,USA
kkashivishwanath@vt.edu

Sureshbabu
The Bradley dept. of ECE
Virginia Tech
Blacksburg,USA
dhanush@vt.edu

***Abstract*— Adaptive Cruise Control is one of the basic features that is required for a vehicle to be categorized as an autonomous vehicle. And verification of the control logics plays an important part for the entire system to be successful. In this project we give importance to the proof and verification aspect of the Adaptive Cruise Control and show how KeYmaera X can be used to verify the control logics of the system. The work done in this project sheds light on some of the challenges in implementation and verification of the models. While our project is focused on a static model, the work is not too far from a dynamic obstacle model.**

Keywords—KeYmaera X, Adaptive cruise control, LEAD, EGO

I. INTRODUCTION

A lot of work has been done in Cyber Physical Systems to prove the safety of dynamic systems that interact with the world. This is especially true for autonomous vehicles. One such example is an adaptive cruise-controlled system. This system involves two vehicles: a lead vehicle denoted as the primary other vehicle (LEAD) and a follower vehicle denoted as the subject vehicle (EGO). The objective of this system is to allow the EGO to follow behind the LEAD efficiently while also ensuring the safety of both vehicles. The LEAD vehicle is free to accelerate and brake as it wishes. There already exists models and proofs for this system, but there is a significant amount of additional work needed to implement this in real hardware. There is generally a gap between theory and real systems that must be closed before we can enjoy the benefits of the research already done. Implementing the theoretic models in real hardware has its own challenges. For example, the sensors might not be perfect and might only have access to some of the information required in the model. Besides, there are also challenges on the embedded systems side to manage all the low-level hardware needed to operate the vehicle. There are many challenges to overcome. There are various levels of autonomy ranging from level 0 (L0) to level 5 (L5). L0 vehicles have no automation capabilities, whereas L5 vehicles are perfectly autonomous. Everything in between is a steppingstone towards L5. A vehicle capable of some automation in certain circumstances is generally referred to as L2. This will be similar to the EGO in the project. On the other hand, a very capable autonomous vehicle but restricted to certain types of roads and weather is referred to as L4. Generally, to build a vehicle which has higher level of autonomy is expensive because of sensors and computational power of embedded processors. Therefore, a cost cutting strategy for the purpose of testing would be building one L4 vehicle capable of determining and analysing all the road conditions and traffic signals to make decisions, while having a much cheaper L2 vehicle simply follow the lead vehicle only making decisions based on what the acceleration of lead [2] vehicle is.

II. RELATED WORK

A. Model

The model proposed by Aréchiga et al. [1] on intelligent cruise control system is shown below.

$$\text{ICC} \equiv (\text{ctrl}; \text{dyn})^*$$
$$\text{ctrl} \equiv \text{lead}_{\text{ctrl}} \,||\, \text{ego}_{\text{ctrl}}$$
$$\text{lead}_{\text{ctrl}} \equiv (a_{\text{lead}} := * \,;\, ?(-B < a_{\text{lead}} < A))$$
$$\text{ego}_{\text{ctrl}} \equiv * \,;\, ?(-B < a_{\text{ego}} < A))$$
$$\cup$$
$$(?\text{Safe}_{\varepsilon;}\, a_{\text{ego}} := *; ?(-B < a_{\text{ego}} < A))$$
$$\cup$$
$$?\,(v_{sv}=0) \,;\, (a_{\text{ego}} := 0)$$
$$\text{Safe} \equiv p_{\text{ego}} + v_{\text{ego}}^2/2b + (A/b+1) + (A\varepsilon^2/2 + \varepsilon v_{\text{ego}}) < p_{\text{lead}} + v_{\text{ego}}^2/2b$$
$$\text{dyn} \equiv (t := 0; t' = 1, p'_{\text{ego}} = v_{\text{ego}}, v'_{\text{ego}} = a_{\text{ego}},$$
$$p'_{\text{lead}} = v_{\text{lead}}, v'_{\text{lead}} = a_{\text{lead}} \,\&\, (v_{\text{ego}} \geq 0 \wedge v_{\text{lead}} \geq 0 \wedge t \leq \varepsilon))$$

They have a formal model of the scenario in dL - a modelling language for cyber physical systems - and have also proved it to be safe. The exact model is shown above. In this model, we have a lead car (LEAD) and a follower car (EGO). The model is simply an infinite loop between a discrete controller and the dynamics of the physical world. The controller allows the LEAD to brake at a maximum rate of -B or accelerate at a maximum rate of A. These constraints on the LEAD, model the real-world limits in our motor and braking systems. The controllers allow the EGO car to brake or accelerate within safe limits, but also enforce the condition that the chosen acceleration is safe, and the velocity is non-negative.

If we assume that sensor update delay is bounded by ε, the safety condition we check for is that if we were to accelerate at a given acceleration, then after ε time we can still apply the maximum brakes such that we do not collide lead car. It essentially ensures there is enough distance to accelerate for ε time and then brake to a stop.
After the controls have set the accelerations of the LEAD and EGO car respectively, the model contains the dynamics using ordinary differential equations (ODEs) obeying the laws of physics. This is a repetitive process i.e. infinite loop.

B. Assumptions

- The vehicles move on a 1-dimensional straight road.
- The LEAD vehicle can choose to accelerate or brake arbitrarily.
- Both vehicles can only accelerate within a range -B < 0 < A.
- EGO has a sensor that can get access of LEAD's position and velocity, but not acceleration directly. The sensor is not perfect and has an update delay of ε.

III. PROPOSED MODEL

We have proved two models using dynamic differential logic.

1. Model 1 proves and simulates control logic of LEAD car considering sensor feedback delay rates on a 1-D static obstacle.
2. Model 2 proves control logic of EGO car based on LEAD car cruise.

A. Model 1 in dL

In our model we consider the delay between different feedback rates of the vehicle's sensor e.g. RADAR, LIDAR etc. The different update rate of the sensors essentially makes the whole system running asynchronously [6], and we have to

```
Functions
  Real A;           /* Lead car's max acceleration */
  Real B;           /* Lead car's max braking */
  Real ctrlT;       /* Time-trigger limit on evolution */
  Real posT;       /* Time interval to update obstacle position from lead car
*/
  Real velT;   /* Time interval to update lead car's velocity from sensors */
  Real safedist;  /* Safe distance between lead car and obstacle */
  Real sensorRange; /* Maximum range of lidar sensor.*/  /* used as the
default value of the
     obstacle range when an object isn't detected. */

End.

ProgramVariables
  Real vlead; /* Lead car linear velocity */
  Real alead; /* Lead car acceleration */
  Real obstacleDist; /* Distance to obstacle */
  Real sensedDist; /* Approximate distance to obstacle based on sensor */
  Real sensedV;     /* Approximate velocity of lead car based on sensor */
  Real t; /* Time */
  Real tempV;/* Temporary variable to upper bound a future velocity */
  Real tempDist;
End.
```

```
Problem
  (
    A > 0  &
   -B < 0  &
    ctrlT > 0  &
    posT > 0  &
    velT > 0 &
    safedist > 0  &
    sensorRange > 0  &
    vlead >= 0  &
    sensedV = vlead &  /* Assume initial sensor value is equal to lead car
velocity */
    posT < ctrlT &
    velT < ctrlT &
    obstacleDist >= vlead^2 / (2*B) + safedist &
    sensorRange >= vlead^2 / (2*B) + safedist /* The obstacle distance must be
greater than the safe + current stopping distance (vlead^2 / (2*B)) */

  ) /* Initial conditions */
  ->
  [
    {

        {
          {?(obstacleDist <= sensorRange);/*If we are in range, we assign a
valued to sensedDist that is within some range.*/
            sensedDist := *; ?((sensedDist-obstacleDist) <= (vlead * posT +
0.5 * A * posT^2) & obstacleDist < sensedDist);
          }



          ++
          {?(obstacleDist > sensorRange);
            sensedDist := sensorRange;/*we simply assume the worst case that
there is an obstacle right at the boundary of the sensor range.*/
          }
        }

      sensedV := *; /*Set arbitrary value for sensor velocity of lead car*/
      ?((alead > 0 & (vlead-sensedV) <= A * velT & sensedV < vlead)| (alead <
0 & (sensedV - vlead) <= B * velT & sensedV > vlead) | (alead = 0 & sensedV =
vlead));
      tempV := (sensedV + A*velT);/*If accelerating our sensed v is old so it
should be smaller than the vlead*/
      /*the relationship between sensedV and the vlead is dependent on the
acceleration.*/
      tempDist :=  (sensedDist - (tempV * posT + 0.5 * A * posT^2));
      /* lead car control */
      {

          ?tempDist > (tempV * ctrlT + 0.5 * A * ctrlT^2) + (tempV)^2 / (2*B)
+ safedist;
            alead := A;/*Accelerate when obstacle distance is greater than
current stopping distance((tempV + A*ctrlT)^2/2*B) + safe distance*/
          ++
          ?tempDist > (tempV * ctrlT) + (tempV^2 / (2*B)) + safedist;
            alead := 0;
          ++
            alead := -B;/*Choice for decelerating */
      }
```

ensure safety under this practical setting. Another difference

```
    t := 0;
    /* ODE's */
    {
     obstacleDist' = -vlead,
     vlead' = alead,
     t' = 1 &
     t <= ctrlT & vlead >= 0  /* Evolution Domain */
    }


   }*@invariant(obstacleDist >= (vlead)^2 / (2*B) + safedist & /*obstacle
distance is always greater than stopping distance + safe distance*/
                sensorRange >= (vlead)^2 / (2*B) + safedist &
                vlead >= 0) /* velocity is non negative */
  ](obstacleDist >= (vlead)^2 / (2*B) + safedist & vlead >= 0) /* Safety
condition */
End.
```

is that Aréchiga et al. [1] looks at the system with respect to the absolute positions of the Ego and Lead. In contrast, our sensors give the distance to an obstacle relative to the car's current position. To fix these differences, we propose the following model.

The model above is written in dL, the differential dynamic logic. The benefits of dL are that we can prove our programs to be safe using the KeYmaera X theorem prover [4]. We now explain each part of the model in detail.

Detailed Model

One key feature, of the model that makes it more practical is that it considers the asynchronous updates between different sensors and the control (general real-world issue), and the time delays between these events. Specifically, we will have two sets of values, one set is the sensed values (e.g., sensed obstacle distance sensedDist and sensed car velocity sensedV), and another set is the real values (e.g. obstacleDist and vlead). Our controller is solely based on the sensed values, and the safety conditions are based on the real values. Therefore, one key thing for the model is to use the sensed values to derive correct bounds for the real values, and design a safe controller based on these bounds. We explain this in more detail below.

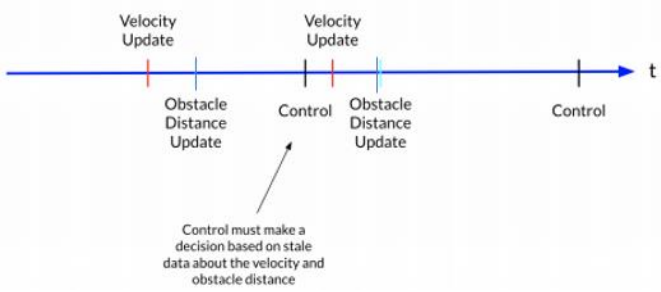

Figure 3.1

As shown in the above Figure 3.1, the model is based on three critical time intervals: ctrlT, posT, and velT. Each time interval is used for a different asynchronous event. The ctrlT interval denotes the time passed between consecutive control decisions. posT denotes the time taken for the car's scanner sensor to update the distance to the obstacle. velT denotes the interval that the car's sensor updates the car's velocity. It is important to note that the control time interval is the largest of the three, and the controller loop is based on that interval. In implementation we assume that sensed distance and velocity are stored in memory denoted as sensedDist and sensedV. At the beginning of each control loop, the controller first reads the sensedDist variable from the memory and uses it to bound the real distance to the obstacle denoted as obstacleDist. There are two cases. The first case is that there is no obstacle within the sensor's effective range (denoted as sensorRange). In that case, the sensor has stored nothing in sensedDist, and we simply assume the worst case that the obstacle is right at the edge of the sensor's range, i.e., we set sensorDist exactly to be sensorRange.
The second case is that the obstacle is within the sensor's effective range, so a certain value is stored in the variable sensedDist. However, since the RADAR sensor updates sensedDist asynchronously with the controller, the value in sensedDist is outdated at the time of the control decision. As the time interval for the RADAR sensor update is posT, we know that sensedDist is at most posT out-of-date, and during this time the vehicle is still moving forward. Therefore, the actual distance between the vehicle and the obstacle must be strictly less than sensedDist, i.e., we have obstacleDist < sensedDist. In addition, we can also derive a lower bound for obstacleDist since in posT time the car could have moved at most (when accelerating) ((sensedDist-obstacleDist) ≤ (vlead * posT + 0.5 * A * posT^2) & obstacleDist < sensedDist).
Similarly, the value of sensedV stored by the car's sensor is also stale at the time of the control decision, but we can use it to bound the car's real velocity vlead. Recall that the time interval that sensor updates sensedV is velT, so sensedV is at most velT time out-of-date. During that time, if the car is accelerating (i.e., a > 0), then its real velocity vlead could be bounded as sensedV ≤ vlead ≤ sensedV + A*velT; if the car is braking, then the real velocity is bounded as sensedV - B *velT ≤ vlead ≤ sensedV; and if the car is either not braking or accelerating, the real velocity should just be the sensed velocity and we have v = sensedV.

Controller

The key idea here is just to use the bounds we derived above in the control strategy. In other words, in the controller, we tempV = sensedV + A * velT as a bound for the real velocity vlead by assuming the car is accelerating with rate A for the velT time since we last got the sensed value sensedV[3].
We then use tempDist = sensedDist * (tempV * posT + 0.5 * A * $posT^2$) as a lower bound for the real obstacle distance obstacleDist, again by assuming the car is accelerating with rate A for the posT time since we last read the sensed data sensedDist. Figure 3.1 shows the asynchronous behaviour between the sensor updates and the controller decisions more visually.

Figure 3.2

The logic for the controller is:
when acceleration (a = A) or moving at a
constant velocity (a = 0), we check that the vehicle would still be able to break safely after ctrlT time. See Figure 3.2 for pictorial explanation. To write this mathematically, we use kinematics to predict the distance travelled by the car in ctrlT time using our tempV, so we can get an upper bound for the real distance the car would have travelled. We then add the distance required to safely stop after we have accelerated with the desired acceleration rate (A or 0) for the control time ctrlT. This calculation is also based on kinematics. Lastly, we add the desired safe distance between the vehicle and obstacle to the previous two distances to get the total required distance in order to safely accelerate or remain at a constant velocity. If the total required distance as calculated is smaller than the lower bound for the obstacle distance tempDist, then we can safely command the desired acceleration rate.
The control is followed by the ODE dynamics [5], which models the physics. The vehicle is moving forward with

velocity vlead. Since the obstacle is not moving, this is the same thing as viewing the distance between the vehicle and obstacle is decreasing with velocity -vlead. Moreover, we know vlead is changing at controlled acceleration rate of A. Time is increasing at unit rate. We have a domain constraint vlead ≥ 0 to ensure that in real world braking does not make the vehicle travel backwards. And another constraint t ≤ ctrlT for our time-triggered controller.

B. Model 2 in dL

In this model we are proving control of EGO car as safe when following a LEAD car. Model of dL logic is shown below.

```
      {
        {?((drel >= dsafe)&(vego <= vset)); vego := vlead ;
/*Match velocity of ego car with lead car - when relative
distance is more than safe distance (speed control)*/
        ++ ?((drel <= dsafe)&(vego <= vset)|(drel <= dsafe)&(vego
> vset)); aego:= -r ; /* assign ego car for decelration when
relative distance is less than safe distance (spacing control) */
        ++ aego := -r;} /* safely assign acceleration or braking
for ego car */
      }
      /* CONTINUOUS DYNAMICS */
      {
        {
            xlead'=vlead,
            vlead'= alead
        }
        {
            xego'=vego,
            vego'=aego,t'=1
            & (vego <= vset & dsafe >= drel)
        }
      }
    }*@invariant(vego<=vset,vego>=0) /* loop invariant */
  ]
  (vego<=vset) /* safety condition */
```

Controller

The idea here is to set the velocity of EGO car as that of LEAD car while accelerating (speed control) maintaining safe distance and checks if velocity of EGO is less than user set velocity vego ≤ vset. We assign acceleration of EGO as -r (retard constant) if relative distance is less than safe distance and vego ≤ vset. We also decelerate by -r if relative distance is less than safe distance and vego > vset (spacing control). The control is followed by the ODE dynamics, which models the physics similar to model1. The EGO car dynamics evolve with constraints when vego ≤ vset and vego ≥0. We have safety condition vego≤vset and drel≥dsafe to make sure ego car does not collide with lead car.

IV. Conclusion

An Adaptive cruise control model simulation were conducted based on KeYmaera X to verify the proposed logic based on the Lead car and Ego car models. We were able to prove the models on KeYmaera X and verify that the conditions used worked. With this method we can verify different logics before implementing the things practically.

In future work, the proposed lead car model was proved with a static obstacle avoidance. We can further continue this for a dynamic obstacle avoidance where keeping a freeway in mind we can control the car to either stop before colliding with the obstacle or change lanes so that the collision can be avoided. On the ego car side, we have considered a mechanism where the ego car follows the lead car blindly and this can be further improved by making the ego car follow the lead car smartly which in turn saves the fuel consumption of the ego car. The proposed models can be used as a base to add constraints and create a more sophisticated model.

References

[1] N. Aréchiga, S. M. Loos, A. Platzer, and B. H. Krogh. Using theorem provers to guarantee

[2] Kyle Hyat and Chris Paukert. Self-driving cars: A level-by-level explainer of autonomous vehicles.

[3] Brandon Bohrer, Yong Kiam Tan, Stefan Mitsch, Magnus O Myreen, and André Platzer. Veriphy:Verified controller executables from verified cyber-physical system models. In ACM SIGPLAN

[4] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp and AndréPlatzer. KeYmaera X: An aXiomatic tactical theorem prover for hybridsystems.
In Amy P. Felty and Aart Middeldorp, editors, *International Conference on Automated Deduction, **CADE'15**, Berlin, Germany, Proceedings*, volume 9195 of *LNCS*, pp. 527-538. Springer, 2015..

[5] AndréPlatzer.
LogicalFoundationsofCyber-PhysicalSystems.
Springer, Cham, 2018. 659 pages. ISBN 978-3-319-63587-3.

[6] J. Vojtech and H. Premysl, "Measurement of noise parameters of CW radar sensors," 78th ARFTG Microwave Measurement Conference, Tempe, AZ, 2011, pp. 1-4, doi: 10.1109/ARFTG78.2011.6183874.

[7] https://www.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html