Reservation-Based Cooperative Intersection Crossing Scheme for Autonomous Driving in the Intersection

Myungwhan Choi, Areeya Rubenecia
Department of Computer Science and Engineering
Sogang University
Seoul, Republic of Korea
{mchoi, rubenecia}@sogang.ac.kr

Hyo Hyun Choi
Department of Computer Science
Inha Technical College
Incheon, Republic of Korea
hchoi@inhatc.ac.kr

***Abstract*—This paper presents a scheme for crossing an intersection with no traffic light where multiple autonomous vehicles coming from different lanes must travel without collision. The focus of this paper is on the design of the reservation-based scheduling algorithm that determines the trajectories of the vehicles at the intersection under the assumption that the vehicles move at a predetermined constant speed in the intersection. Simulation shows that our solution can effectively schedule the entrance of the vehicles to the intersection and maintain the minimum distance between vehicles while traveling in the intersection. It is also shown that our proposed algorithm is fast enough to be used in real time.**

***Keywords*—autonomous driving, intersection traffic management, reservation-based, collision avoidance**

I. Introduction

Conventional traffic system uses traffic lights as road signals to control the flow of traffic at an intersection. However, the use of fixed time controlled traffic light is not efficient because vehicles are required to stop and wait at the entrance of the intersection when the traffic signal is red, even though no other vehicle is currently crossing the intersection. Hence, the use of autonomous intersection crossing scheme is considered as a solution for traffic congestion in a conventional traffic system.

For the autonomous intersection management, the vehicles and the infrastructure can communicate with each other so that the intersection can be utilized in proportion to the actual traffic volume. Consequently, it is necessary to have an efficient intersection crossing scheme that ensures collision-free travel of the vehicles at the intersection. Along with the autonomous driving scheme, this scheme can be used to design the autonomous driving in the intersection.

A variety of systems have been proposed on autonomous intersection management. One approach is reservation-based systems as used in [1], [2], [3]. This kind of system has intersection manager that supervises and grants access of the intersection to the vehicles. In [4], an auction-based scheduling is implemented in which biddings are done to decide the right of way. Decomposition and back-tracking is used in [5]. In [6], an algorithm using ant colony system is developed. In [7], this problem is formulated as a scheduling problem and a mathematical approach is taken using priority graphs. A sequence-based protocol is developed in [8] in which the intersection is managed by giving sequence to the vehicles. In [9], an algorithm is developed using game theory framework.

The focus of this study is to present a reservation-based cooperative intersection management system using red-black trees which turned out to be very efficient data structure for our purpose. From our previous paper [10], we improved the system model to make it more practical; Vehicle size is not fixed, it can be variable. Vehicles can make curve-type turns. The minimum distance between vehicles can be also maintained in the intersection.

This paper is organized as follows: In Section II, the system overview is described which includes the intersection model and the expected behavior of the vehicles. The scheduling algorithm is explained in detail in Section III. Section IV describes the simulations performed to show that the proposed scheme can be implemented in real time and Section V concludes the paper.

II. System Overview

We consider a four-way intersection with three lanes per way as shown in Fig. 1. Each lane has an acceleration zone and queueing zone. Vehicles enter their respective lanes and travel across the intersection. Vehicles can be of any size and can turn left, turn right, or go straight in the intersection. The vehicles must successfully traverse the intersection without colliding with other vehicles.

To ensure collision-free travel of the vehicles in the intersection, the intersection is divided into N by N cells as also shown in Fig. 1. When a vehicle travels across the intersection, it occupies a sequence of cells thus the cells serve as the slots in the intersection that a vehicle will occupy. Therefore, there will be no collision among the vehicles passing the intersection as long as a cell is occupied by only one vehicle at a given time.

In our system, we discretize the time using time step size Δt and simulate the travel of the vehicles every discrete time instant t_i. We set s_{max} as the maximum speed of the vehicle allowed at the road.

How a vehicle maneuvers along the road and intersection is described as follows:

1. Vehicle enters the road and is not allowed to change its predetermined lane.
2. Vehicle travels at s_{max} unless when it should slow down due to queue of vehicles ahead. It can slow down and stop to

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01061155 and NRF-2017R1D1A1B03035324).

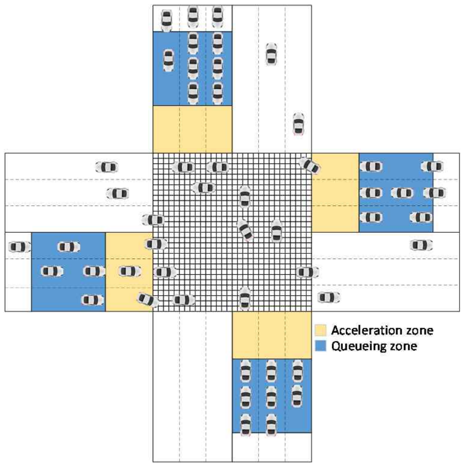


Fig. 1. Model of the four-way intersection with acceleration and queueing zone.

avoid collision with the vehicle ahead and it can accelerate when there is enough space from the vehicle ahead.

3. Vehicle sends its reservation request to the intersection manager as soon as it arrives at the queueing zone.
4. Once the intersection manager receives a request, it schedules the vehicle's entrance to the intersection such that it can travel across the intersection without colliding with other vehicles. After scheduling, it will inform the vehicle of its schedule.
5. The vehicle receives confirmation from the intersection manager that includes the time it can enter the intersection.
6. The vehicle determines the time it must enter the acceleration zone such that it arrives at the intersection at the received scheduled time. The vehicle cannot enter the intersection without being scheduled by the intersection manager.
7. Vehicle traveling along the road controls its speed such that it arrives at the intersection at the scheduled time.

Since a vehicle can only be scheduled when it has entered the queueing zone and it should have already been scheduled before it enters the acceleration zone, it should be inside the queueing zone for at least one time step before it arrives at the entrance of the acceleration zone. Consider a vehicle with no car ahead that is positioned right before the entrance of the queueing zone at discrete time t_i as shown in Fig. 2. If it travels at s_{max}, the fastest speed allowed, then it must be inside the queueing zone at t_{i+1} in order to be scheduled before it can enter the acceleration zone at t_{i+2}. In order for the vehicle at t_i that travels at s_{max} to be at the queueing zone at t_{i+1}, the length of the queueing zone should be at least $s_{max}\Delta t$. Likewise, if the vehicle travels at a slower speed than s_{max}, then it will also be inside the queueing zone at t_{i+1}.

III. Scheduling System

A. Using Red-Black Trees

In our reservation-based scheduling system, to make sure that there is no collision, we check if a cell can be occupied by

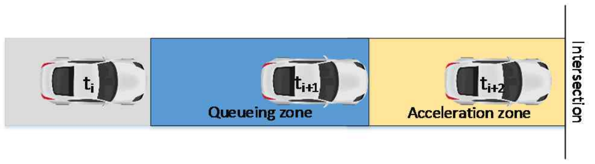


Fig. 2. Length of the queueing zone.

the vehicle on its requested arrival and departure time. This checking is done for each cell that a vehicle needs to occupy.

In order to do this, the major operations to be done to the data are insertion for storing the scheduled reservations; searching from the reservations to check whether the vehicle can enter the intersection without colliding with other vehicles; searching for the time that the vehicle can enter the intersection; and update and deletion to maintain the reservations. For this reason, a data structure that is efficient for this purpose should be used in our algorithm.

Red-black tree is used in our algorithm to store the reservation information of a vehicle to a cell. A red-black tree is a self-balancing binary search tree that keeps the height of the tree as small as possible such that the height of the tree is at most log (n) therefore a balanced binary search tree provides O(log n) searching time.

In our approach, there is a red-black tree for each cell and the nodes in the tree represent the vehicles that are currently reserved in that cell. The node contains the arrival time and departure time of a reserved vehicle in that cell. To determine whether the cell is available on the requested time, the cell's tree is searched to verify that the requested time is not overlapping with any of the existing values in the nodes.

B. Vehicle Generates Request

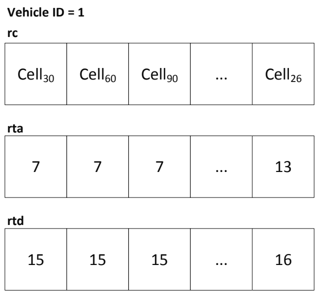
1) Reservation Request Information

When a vehicle sends a request, it provides the following information to the intersection manager:

- Vehicle ID
- *rc*, list of cells it will occupy
- *rta*, list of arrival times corresponding to each cell in *rc*
- *rtd*, list of departure times corresponding to each cell in *rc*

The information for the request which are *rc*, *rta*, and *rtd* are maintained using arrays. There is one array each for the *rc*, *rta*, and *rtd*. An array is a linear list and the elements can be accessed through its index. Let k be the index of the array where $0 \leq k \leq m-1$ and where m is the size of the array and in this case is equal to the number of cells the vehicle will occupy. We define rc_k, rta_k, and rtd_k as the kth element of *rc*, *rta* and *rtd* respectively. The vehicle will arrive at the cell in rc_k after rta_k time steps from the current time and will depart from the cell in rc_k after rtd_k time steps from the current time.

As an example in Fig. 3, the vehicle will arrive at $cell_{30}$ after 7 time steps and depart from $cell_{30}$ after 15 time steps. It will also arrive at $cell_{60}$ after 7 time steps and depart after 15 time steps and so on. The cells in *rc* are sorted in increasing order based on their corresponding arrival time. Since multiple cells can be


Vehicle ID = 1

rc

Cell$_{30}$	Cell$_{60}$	Cell$_{90}$	...	Cell$_{26}$

rta

7	7	7	...	13

rtd

15	15	15	...	16

Fig. 3. Example of a reservation request.

occupied by a vehicle at the same time, there can be similar values among the elements in *rta* and likewise with *rtd.*

2) Generating Reservation Request

Before the vehicle sends a request, it must determine which cells it will occupy and the time it will arrive and depart each cell. This is determined depending on the vehicle's size, speed, and moving direction.

To get these needed information, we simulate the vehicle's travelling movement at the intersection and then determine the cells it occupies at every change in position. This is to be done at each time step until it exits the intersection. As an example in Fig. 4, a vehicle that will turn right will occupy the colored cells at each time step. We can easily check if the vehicle occupies the cell if the vehicle's perimeter lines intersect with any of the cell's borders.

The smoothness of the vehicle's movement depends on the time step size. We get the change in position of the vehicle at every time step thus smaller time step size results to smoother movement of the vehicle in the simulation.

More details on the request generation is described as follows:

1. Initialize *rc*, *rta*, and *rtd* as empty arrays.
2. Compute d_{int} which is the distance to be travelled by the vehicle every time step at the intersection.

$$d_{int} = s_{int}\,\Delta t$$

where s_{int} is the vehicle's speed at the intersection.

3. Compute t_e which is the earliest time the vehicle can reach the intersection from its position when it sent the request. In computing t_e, we assume that the vehicle will travel at s_{max} from its position when it sent the request until it enters the acceleration zone and then while at acceleration zone, it will accelerate to s_{int}.

$$t_e = \begin{cases} \left\lceil \frac{1}{\Delta t}\left(\frac{d_{ca}+l_a}{s_{max}}\right)\right\rceil, & s_{int} = s_{max} \\ \left\lceil \frac{1}{\Delta t}\left(\frac{d_{ca}}{s_{max}} + \frac{2l_a}{s_{int} - s_{max}}\right)\right\rceil, & s_{int} \neq s_{max} \end{cases}$$

where d_{ca} is the distance of the vehicle from its position when it sent the request until the entrance of acceleration zone and l_a is the length of acceleration zone as shown in Fig. 5.

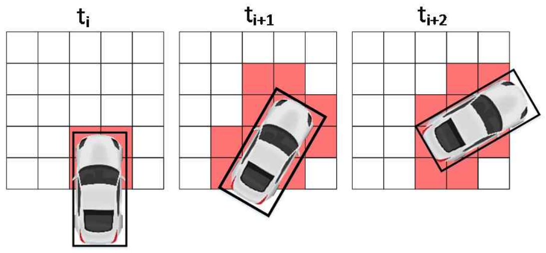


Fig. 4. The cells to be occupied by a vehicle at a specific time step.

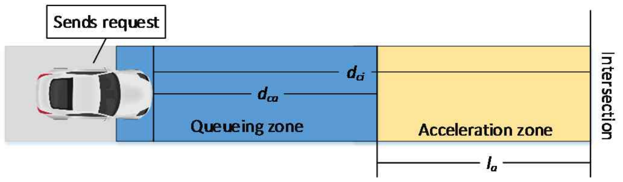


Fig. 5. The d$_{ca}$ and d$_{ci}$ of a vehicle that sent request.

To make sure that the vehicle enters the intersection only after the vehicle ahead has entered the intersection, we set t_e equal to the vehicle ahead's rtd_0 if t_e is less than the vehicle ahead's rtd_0.

4. Minimum distance between cars d_{cc} must be maintained while traveling in the intersection. To achieve this, the arrival time of the vehicle to a cell will be delayed by t_d.

$$t_d = \begin{cases} \left\lceil \frac{1}{\Delta t}\left(\frac{d_{cc}}{s_{int}}\right)\right\rceil, & t_d \geq \Delta t \\ \Delta t, & t_d < \Delta t \end{cases}$$

Note that t_d should be greater than or equal to Δt to be able to delay the arrival time by t_d. Otherwise, if $t_d < \Delta t$ then minimum distance $d_{cc} = s_{int}\ \Delta t$ since $t_d = \Delta t$.

5. Position the vehicle at distance of d_{ci} from its position when it sent the request. Also shown in Fig. 5, d_{ci} is the distance of the vehicle from its position when it sent the request until its position after t_e time steps.

$$d_{ci} = \begin{cases} t_e\,(d_{ca} + l_a), & s_{int} = s_{max} \\ d_{ca} + \dfrac{\left(t_e - \frac{d_{ca}}{s_{max}}\right)^2 a}{2}, & s_{int} \neq s_{max} \end{cases}$$

where a is the acceleration of the vehicle at the acceleration zone and is obtained by

$$a = (s_{int} - s_{max})\left(\frac{s_{int} - s_{max}}{2l_a}\right).$$

6. From this position of the vehicle, at distance of d_{ci} from its position when it sent the request, we simulate the vehicle's movement and get the cell it occupies at each time step until it exits the intersection. Repeat the following steps until the vehicle is completely outside the intersection:
   a. Get the vehicle's position after travelling a distance of d_{int} from its current position.

b. Get the cells that the vehicle occupies on this position.

c. For each cell that the vehicle occupies, if it is not yet in *rc* then insert the cell to *rc*, insert $t_e - t_d - 1$ to *rta* and insert t_e +1 to *rtd*. Otherwise, if the cell is already in *rc*, meaning the cell was already occupied in the previous time step, thus change its *rtd* to the current value of $t_e + 1$.

d. Increment t_e by one for the next time step.

Finally, after simulating the movement of the vehicle in the intersection, we have the values for *rc*, *rta*, and *rtd* which will be the information to be sent to the intersection manager for the reservation request.

C. Intersection Manager Receives Reservation Request

Once the intersection manager receives a request, it must make sure that the requested time interval [rta_k, rtd_k] for the cell in rc_k will not collide with the currently existing reservations on the cell. Thus, there must be no overlap between the requested time and the currently reserved times. When a cell is not available on its requested time, the intersection manager will find the next earliest available time that the vehicle can occupy the cell. Therefore, the vehicle needs to send request only once since when its request is not possible, the intersection manager will immediately find the next earliest possible time it can enter the intersection without colliding with other vehicles.

For each cell in *rc*, we access its red-black tree and search over the tree to check whether the cell is available on the requested time through the function *checkAvailability*. If the cell is available, we go to the next cell in *rc* and continue checking the availability of the remaining cells until we have checked all of the cells in *rc*. On the other hand, when the cell is not available, we search for the earliest time that the cell will be available through the function *findTime*. The result of *findTime* is a number, we define as *inc,* that when added to the requested arrival and departure time is equal to the earliest time that the cell is available. *inc* should be added to all of the elements in *rta* and *rtd*. However, when *inc* is added to all of the values of *rta* and *rtd*, then we must check again all of the previously checked cells in *rc* if the new time of *rta* and *rtd* on those cells are still available. Thus, we repeat *checkAvailability* again from the first cell in *rc* and if the already checked cell is now not available then we proceed to *findTime*. Details on the functions *checkAvailability* and *findTime* are shown in next subsections.

After finding the available time for all of the requested cells in *rc*, the final values in *rta* and *rtd* are the arrival and departure times that vehicle can access each cell the earliest without collision. The values of *rta* and *rtd* are now the scheduled time of the vehicle per cell and their values are inserted to the cells' respective trees. Moreover, rta_0 is the scheduled arrival time of the vehicle to the first cell that it will occupy thus rta_0 is the scheduled time of the vehicle to enter the intersection.

Since the reserved times are relative to the current time, the values in the nodes of all of the trees should be updated every time step by decreasing each arrival and departure time by one at each time step. A node is deleted from the tree when the node's departure time is zero which means the vehicle has already departed the cell.

1) checkAvailability

For each cell in *rc*, the cell's tree is checked if there is conflict with the requested time. This function returns null if the cell is available and returns the node where there is conflict otherwise.

Let *T* be the red-black tree of the cell in rc_k. Let *x* be a node in *T*. Let x_{ta} be the arrival time of node *x*. Let x_{td} be the departure time of node *x*. Let x_l be the left node *x* and x_r be the right node *x*. We go through the nodes of *T* until there is conflict or until the end of the tree has been reached. First, we set *x* to the root of *T*. If $rtd_k \leq x_{ta}$, we go to the left of *x* hence $x = x_l$. Else, if $rta_k \geq x_{td}$, we go to the right of *x* hence $x = x_r$. Otherwise, it means there is a conflict so the cell is not available and node *x* will be the returned value.

As an example we have red-black tree *T* as shown in Fig. 6 and we consider a case where the cell is not available when $rta_k = 15$ and $rtd_k = 20$ since it will conflict with the values in node *D*. First we set *x* to the root of *T* thus *x* = node *A*. $rta_k > x_{td}$ thus $x = x_r$ = node *C*. Then $rtd_k = x_{ta}$ hence $x = x_l$ = node *D*. Then $rtd_k > x_{ta}$ and $rta_k < x_{td}$ which means there is an overlap between [15, 20] and [13, 16] thus cell is not available thus node *D* is returned wherein the conflict occurred.

Another example in which the cell is available is when $rta_k = 3$ and $rtd_k = 5$. First, set *x* = node *A*. $rtd_k = x_{ta}$ hence $x = x_l$ =node *B*. $rta_k = x_{td}$ hence $x = x_r$ = null thus cell is available and null is returned.

2) findTime

When *checkAvailability* did not return null, it means the cell in rc_k is not available at the requested time hence we find the earliest time that the vehicle can occupy that cell. We find the value of *inc*, which is to be added to the values of elements in *rta* and *rtd*.

Let node *x* be the returned node of *checkAvailability* where the conflict occurred. Let *s* be the successor of *x* which is the node whose arrival time is the smallest of all the nodes greater than *x*. Let s_{ta} be the arrival time of *s* and x_{td} be the departure time of *x*. Let $int = rtd_k - rta_k$ and $gap = s_{ta} - x_{td}$.

Starting from *x*, we find *s* such that $int \leq gap$ which means the time interval of the request is less than or equal to the gap between two adjacent nodes. This means we can reserve the vehicle between x_{td} and s_{ta} with new value of $rta_k = rta_k + inc$ and $rtd_k = rtd_k + inc$ where $inc = x_{td} - rta_k$.

Let us use the same example in the previous subsection where the cell is not available when $rta_k = 15$ and $rtd_k = 20$ and red-black tree of cell is same in Fig. 6. As illustrated in Fig. 7, set $int = rtd_k - rta_k$. From *checkAvailability*, conflict occurred at node *D* thus *x* = node *D*.

Then we get the successor of *x*, *s* = node *C*, and we get $gap = s_{ta} - x_{td}$. As shown in Fig. 7, $int > gap$ thus we continue to get the successor of *x*. For the next iteration, we set *x* = *s* = node *C* and *s* = node *E* as illustrated in Fig. 8. Then we compute $gap = s_{ta} - x_{td}$. Now $int < gap$ thus we can compute $inc = x_{td} - rta_k$. The new value of $rta_k = rta_k + inc$ and $rtd_k = rtd_k + inc$ which means

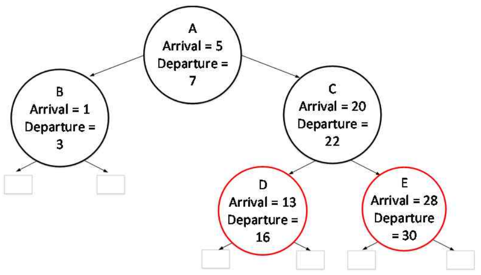


Fig. 6. Example of a red-black tree of a cell.

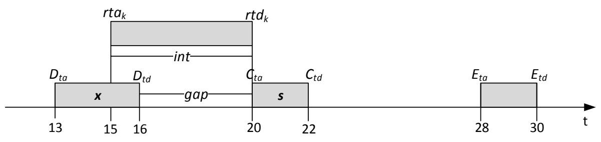


Fig. 7. When x is node D and s is node C then int > gap.

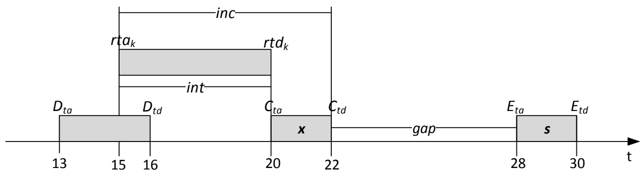


Fig. 8. When x is node C and s is node E then int < gap thus we get value of inc.

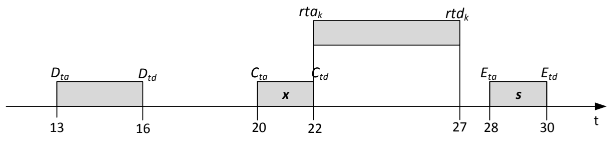


Fig. 9. New values of rta_k and rtd_k after adding inc.

the cell can be reserved to this vehicle at the new time interval [rta_k, rtd_k] the earliest without colliding with other reserved vehicles as illustrated in Fig. 9.

D. Vehicle Receives the Schedule

After the intersection manager has completed the vehicle's scheduling, it will inform the vehicle of t_s which is its scheduled time to enter the entrance of the intersection. A vehicle can only enter the acceleration zone given that it will enter the intersection at t_s.

When there is no car ahead of the vehicle and it is near the entrance of the acceleration zone, it should either stop before the entrance of the acceleration zone or continue to travel and enter the acceleration zone. To determine whether the vehicle can enter the acceleration zone, we do the following:

1. Compute t_a, the time it will arrive at the entrance of the intersection if it continues to travel based on its current position and speed.

$$t_a = \left\lfloor \frac{1}{\Delta t} \left(\frac{d_a}{s_c} + \frac{2l_a}{s_{int} - s_c} \right) \right\rfloor$$

   where d_a is the distance of the vehicle from its current position until the entrance of acceleration zone and s_c is the vehicle's current speed.

2. If t_a is equal to t_s then the vehicle will arrive at the intersection entrance as scheduled hence the vehicle should continue to travel the queueing zone at s_c, then accelerate to s_{int} while at the acceleration zone, and then travel along the intersection at s_{int}.

3. Otherwise, the vehicle must stop at the entrance of the acceleration zone since it will arrive earlier than scheduled if it continues to travel to acceleration zone.

4. If the vehicle stopped at the entrance of the acceleration zone, we decrease t_s by one in the next time step and repeat from the first step until the vehicle can enter the acceleration zone.

IV. Simulation

Simulations were run to show the applicability of the proposed scheme in real time system. Simulation environment is given below:

- Processor of Intel Core i7-4790 with 3.60 GHz clock speed.
- Vehicles are generated with exponentially distributed intergeneration time.
- Vehicles travel the intersection at 10 m/s.
- Vehicle's size is 3 m by 5 m.
- Minimum car distance is 1 m.
- Intersection size is 30 m by 30 m.
- On each way, 20% of the vehicles turn left, 20% turn right, and 60% go straight.

The average of the maximum number of vehicles in the intersection is shown in Fig. 10. In the result, different cells sizes and time step sizes are compared. The cell sizes used are 5 meters (36 cells), 1 meter (900 cells), 0.5 meters (3600 cells), and 0.25 meters (14400 cells). The result shows that the average of the maximum number of vehicles in the intersection increases as the cell size becomes smaller until 0.5 meters. There is no increase in the number of vehicles in the intersection when cell size is smaller than 0.5 meters. Also, comparing the time step sizes of 1 second, 0.5 second, and 0.1 second, using smaller time step size increases the number of vehicles. Thus, it is best to use the cell size of 0.5 meters and time step size of 0.1 second.

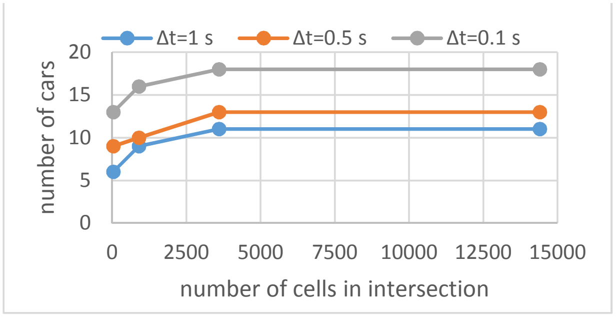


Fig. 10. Comparison of the average of the maximum number of vehicles in the intersection on different cell sizes and different time step sizes.

However, it also worth noting that using an arbitrary small time step size does not always result to better occupancy of the intersection. For example, with the previously specified simulation environment, using time step size smaller than 0.1 second does not improve the occupancy of the intersection hence it is not beneficial to use a time step size smaller than 0.1 second.

Additionally, we must make sure that the scheduling of the vehicles can be completed within the time step size thus the scheduling run time must be less than the time step size. Using time step size of 0.1 second, the simulation result shown in Table I shows the average run time of the scheduling per time step. The result shows that decreasing the cell size results to an increase in the run time per time step as expected. The more important observation is that the average run time per time step is less than the time step size thus the scheduling algorithm can provide the vehicles' reservation schedules within the time step interval. This proves that our proposed algorithm is fast enough to be used in real time when time step size is 0.1 second.

A screenshot of the simulation when cell size is 0.5 meters and time step size is 0.1 second is shown in Fig. 11. This figure shows a screenshot wherein 20 vehicles are in the intersection. In this figure, all vehicles are moving straight in the intersection except for one vehicle which is turning right. The vehicles also maintained the required minimum distance.

TABLE I. AVERAGE RUN TIME OF THE ALGORITHM ON DIFFERENT CELL SIZES WITH TIME STEP SIZE OF 0.1 SECOND

Cell size (m)	Number of cells in intersection	Average run time per time step (s)
5	36	0.0002
1	900	0.0008
0.5	3600	0.0024
0.25	14400	0.0065

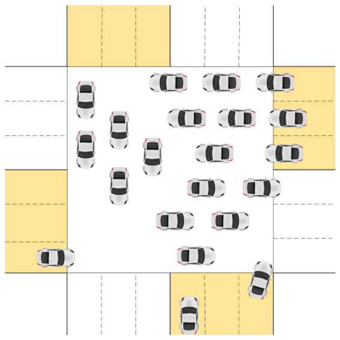


Fig. 11. Screenshot of the simulation with 20 vehicles in the intersection.

V. CONCLUSION

We presented a reservation-based cooperative intersection management system using red-black trees. In our system, vehicles can be of any size, can make curve-type turns, and the minimum distance between vehicles can be maintained.

We showed through simulation that our system is fast enough to be used in real time and so it has potential to be used as a scheduling system for an autonomous driving in the intersection.

Our algorithm, however, works under an assumption that vehicles are moving at a constant speed in the intersection. To make our system more practical, we plan to make our system adaptive to the changes in the moving speed of cars in the intersection.

REFERENCES

[1] A. de La Fortelle. "Analysis of reservation algorithms for cooperative planning at intersections." 13th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 445-449, 2010.

[2] K. Dresner, and P. Stone. "Multiagent traffic management: An improved intersection control mechanism." Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 471-477, 2005.

[3] M. Zhu, X. Li, H. Huang, L. Kong, M. Li, and M. Wu. "LICP: A look-ahead intersection control policy with intelligent vehicles." 6th International Conference on Mobile Adhoc and Sensor Systems, pp. 633-638, 2009.

[4] D. Carlino, S. D. Boyles, and P. Stone. "Auction-based autonomous intersection management". 16th International IEEE Conference on Intelligent Transportation Systems, pp. 529-534, 2013.

[5] J. Wu, A. Abbas-Turki, and A. El Moudni. "Discrete methods for urban intersection traffic controlling." 69th Vehicular Technology Conference, pp. 1-5, 2009.

[6] J. Wu, A. Abbas-Turki, and A. El Moudni. "Cooperative driving: an ant colony system for autonomous intersection management." Applied Intelligence, 37(2), pp.207-222, 2012.

[7] J. Gregoire, S. Bonnabel, and A. de La Fortelle. "Optimal cooperative motion planning for vehicles at intersections." arXiv preprint arXiv:1310.7729, 2013.

[8] F. Perronnet, A. Abbas-Turki, and A. El Moudni. "A sequenced-based protocol to manage autonomous vehicles at isolated intersections." 16th International IEEE Conference on Intelligent Transportation Systems, pp. 1811-1816, 2013.

[9] I. H. Zohdy, and H. Rakha. "Game theory algorithm for intersection-based cooperative adaptive cruise control (CACC) systems." 15th International IEEE Conference on Intelligent Transportation Systems, pp. 1097-1102, 2012.

[10] M. Choi, A. Rubenecia, and H.H. Choi, "Reservation-Based Cooperative Traffic Management at an Intersection of Multi-lane Roads," Proc. of ICOIN 2018, Chiang Mai, Thailand, pp.456-460, Jan. 2018.