Autonomous Intersection Management

Ahmed D. Fayed[1], Alex Winger[2], Dr. Nora Ayanian
[1, 2] Department of Computer Science, University of Southern California, CA
[1] fayed@usc.edu [2] winger@usc.edu

Abstract – *In the duality of the new age of the automotive industry producing increasingly sophisticated and smarter cars and the ever-growing number of vehicles on the road, current intersection technology is becoming outdated, inefficient, and unable to take full advantage of the recent advancement of in-built sensor arrays and artificial intelligent software in modern cars today. In this article, we discuss the potential benefits of implementing new intersection technologies that can harness these added capabilities to alleviate traffic congestion and collisions. We focus solely on fully-autonomous vehicles and describe a centralized system architecture based on dedicated-lanes that provides reservations for vehicles to cross the intersection following a collision-free path that minimizing delays.*

Keywords: Autonomous, Self-driving vehicles, Intersection Management

1 INTRODUCTION

Today, almost all of us have issues and negative criticism on the state of traffic in our lives, especially people living in densely packed urban cities. Human drivers are notoriously known for unpredictability, poor reaction time, and on average, are unaware of good traffic-alleviating driving practices. Such practices like zipper-merging and maintaining a consistent gap between them and the car in front of them to prevent phantom intersections and traffic snakes are common only to a few drivers on the road. Furthermore, human drivers that are intoxicated, sleepy, or exhibit reckless driving cause real tragedies in our societies ranging from loss of life or costly loss of property to a compound form of congestion. In a 2019 report, The Texas A&M Transportation Institute detail that on average we spend 54 hours a year on congestion alone (The Texas A&M Transportation Institute, 2019) [1].

Numerous films or stories set in the future depict worlds that utilize three dimensions of travel; explicitly mass-produced consumer flying vehicles that enable the use of verticality in cities versus the two-dimensional ground lock we experience today. While these worlds are far in the future the benefits of 'unlocking the gridlock' can be measured and strived for. In doing so, we can enable emergency services such as policing, emergency medical services, or fire and rescue service to reach their destination and assist quicker. Even though companies have started exploring this verticality such as boring underneath cities for a new parallel network of roads, Elon Musk's The Boring Company, this technique cannot be implemented in all cities over the world and, not to mention, is incredibly expensive to construct and maintain.

Autonomous driving or self-driving cars remove the drawbacks of ill-stated humans and their inefficient bad practices. They can reliably judge distances, velocities from farther away. They can allow better communication on where each car intends to go, allowing for coordinated symbiotic behavior, and react almost instantly with far less error. This natural progression of travel is necessary when viewed in this light. With many companies introducing autonomous technology built-in to new cars in their fleet as well as start-ups providing an aftermarket package that enables autonomy, it is intuitive to exploit this to renovate our much-outdated traffic management systems.

2 APPROACH

The task of designing fully autonomous transportation from a point A to a point B is complex and hugely wide in nature. We narrow down our focus to intersections since as mentioned in Dresner and Stone's publication around 25-45% of all collisions on the road occur in intersections. Continuing, most autonomous hardware and software designed in modern cars today addresses autonomy that can be activated on highways or continuous segments of road that do not require much vehicle to vehicle coordination. Systems such as pedestrian avoidance and emergency-braking can handle small case scenarios, but few efforts have been made by the automotive industry on technologies that can handle coordinated behavior that would be required in a situation such as a road intersection. This is understandable since these systems would need to be generic enough to work with various automakers and their respective technologies and that is a difficult problem.

Henceforth, we propose a centralized architecture with minimal communication necessary to allow easier adoption and applicability in the real world. This also respects privacy and minimizes the need of sophisticated communication hardware to be included in the vehicles. We work with hardware and software already being introduced into vehicles today and that is becoming the norm or standard in the competitive field of automobiles. Such hardware and software include adaptive cruise control that can achieve desired velocities in a comfortable, reliable, and safe manner; as well as systems that provide steering like lane-keep assist that can execute precise steering control to achieve a trajectory. Our proposed system would invoke these already present architectures to execute specific trajectories through the intersection that should not cause any vehicle to vehicle collisions nor collisions with the environment in sense of curbs, signs, or any static obstacles in the intersection.

To keep a tractable problem in sight we place in certain assumptions such as perfect execution of given trajectories, no communication loss or delays, and no breakdowns of the vehicles anywhere before or after the intersection. An Intersection Manager node would be executing its operations at an intersection that is desired to be automated and assuming that a powerful computer system would be placed that has enough computational resources to perform these computations in real time.

A set distance before or away of the intersection named *dMax* would be the range at which new incoming vehicles are realized by the Intersection Manager (IM) and also serves at the point where these vehicles would start broadcasting their requests to the IM to cross the intersection. This is what we call a reservation. A reservation, if issued, would detail specific coordinates and velocities for the fully autonomous car's state to be at for every timestep along a pre-planned trajectory to cross into its desired lane across the intersection. It is important to note our system does not assume single-integrator dynamics. It considers smooth acceleration, both positive and negative, to achieve set velocities. Expanding, we also assume that cars cannot change lanes once they have entered this *dMax* area and that they are coming in in their desired lane. Explicitly, left lane to execute a left turn, middle lane to go straight through to the opposite cardinal direction across the intersection and right lane to turn right.

If denied a reservation the vehicle would perform the nominal trajectory of stopping at the typical line before entering the intersection area unless a vehicle in front of it is limiting this trajectory at which point it would invoke its own adaptive cruise controller and maintain a distance, car gap, that is greater than or equal to a safety metric *dSafe*. These variables can be set in the system for each intersection depending on intersection-specific characteristics and what is required by the manufacturer following state or country law.

3 IMPLEMENTATION

We decide to implement this in *ROS* since it handles and provides a framework for communication as well as being common implementation used in many

robotics applications using *Python* as the scripting language. We partition the system as follows:

ROS FRAMEWORK[1]

A ROS node named *carManager* (CM) running the process of virtualizing the naturally distributed work that each car would be handling for itself in the real world. It is recomputing continuously following a timestep variable, *timestep_size*. It also contains a function to spawn a certain number of cars, *num_cars* over a set period of time from the start of the simulation, *time_to_complete*. Each car can be spawned with a set of variables: *car_id, lane_id, t, x, y, heading, angular_V, vel, acc, following, reservation, priority, length, width, max_V, max_A, min_A, max_lateral_g.*

A ROS node named *IntersectionManager* (IM) runs the process of granting or rejecting requests from the CM. The IM calculates the trajectory for the requesting vehicle and determines if there is a collision or not.

Once the simulation is complete the list of completed cars is sent to a visualization script *visulizeSim* that uses *RVIZ* to draw the simulation.

CAR MANAGER[1]

The CM, as discussed above, is responsible for virtualizing the cars in the simulation. Using its function to spawn the cars it maintains a list of all the cars where each car has a set of variables storing information about its state at each timestep as well as static variables dictating car limits and its size.

The variable *car_id* stores the id of this car in sequential order. *lane_id* stores which lane the car is in; this does not change. Lane numbers go from 0 corresponding to North lane turning right into West, 1 corresponding to North lane going forward into South, 2 corresponding to North lane turning left into East, 3 corresponding to East lane turning right into North, 4 corresponding to East lane going forward into West, 5 corresponding to East lane turning left into South, and so on. Variable *t* is a list starting from the spawn time of the car increasing by *timestep_size* all the way up to *end_time* which is the end of the simulation time. This usually needs to be set intuitively approximating how long the simulation should take after *time_to_complete* when all vehicles are spawned and in the intersection area *dMax*. Variables *x, y, vel, acc, heading, following,* and *reservation* store those corresponding values for each timestep, so they follow the size of list *t.*

Now that the cars have been spawned, they are spawned with a uniform distribution of velocity ranges that we have set to range from about 40, 45 mph. However, all units are used in meters, seconds, and meters/second in code for consistency and correct mathematics. Each timestep the CM attempts to send reservation requests to each car that has not been reserved yet. If it gets a reservation or if a car was already reserved it allows the car to follow that given trajectory from the IM without manipulation. However, if the car has not been reserved and update() function is called.

This update function works as follows, it computes distances from the car and the final stopping line, *stopping_distance* as well as loops for cars before this car to check for a *follow_car*. This follow car must have the same *lane_id* and must not have entered the intersection, it checks for that by calling *check_for_exit()* which returns True if it has exited the lane and False otherwise. Establishing if a follow car exists or not is essential in invoking the adaptive cruise controller (ACC). Using the initial velocity the car came into the IM's awareness with, which is the first velocity when it spawned or *vel[0]*, the CM computes a nominal deceleration rate and trajectory the car should follow to come to a safe and complete stop at the stopping line. The ACC provides another trajectory to execute following of the follow car. The car manager then picks the minimum acceleration of both trajectories, either to slow down to avoid collision with the car up ahead or to slow down for the line. For example, if the current state (position, velocity, and acceleration) would cause a collision with the follow_car it would invoke the ACC. Otherwise, if invoking the ACC would cause the car to speed up more than is desirable for a smooth deceleration curve to the stopping like then it would select this trajectory (See Fig. 1 below). Details of how the ACC works are not mentioned for the sake of its complexity since it

must account for many different cases and maintain the *dSafe* constraint to guarantee safety and no collisions. There were some scenarios when demanding to spawn a large number of cars in a small-time frame with a large range of initial speeds that the ACC would not be able to realistically slow down the cars in the small-time frame between them entering the *dMax* area. would spawn in. As well as, rounding errors through python that need if statements to catch and set to zero.

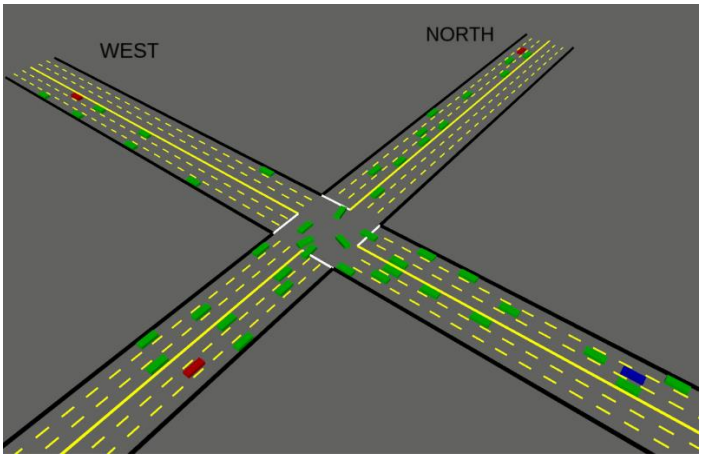

Fig. 1 Showing cars in different colors depicting whether a reservation has been issued [GREEN]. Otherwise the adaptive cruise controller is being invoked [BLUE], or nominal deceleration to intersection line [RED].

INTERSECTION MANAGER[2]

When the Intersection Manager receives a request, it needs to check if the requesting vehicle can make it through the intersection without colliding with any of the vehicles that already have a reservation through the intersection. It does this by granting reservations in a First Come, First Served (FCFS) fashion and through the use of a specific policy. The policy is the set of rules used in determining if a request can be accepted or not, and the Intersection Manager is designed to be able to switch the policy at any time. Four different policies have been developed and compared in this paper, including policies to simulate stop signs and traffic lights, the proposed policy in [2], and a policy we developed.

Traffic Light Policy

A traffic light system consists of many different phases to determine which lights are green at any given time. The system can then cycle through these different phases to allow for vehicles to make it through the intersection safely. Through the use of timers, a lower and upper limit can be put on how long the system is in each of the phases. The minimum and maximum times the system stays in a phase are called the minimum green time and maximum green time respectively. The minimum green time is influenced by driver expectations and queue lengths while the maximum green time can be reset while demand is high for a specific phase and there are no conflicting lanes waiting. For the policy in this project, the traffic light system implements a minimum green time and maximum green time based on the common times discussed in the Signal Timing Manual [4]. There are four total phases which are setup so that phase 1 is the North and South left turn lanes, phase 2 is the North and South straight and right turn lanes, phase 3 is the East and West left turn lanes and phase 4 is the East and South straight and right turn lanes.

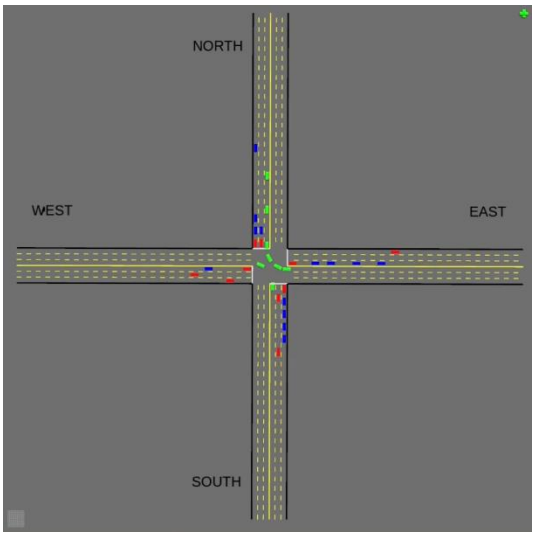

Fig. 2 Traffic Light Policy

Stop Sign Policy

The stop sign policy requires that each of the vehicles must come to a complete stop at the intersection before proceeding through. Once this requirement has been met, the vehicle is able to proceed through the intersection when there is a clear space to do so. Because the entire system has the assumption of being fully autonomous, the vehicles are able to take advantage of smaller gaps in the intersection and move

through in a more efficient manner than what we see with human drivers.

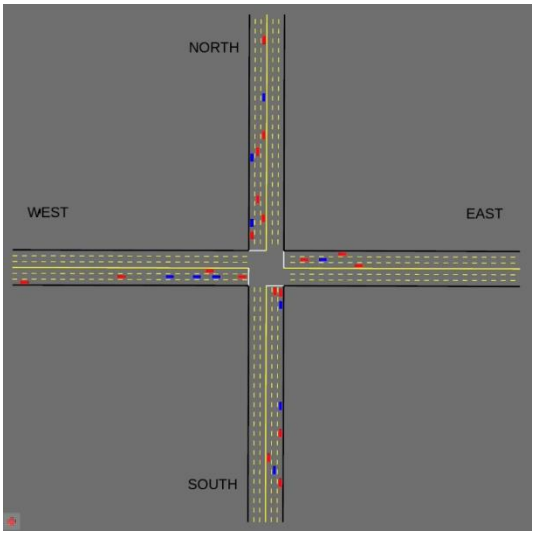

Fig. 3 Stop Sign Policy

Dresner and Stone (Two-check) Policy

Dresner and Stone [2] proposed a policy where they calculate a maximum velocity the vehicle is able to travel through the intersection and create a trajectory based on this velocity and the position of the vehicle. This trajectory is checked for collisions with the already reserved vehicles and if a collision is detected, a new trajectory is created based on the current velocity of the vehicle. The second trajectory is then checked for collisions. If either trajectory is collision free, the request is granted, otherwise the request is rejected. When implementing this policy, we initially understood this max velocity to be the max velocity of the vehicle, rather than a combination of many variables as described in [3]. By the time we realized we were mistaken, there was not enough time to fully implement the exact calculation they make for max velocity as we needed to focus on finishing the Collision-Based Trajectory policy.

Collision-Based Trajectory Policy (CBT)

The policy proposed in this paper creates a trajectory based on the desired velocity and position of the vehicle. The Intersection Manager checks this trajectory for any collisions and returns information about the first collision it encounters. If the trajectory is collision free, then the request is accepted. When a collision occurs, the headings of the two colliding vehicles is used to determine whether the requesting vehicle should slow down or speed up to avoid the collision. To slow down the vehicle, the policy identifies the time the vehicle needs to reach the point of collision for the already reserved car to have made it past and there is no longer a collision. This is done by keeping the position of the car at the point of collision constant and continuously checking if it is still in collision at the next time step until it is clear of collision. To speed up the vehicle, the time is kept constant and the next position of the vehicle along the original trajectory is checked for collision until there is not a collision. Both slowing down and speeding up results in a new point the vehicle must get to, which is used to create a new trajectory from the start to this point and that trajectory is checked for collisions. If there is no collision, the policy then computes a trajectory from this point to the end using the desired velocity of the vehicle. In the event of a collision, the trajectory is recomputed using the velocity of the vehicle when it reaches the new point. If both trajectories end portions result in a collision, the first part of the trajectory is recalculated using the opposite speed change as before. For example, if the first part of the trajectory is calculated by speeding up the vehicle but there is always a collision, then the process repeats by trying to calculate the trajectory by slowing down the vehicle. If a collision free trajectory is found, then the request is granted, otherwise it is rejected.

Collision Detection

In order to do collision detection, the intersection was discretized into a grid and when a vehicle is granted a reservation, any cell the car is touching is marked as occupied. The grids are stored in a circular array so they can be accessed when needed and old states can be cleared and reused at a later time. The Intersection Manager uses the times of each point in the trajectory to know which intersection states to look at. It will grab that state from the array and attempt to place the requesting vehicle on the grid. This is done by getting the four corners that make up the bounding box of the vehicle and then rotating and translating them to match the position and heading given at the point in the trajectory. In order to mark the cells in the grid, two points that are connected are selected, the left most point is used as the starting point and the right most point is the ending point. The cell which contains

the starting point is marked as occupied and then a reference point is selected to determine which cell the line goes through next. This reference point is the top right corner for positive sloped lines and the bottom right for negative sloped lines. The slope of the line connecting the starting point and the reference point is calculated and then used to determine which cell to mark next. For positive sloped lines, if the slope to the reference point is greater than the slope of the actual line, the cell to the right is marked, otherwise the cell above is marked. For the negative sloped lines, a larger slope to the reference point means the cell to the right is marked and a smaller slope means the cell below is marked. The marked cell is then used to get a new reference point and the process continues until the cell containing the ending point is reached. If at any point the cell that is trying to be marked is already marked as occupied by another vehicle, the trajectory is not considered collision free and therefore a request cannot be granted based on that trajectory.

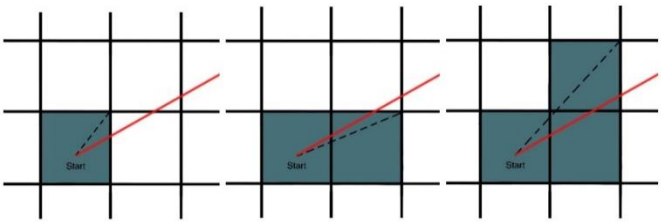

Fig. 4 Process of Marking Occupied Cells

There are some parameters that should be considered during this process. The first is the granularity of the grid. If the granularity is too large, then vehicles in adjacent lanes could be considered in collision due to the lanes going through the same lane. On the other hand, if the granularity is too small, the computation time will increase for each check in the process. The next important parameter to consider is the parameter used to maintain a specified safe distance between each vehicle (dSafe). This can be done by expanding the bounding box for a vehicle by half of dSafe. This ensures that even if adjacent cells of the grid are marked for two different vehicles, there is this guaranteed safety distance.

VISULIZATION[1]

The script draws the environment consisting of lanes, lane markings, and cardinal direction annotations using variables that can be set for different intersection dimensions such as *dMax, lane_width, timestep_size, end_time*. It also uses the *following* and *reservation* variables to impose a color on the vehicles, displayed by rectangles the size of *length* and *width*. This visualization is redrawn at each timestep following the rate given by *timestep_size* mimicking the CM's rate for correct and accurate visualization. At each timestep the car's *x, y,* and *heading* variables are used to correctly depict it in the simulation as can be seen in Fig. 1. If at this current timestep the car has been issued a reservation, then it is drawn in green. Otherwise, if at this current timestep the car has not been issued a reservation yet and is invoking its ACC to slow down to maintain at least a *dSafe* distance from the car in front of it then the variable *following* corresponds to True and hence the car is drawn in blue. Finally, if the car has not been issued a reservation and is not following a car in front of it then it is drawn in red.

4 RESULTS

For this project, we ran the simulation under many different scenarios. Each set of vehicles that was generated was run through the simulation using each of the policies mentioned in this paper. The table below shows the delays for each policy when a certain number of cars were spawned over a set time, 20 seconds for this experiment. The delays show that the Collision-Based Trajectory policy is much more efficient than the methods of control present today, stop signs and traffic lights. In addition to this, it is also seen that the Collision-Based Trajectory policy is more efficient than the two-check policy presented in [2]. Unfortunately, these numbers cannot be used to conclusively say there is an improvement in delay using the policy presented here over the policy presented by Dresner and Stone because we were not able to implement their method exactly as described in the implementation section above.

Policy	20 cars/ 20 sec	40 cars/ 20 sec	60 cars/ 20 sec	80 cars/ 20 sec	100 cars/ 20 sec	120 cars/ 20 sec
Collision-Based Trajectory	0.031	0.068	0.084	0.206	0.326	1.957
Dresner/ Stone	0.026	0.057	0.105	0.575	0.667	2.898
Traffic Light	12.078	11.485	14.336	24.584	26.250	25.607
Stop Sign	10.610	12.849	15.846	19.062	22.582	23.050

Table 1: Average delays (seconds) of each policy over simulations with different spawn rates

One interesting behavior that showed up in the results is that the stop sign policy had lower delay than the One interesting behavior that showed up in the results is that the stop sign policy had lower delay than the traffic light policy in many of the simulations, including those with higher spawn rates. There are a few reasons that could be causing this unexpected behavior. The first reason is that the traffic light is mostly based just on the timers set for changing the phase, so a vehicle which shows up right after a phase change has to wait for the phase to cycle back to its lane for it to go through the intersection. The second reason is that the stop sign policy is implemented to also assume for every vehicle being fully autonomous. This allows for a vehicle to be able to move through the intersection sooner than with human drivers can today. The combination of having the traffic light implementation increasing its delay a bit and the optimality of the stop sign policy decreasing its delay, it makes sense for the stop sign policy to be more efficient than the traffic light policy.

5 STRENGTHS & WEAKNESSES

The biggest strength in the system we developed for this project is its robustness and capability to be expanded further than what we have done here. By allowing the system to change policies easily, traffic can be controlled differently at any moment, which is beneficial since traffic is always changing with time of day, week and even season. Changing policies also allows an intersection manager to get emergency vehicles through an intersection in the fastest possible way and allow for pedestrians to cross safely. In addition to being able to switch policies, the intersection manager can be modified to apply these policies to different types of intersections and lanes, not just the four-way three-lane dedicated intersection we used for experiments. Another strength of the system is that the Collision-Based Trajectory policy can be expanded to account for multiple collisions instead of just the one that was done for this project. This allows for a collision free trajectory to be found in segments so the vehicle can accelerate or decelerate to avoid the other vehicles in the intersection.

Although there are plenty of strengths in the system, there are also a few weaknesses. The biggest of these weaknesses is the reliance on not having any failures or errors in the system. The system is not capable of accounting for vehicles which break down in the intersection, or those that are not able to complete their given reservations. If a vehicle breaks down in the intersection, the current Intersection Manager will not be aware of this and will still assign that space of the intersection to other vehicles, causing collisions. Another weakness is the assumption of perfect communication. Communication failures could lead to issues such as the one just mentioned, or it could cause vehicles to stop at the intersection and wait for a reservation even if the intersection is clear. One final weakness of the project is the abstraction of complex systems to help with controlling a fully autonomous vehicle, such as an adaptive cruise controller. We attempted to create these systems as simple as possible to turn the focus of the project to the Intersection Manager, but this caused many small issues later down the line when testing, so we needed to spend time improving some of these systems.

6 RELATED WORK

Kurt Dresner and Peter Stone have done a lot of research in the area of Autonomous Intersection Management and have developed multiple simulators to show their methods which apply to both autonomous and hybrid systems. We used many of the ideas they address in their work [2] and [3], for fully autonomous systems, in developing our system. Some of the key parts are the ideas of discretizing the intersection into a grid and allowing the vehicles to reserve space in that grid in a First Come, First Served order. In their work,

Dresner and Stone say that due to acceleration, there could be an infinite number of trajectories a vehicle could follow to go through the intersection. Because of this fact, they discussed a couple of policies they attempted to use that did not involve the acceleration too much. They mention their first attempt failed because it would cause a deadlock between vehicles once they started to move too slowly because the policy would attempt to reserve vehicles solely based on its current velocity. The next approach they attempted was to check a trajectory where the vehicle accelerates to a max velocity and then a trajectory where the vehicle maintains its current velocity.

Many of the points discussed in the Strengths and Weaknesses section are also addressed in [2] and [3]. The implementation by Dresner and Stone accounts for communication failures, vehicles not being able to follow a granted reservation exactly, and the idea of policy switching. They include timeouts for requests so the vehicles are not stuck waiting for a response if a request is dropped, while also allowing vehicles to send update and cancel messages to the Intersection Manager in the case they are not able to make their reservations. In addition to this, their implementation accounts for errors in positioning and velocity by creating a buffer around each vehicle when it is reserving space in the intersection.

7 SUMMARY

In this article we have presented the motivation behind our reservation-based intersection controller for handling fully autonomous vehicles that follows a great deal Dresner and Stones'. We have explained in detail the implementation of our system and minimalistic communication protocol using a ROS framework consisting of a Car Manager, an Intersection Manager, and the visualization of the simulation.

Firstly, the Car Manager represents the virtualization of the fully autonomous cars and implements the sensing, computation, and execution of what would be in the real-world a complete adaptive cruise control system built-in into the cars. This system would typically follow speed limits, follow with a desired safe distance, and eventually drive the car to a complete stop at the typical stopping line before the intersection. Moreover, we discussed the Traffic Light policy mimicking a basic traffic light system, the Stop Sign policy also mimicking as typical 4-way stop sign, Dresner and Stone's two-check policy, as well as our CBT policy. We have shown results in the form of a simulation and average computed delay metric, following Desner and Stone's description of delay, that allowed a direct comparison of our policy's exhibited delay with the others. Thus showcasing how we can achieve near optimal delays and tackle intersection collisions in low and high traffic scenarios.

8 REFERENCES

[1] The Texas A&M Transportation Institute. *2019 Urban Mobility Report.* Texas, Aug. 2019. Retrieved from https://static.tti.tamu.edu/tti.tamu.edu/documents/mobility-report-2019.pdf

[2] K. Dresner and P. Stone, "A Multiagent Approach to Autonomous Intersection Management," Journal of Artificial Intelligence Research, vol. 31, no. 1, pp. 591–656, Jan. 2008

[3] K. M. Dresner, *Autonomous Intersection Management.* University of Texas at Austin, Dec. 2009. Retrieved from https://repositories.lib.utexas.edu/handle/2152/ETD-UT-2009-12-689

[4] National Academies of Sciences, Engineering, and Medicine 2015. Signal Timing Manual - Second Edition. Washington, DC: The National Academies Press. https://doi.org/10.17226/22097.

9 APPENDIX

Superscript shows which team member contributed to the work. Superscript numbers correspond to the authors at the top of the report.

Github link to project repository: https://github.com/adfayed/aim_ros