# Automation Intersection Management

Charles Arsenal Okere
*B. Eng (ELE)*
*Hamm-Lippstadt University of Applied Sciences*
Lippstadt, Germany
charles-arsenal.okere@stud.hshl.de

Muhammad Amirul Hakimi Bin Zaprunnizam
*B. Eng (ELE)*
*Hamm-Lippstadt University of Applied Sciences*
Lippstadt, Germany
muhammad-amirul-hakimi.bin-zaprunnizam@stud.hshl.de

Muhammad Amjad Bin Abd Malik
*B. Eng (ELE)*
*Hamm-Lippstadt University of Applied Sciences*
Lippstadt, Germany
muhammad-amjad-bin.abdul-malik@stud.hshl.de

Muhammad Iqbal Bin Mohd Fauzi
*B. Eng (ELE)*
*Hamm-Lippstadt University of Applied Sciences*
Lippstadt, Germany
muhammad-iqbal.bin-mohd-fauzi@stud.hshl.de

Muhammad Farid Izwan Bin Mohamad Shabri
*B. Eng (ELE)*
*Hamm-Lippstadt University of Applied Sciences*
Lippstadt, Germany
muhammad-farid-izwan.bin-mohamad-shabri@stud.hshl.de

***Abstract*—The increased pollution created by automobiles stopped in heavy traffic in big cities is highly costly as compared to when traffic moves freely. Self-driving car technology has come a long way in recent years. Assisted and semi-automated driving technologies are now available, with optimistic estimates for completely autonomous vehicles in the future decades. As the environment changes, congestion relief in situations including driverless or semi-autonomous vehicles faces new opportunities and obstacles. The goal of this research is to improve intersection traffic flow by allowing vehicles to travel freely and securely. One of the most stressful components of any transportation network is intersections. Because the hubs' function is to coordinate several traffic patterns, each with its own set of goals and preferences, this tension is inherent. The problem was solved by creating three ways for modeling any intersection, identifying the paths with the fewest possible points of confusion between their in and out directions, and using queuing theory to optimize the vehicle arrival rate for maximum intersection performance.**

## I. Introduction

Intersections are frequently traffic barriers in a city's road network. The management of intersection traffic has long been a prominent research topic. Traditionally, signals have been employed to allow conflicting traffic streams' rights of way. Several studies have been conducted to optimize traffic signals that are either fixed-time control, actuated control, or adaptive control to improve the efficiency of transportation networks [1]. Nonetheless, signal delays continue to put substantial pressure on urban transportation networks. In the United States, for example, 16 billion vehicle hours of delay are predicted on major highways [2]. With the advent of CAV technology, new avenues for traffic control have become available. Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications, for example, enable the transmission of real-time traffic information from intersections to vehicles for trajectory planning, as well as the collection of comprehensive vehicle trajectory data, including speeds and locations, for signal control. Signal optimization with real-time CAV trajectory information is a simple application [3] that complements infrastructure-based detector data. To reduce energy/fuel use, emissions, delay, and other issues, CAV trajectories can also be optimized by adjusting vehicle speeds or acceleration rates in response to predetermined signal timings. [4]. There is also some literature [5] on a unified framework for optimizing both traffic signals and vehicle trajectories simultaneously. Under the assumption of ubiquitous CAVs, signal-free junctions have been proposed in which vehicles negotiate traffic lights through cooperative communication with one another. The study of reserves is a vital subfield of philosophy.

This paper is organized as follows. Section II explains the concept idea of intersection management concept . Section III discuss the problems have arises. Section IV explains more about the approaches that have taken. Section V explains the implementation that have been done and provides simulation for the model intersection management.

## II. Intersection Manager Concept

An intersection manager and vehicle agents make up the reservation-based system. Each incoming vehicle agent requests a specific time slot from the junction manager so that it can proceed through the intersection in the manner of its choosing. Due to concerns with previous bookings, the request must be approved or declined by the intersection manager. Simple but effective methods, like the "first come, first served" (FCFS) strategy [6], the "auction strategy" [7],

and the ”platooning BATCH approach” [8], determine the order in which vehicles receive maintenance.

Furthermore, because of unequal traffic on main and smaller roads, the regularly used traffic lights may result in long time delay at an intersection and, as a result, causing traffic jams. Batch processing of reservation requests was presented as an improvement over the FCFS mechanism for enforcing liveness criteria [8]. Despite significant efforts to manage CAVs at intersections, the advantages of reservation-based control over conventional signal control have yet to be proven, particularly at peak demand levels and have not been thoroughly explored. However, comparisons were drawn using concrete numerical examples. Changes to three numerical parameters can have vastly different effects on the final result. Therefore, it would be more persuasive to compare these two control systems in a logical manner, such as by employing queuing theory to analyze latency and throughput at signalized junctions. Furthermore, due to the rule-based nature of reservation-based control, vehicle trajectories are not optimized, so system optimality is not guaranteed.

Constructing a constrained nonlinear optimization model with a focus on safety to reduce trajectory overlap has received a lot less attention than the shift from reservation-based to optimization-based approaches. There was a focus on achieving the best possible acceleration and deceleration from the vehicle. A hybrid of the active set method and an optimizer was used to solve this model. However, due to this intractable objective and the complex nonlinear model, optimal efficiency cannot be achieved. Researchers [9] found that there are three distinct areas for different types of trajectories at an intersection. The team then presented an interior-point-solved constrained nonlinear optimization model for minimizing vehicle delay. Optimization was performed only for newly loaded vehicles; all previously calculated trajectories were saved as backups. [10] used the cell transmission model to develop a lane-based traffic flow model and offered a linear programming strategy for controlling traffic lights at intersections without human intervention. This study fills these gaps in the literature by using queuing theory to assess the efficacy of reservation-based control and by proposing a practical model for CAV management at isolated crossings. The best maintenance schedule for this problem is then determined using an optimization model.

## III. Problem Statement

The majority of today’s junctions are regulated by traffic lights, stop signs, or roundabouts. Even when people follow these guidelines, however, intersections cause a lot of traffic congestion and accidents. Intersections account for around one-quarter of all accidents and one-third of all fatal accidents, although taking up a small percentage of the route. Autonomous vehicles appear to be well on their way to becoming a reality. They will have to follow the same intersection control procedures that were established for people at first, but because most cars on the road are autonomous, this study is examining whether we can do better by using an AI-based coordination mechanism. We devised a new multi-agent intersection control technique specifically for autonomous cars in this study. The goal reservation protocol has the ability to drastically reduce congestion and thus fuel usage by utilizing much more of the intersection at any given moment than traffic signals and stop signs. By removing the requirement for a great deal of deceleration and acceleration, hazardous emissions can be significantly minimized.

## IV. Approach

In this section, we will be explaining about the approach that we have taken in order to solve the problems that have been stated beforehand. As a consequence, we have created a scenario to simulate the situation in where all of the components that we have decided to use, communicating and interacting with each other to manifest our system as a whole. A few diagrams such as activity diagram, sequence diagram and use case diagram are included to have a better view on understanding the idea behind the solution that we have implemented into our project.

### A. Diagrams

We will start with two activity diagrams which are created to explain the flow of the activities from the scenario happened in our system. One from the side of autonomous vehicles which are drove by the driver and the other one denotes the server hub side. Next, we will go to the use case diagram, followed by the sequence diagram and lastly, the sequence diagram.

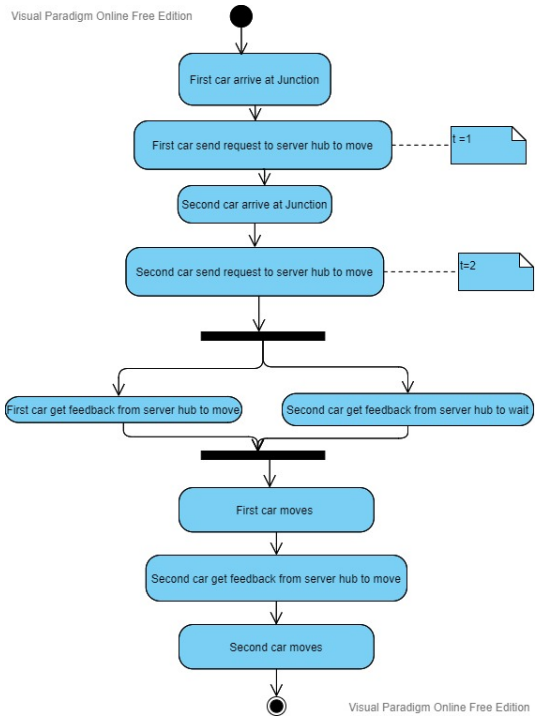


Fig. 1. Activity diagram for autonomous vehicles.

The scenario based on “Fig. 1” started from the initial node going to when two cars arrive at the 4-way junctions at the

same time. As they arrive the junctions, they will immediately communicate with the server hub. This is done by notifying and sending the signal to the server hub on the direction that they desired to go. The server hub will then decide which car will go first by assigning the cars into a queue. The first car en-queued in the schedule or queue will be the first one to get the priority to move first just as how the First In First Out (FIFO) algorithm works. Then, the server will send the signal back to the cars as feedback on the decision and instruction that has been made on who is going to get to move first and who should be waiting in the queue. As both cars received the signals from the server, they will then move according to the message acquired from the server. The next activity diagram will be explained below.

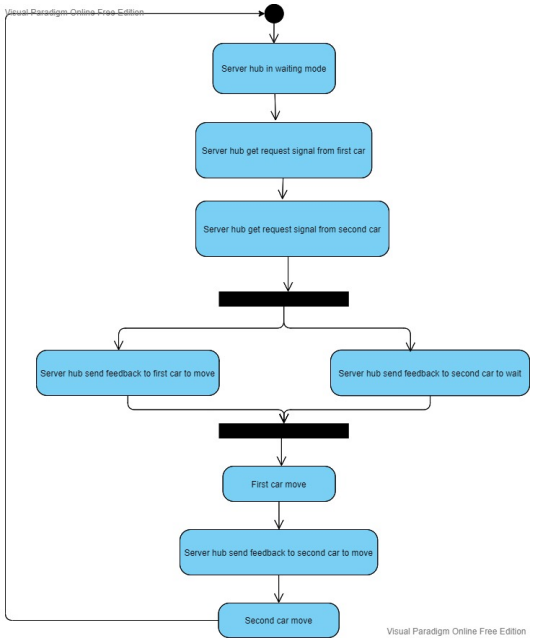


Fig. 2. Activity diagram for server hub.

The activity diagram as shown "Fig. 2" denotes the point of view from the server and how it will react accordingly to the situation. Since we cannot have two cars in the same number sequence of queue, the server will assign any of cars that enter the junction into the list of queues randomly as for example, first and second queue, even though both cars arrive at the same time. This will be synthesized by the server as if the cars arrive one after another and create a sequence of first car requesting desired direction to the server hub at t = 1, followed by the second car at t = 2. As the server received the signals from both cars, it will generate a schedule based on the priority algorithm of FIFO. Then, the server will send the feedback signals back to the cars saying that the first car in the sequence will get the first priority to move while the second car will have to wait for a while before getting to move into its desired direction, just after the first car successfully moving towards its desired direction.

To attain the aim of a totally autonomous driving car, the vehicle must have a thorough understanding of its surroundings. One aspect of the puzzle is onboard sensors, radars, and cameras. To provide a global perspective, additional data from surrounding vehicle sensors, intelligent traffic management systems (TMS), and pedestrians were collected. It is also necessary to establish a global traffic-information network. Getting this information across V2X is responsible for delivering air to all entities. Taking a broad picture of the vehicle's own circumstances It is conceivable to drive autonomously in the presence of these factors and nearby traffic. Information that is required is based on onboard sensors and V2X data, which are then analyzed by a sensor-fusion algorithm.

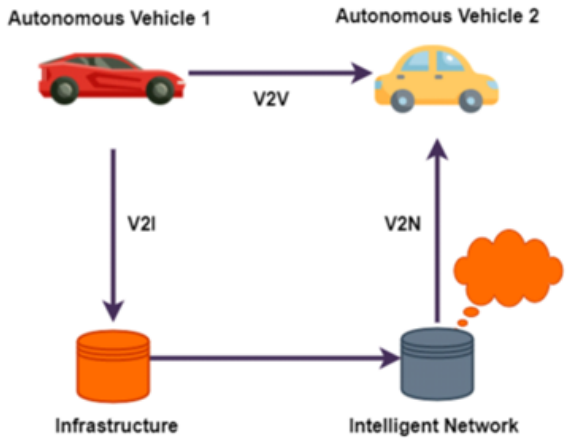


Fig. 3. Autonomous Cross Road Infrastructure Scenario

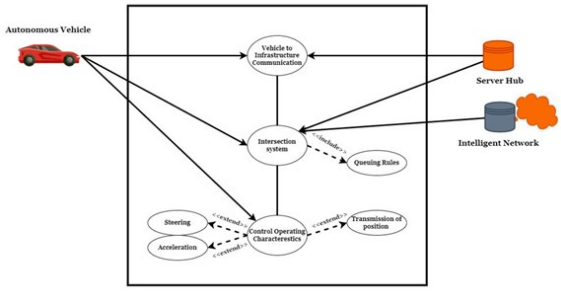


Fig. 4. Autonomous Cross Road Infrastructure Use Case Diagram

- Shared information between automobiles and roadside units (RSU) or intelligent roadside stations is referred to as vehicle server or infrastructure communication. By delivering information or directives to vehicles or receiving appropriate sensor data from them, the roadside infrastructure dynamically regulates traffic in real-time.
- Communication is in charge of broadcasting global information to all cars or streaming data to high-bandwidth-demanding apps. To put it another way, V2N refers to a vehicle's non-real-time capable connection to the Internet or cloud computing services. Traditional TMS are ill-equipped to deal with tomorrow's major issues. When it

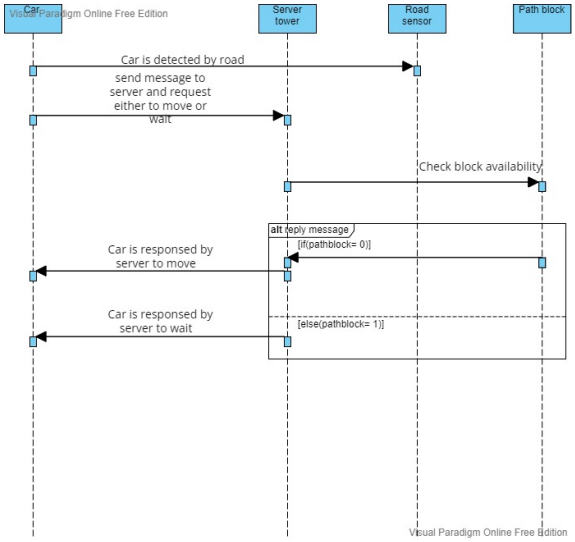


Fig. 5. Sequence diagram.

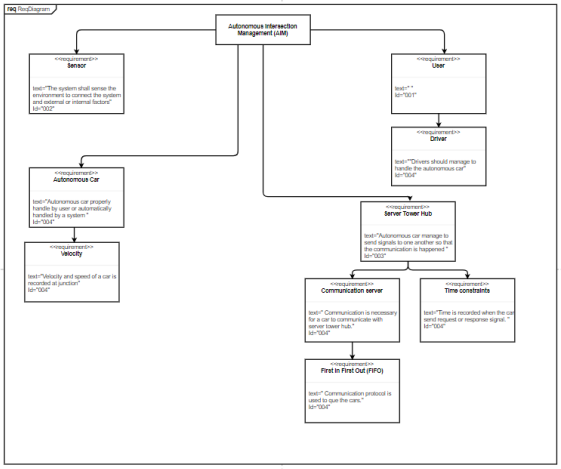


Fig. 6. Requirement diagram.

comes to road crossings, this is especially true. Changing the color of traffic signals from green to red on a regular basis prevents them from adapting flexibly to changing traffic loads. As a result, the proposed use case is built on a variety of smart city ideas and use cases that are backed by a V2I system.

"Fig. 5" illustrates the sequence of messages between the car, sensors placed on the roads, server tower and the path block in the interaction regarding the scenario that we have made clear earlier. The objects stated before are represented by lifelines, and the messages that they exchange over time during the interaction. This sequence diagram starts with the cars being detected by the sensors on the road to acknowledge that the cars are arriving at the junction. Then the cars send signals to the server tower requesting to move to its desired directions.

Subsequently, the server will check the availability of the path block to determine whether the path is available and most importantly safe for the cars to cross the junction. The request from the car to head to its desired direction will either be rejected or granted depending on the data obtained from path block. From the sequence diagram, we can observe that if the signal sent to the server tower is equal to '0', this denotes that the path is clear and the car is allowed to move. Meanwhile, if the signal received by the server tower from the path block is equal to '1', this means that path is not clear or still in occupied. In this case, the server tower will send signal for the car to wait until the path is available again in order for the car to head to its desired direction or destination.

Next, we dive deep into the requirement diagram see how the objects in our system are actually related with each other. This diagram shows the requirements that we need in this project such as the object or components included in our project with a little description on the functionalities and purpose. As we can see from the requirement diagram shown in "Fig. 6" , there are four main parts or requirements in the system of our projects which are the sensors, server tower hub, autonomous cars and the user itself. The user indicates the driver and they will manage to handle their own autonomous car. Server tower hub contains the communication server as a medium for the car and the tower to communicate to each other by exchanging data through the signals sent to each other and time constraints which will be utilised during the process of signal request and response. First In First Out (FIFO) algorithm is used in the communication server as to indicate which car that arrives first will get into the queue first and will get the first priority to send the request signal and finally move to its desired direction the earliest.

### *B. UPPAAL*

In this section, we will prove our model using one of model checker named UPPAAL. UPPAAL is a tool environment for modeling, validating, and verifying real-time systems modeled as networks of timed automata with data types. We have designed 3 state chart models that are dependent with each other These models will show how our system works generally.

The first state chart model is "Car". Firstly, in this model we showed that the scenario of the car arriving at the junction. It will later start to communicate and giving signal to the server. Based on "Fig. 7" we can see the car trigger the message to synchronize with the "Server" state model. The car will be waiting for the signal from the server to confirm whether it should move or wait. If the path of the junction is clear, the car has permission to continue to its desired direction or else the car must wait until the path is cleared.

The next model is "Server" as depicted in "Fig. 8". Initially the server will be idle if there is no request or response from cars at the junction. As mentioned above, the car will be sending the signal to the server to give information that it ready to enter the queue. Here the server will sort the car in the queue based on First Come First Serve (FCFS). Before the

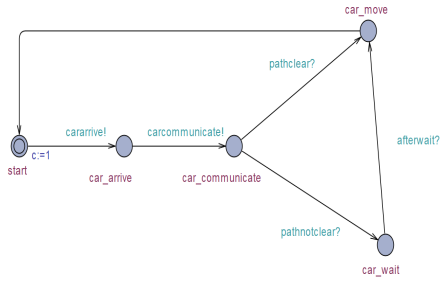


Fig. 7. ”Car” state chart model.

server return the signal to the car, it must do a crucial task which is checking path condition. This is to ensure that the cars are safely move at the junction and to avoid any intersection or accident. The server will later return to idle if all the tasked has been handled.

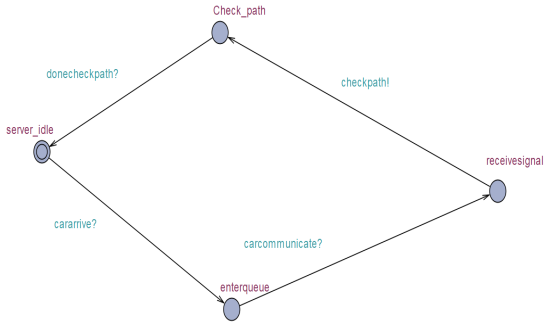


Fig. 8. ”Server” state chart model.

The last model called “Path” as illustrated in “Fig. 9” . This model shows the situation when the server starts to check path. There are only two states at the junction whether it is available or not available. When the path is clear, the server will grant permission to the car to move to the requested direction. Otherwise, the car needs to wait at certation time until the path is clear as shown as Figure below.

The results of the simulation can be referred in both “Fig. 10” and “Fig. 11”. “Fig. 10” shows when the car has permission to move at the junction and “Fig. 11” shows that the car need to wait for instances based on the queue and to clear the bath. All the state charts can return to its origin without having any issue or deadlock.

## V. Implementation

This section will go over the implementation of the traffic system model using VHDL and RTOS in more details. The code example of the system is also provided but for further information about the coding, can be referred to reference [11]. After that, it will demonstrate how the traffic system operates in real time.

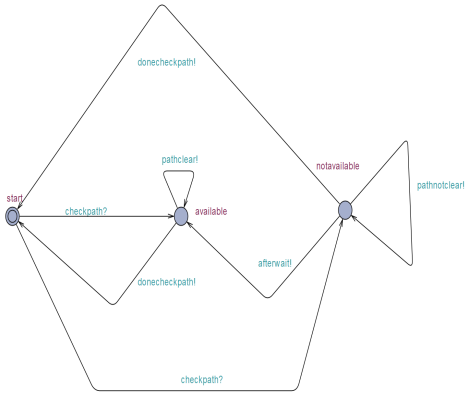


Fig. 9. ”Path” state chart model.

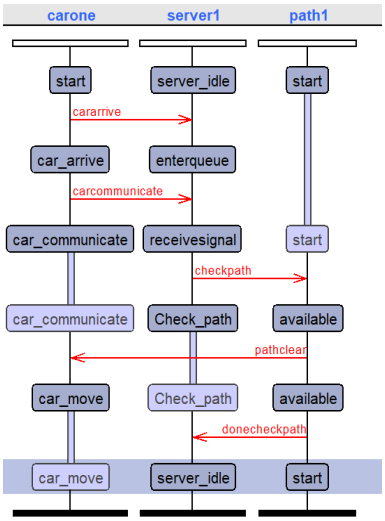


Fig. 10. Results of UPPAAL simulation.

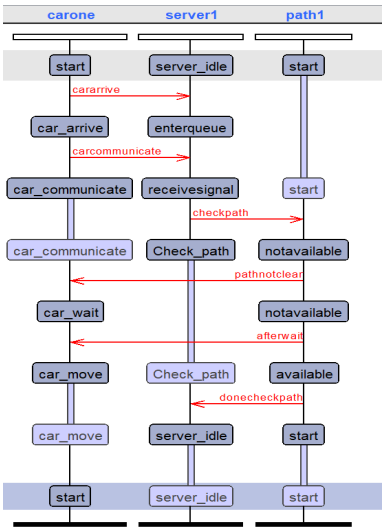


Fig. 11. Results of UPPAAL simulation.

### A. Real Time Operating System(RTOS)

Real Time Operating System (RTOS) will be used in our traffic system implementation. As previously discussed, the scenario that have created will include dummy cars such as car1, car2, and so on, as shown in “Fig. 12”. The server will place these cars directly in the queue upon their arrival, as shown in “Fig. 13”. In this scenario, the car that arrives first has a higher priority than the other cars when it comes to moving at the intersection. As a result, the cars that arrive first will be placed first in the queue.

```
//create dummy cars
int A=1;
int B=2;
int C=3;
int D=4;
int F=5;
int G=6;
int H=7;
int I=8;
```

Fig. 12. Dummy cars are created

```
//create queue
Car_Queue = xQueueCreate( 4, sizeof( unsigned int ) );
//put cars in the queue
xQueueSendToBack(  Car_Queue, &A, 0);
xQueueSendToBack(  Car_Queue, &B, 0);
xQueueSendToBack(  Car_Queue, &C, 0);
xQueueSendToBack(  Car_Queue, &D, 0);
```

Fig. 13. Cars will put in a queue system

In addition, as shown in “Fig. 14”, three tasks are being created: car communicate, car move, and car wait. Firstly, the task car communication will be created and assigned the highest priority ”2”, among other tasks. However, the tasks car move and car wait will be assigned the same priority, which is priority ”1”. This means that the task car communication will be executed first, before the other tasks with priority ”1”.

As shown in “Fig. 15”, the server will first determine whether or not the car has arrived at the junction for task car communication. If the car arrives, the communication between the car and the server will begin. Communication between the car and the server is required so that the server knows the desired direction of the cars. This scenario has only four directions: north, south, east, and west. After that, if the queue car is already full, no car will be added to the queue. If the

```
//CreatingTask
//Task move is created with priority '1'
xTaskCreate(car_move, "Car move", 1000, NULL, 1, &TaskHandle_1);
//Task wait is created with priority '1'
xTaskCreate(car_wait, "Car wait", 1000, NULL, 1, &TaskHandle_2);
//Task communication of the car is created with priority '2'
xTaskCreate(car_communicate, "Car communicate", 1000, NULL, 2, &TaskHandle_3);
```

Fig. 14. Communicate, Wait and Move tasks are created

queue car is empty, the server will place the new car in the vacant queue space. The server will add the car to the move list once it has finished communicating with the server. The car moves according to the move list as shown in “Fig. 16” , and only one car can move at a time. As a result, the accident can be avoided. This task communication will be suspended once the car and the server have finished communicate.

```
//refill car when queue is empty
  if(uxQueueMessagesWaiting(Car_Queue) == 0 &&  change == 0)
     {    Serial.println("new car arrive");
          xQueueSendToBack(  Car_Queue, &F, 0);
          xQueueSendToBack(  Car_Queue, &G, 0);
          xQueueSendToBack(  Car_Queue, &H, 0);
          xQueueSendToBack(  Car_Queue, &I, 0);
          change =1;
     }
//refill new car when queue is empty
  if(uxQueueMessagesWaiting(Car_Queue) == 0 &&   change == 1)
      {   Serial.println("new car arrive");
          xQueueSendToBack(  Car_Queue, &A, 0);
          xQueueSendToBack(  Car_Queue, &B, 0);
          xQueueSendToBack(  Car_Queue, &C, 0);
          xQueueSendToBack(  Car_Queue, &D, 0);
          change =0;
      }
```

Fig. 15. System check either the new car has arrived or not to be put in the queue

```
if(uxQueueMessagesWaiting(Car_Queue) !=0)
    {for(int o=0;o<4;o++)
        {   xQueueReceive(Car_Queue,&x,0);
            move_list[o]=x;
             Serial.print("car ");
             Serial.print(move_list[o] );
             Serial.println(" request");
             Serial.print("to ");
             Serial.print(direction_car[o] );
             Serial.println(" ");
        }
             Serial.println("finish request ");
    }
 vTaskSuspend(TaskHandle_3);
```

Fig. 16. System put the cars in move list

Following the suspension of the task communicate, it is now the turn of the tasks car move and car wait to run concurrently, as shown in “Fig. 17” and “Fig. 18”. The car’s movement will now begin in task car move. The car that is placed first on the

move list will be moved first. The other cars will have to wait. Furthermore, it should be noted that the car will only move if the move signal is received. If they do not receive the move signal, they will remain in the waiting mode.

```
if(move_list[0]!=0)
      { Serial.print("car ");
        Serial.print(move_list[0] );
        Serial.println(" move");
        signal_wait=1;
      }
  vTaskSuspend(TaskHandle_1);
  if(move_list[1]!=0)
      { Serial.print("car ");
        Serial.print(move_list[1] );
        Serial.println(" move");
      }
   if(move_list[2]!=0)
      { Serial.print("car ");
        Serial.print(move_list[2] );
        Serial.println(" move");
      }
   if(move_list[3]!=0)
      { Serial.print("car ");
        Serial.print(move_list[3] );
        Serial.println(" move");
      }
```

Fig. 17. Moving car based on move list.

```
 if(move_list[1]!=0 && signal_wait==1)
     {
       Serial.print("car ");
       Serial.print(move_list[1] );
       Serial.println(" wait");
       Serial.print("car ");
       Serial.print(move_list[2] );
       Serial.println(" wait");
       Serial.print("car ");
       Serial.print(move_list[3] );
       Serial.println(" wait");
       signal_wait=0;
     }
  vTaskResume(TaskHandle_1);
```

Fig. 18. Waiting car after receiving the wait signal.

This is how example model of the system works in RTOS. In the next part, VHDL model will be explained so that, the system can be realised on hardware.

### *B. VHDL*

The traffic system will be implemented in VHDL in this section. The primary goal of this implementation is to demonstrate how communication works in real-time systems between the car and the server. Before the implementation in VHDL, it should be noted that a Finite State Machine (FSM) must be created, as shown in “Fig. 19”. In this FSM, an entity is created called ”Myserver” with three states: idle, makeQueue, and getSignal. Table 1 describes the input and output of the ”Myserver.” The coding for process input and output is shown in “Fig. 20” and “Fig. 21”. Following that, the simulation will begin to test the input adn output logic of the ”Myserver” system. This can be illustrated in “Fig. 22”.

TABLE I

| State | Input | Output |
|---|---|---|
| Idle | • car=”00000001” indicates that no car has arrived | • signal car= ”00000001” denotes no queue |
| | • car=”00000010” denotes car arrival | • signal car = ”00000010” denotes enter queue. |
| makeQueue | • car=”00000011” represents car communication | signal car= ”0000100” indicate that the server has returned to idle mode. |
| getSignal | • car=”00000011” symbolizes car communication | • signal car= ”00000011” denotes a get signal |
| | • car=”00000011” indicates that the car is not communicating | • signal car= ”00000011” indicates that the server has returned to idle mode |

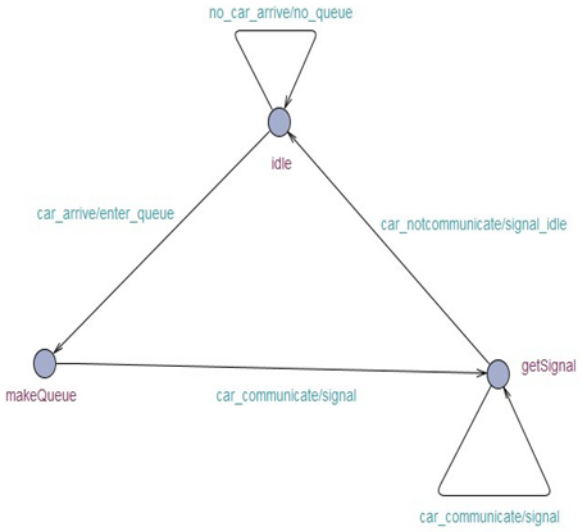


Fig. 19. Finite State Machine (FSM) of communication system named Myserver.

## VI. CONCLUSION

In the nutshell, this paper explained about our project which is Autonomous Intersection Management (AIM). In this project, a simple design model of AIM has fully discussed.

```
input_comb:process(current_state,car)

begin
        case current_state is
                when idle => if car="00000001" then
                                next_state<= idle;
                                elsif(car="00000010") then
                                next_state<=make_queue;
                                end if;

                when make_queue=> if car="00000011" then
                                next_state<=get_signal;
                                --else next_state <= make_queue;
                                end if;

                when get_signal=> if car="00000100" then
                                        next_state <= idle;
                                        elsif(car="00000011") then
                                        next_state <= get_signal;
                                        end if;
                end case;
                end process;
```

Fig. 20. Input coding of "Myserver" system.

```
output_comb:process(current_state, car)

begin
    case current_state is

            when idle => if car = "00000001" then
                        signal_car <= "00000001";
                        elsif (car = "00000010")  then signal_car <= "00000010";
                        end if;

            when make_queue => if car= "00000011" then
                               signal_car <= "00000011";
                               else signal_car <= "00000100";
                               end if;

            when get_signal => if car= "00000100" then
                               signal_car <= "00000100";
                               elsif(car= "00000011") then signal_car <= "00000100";
                               end if;
end case;
```

Fig. 21. Output coding of "Myserver" system.

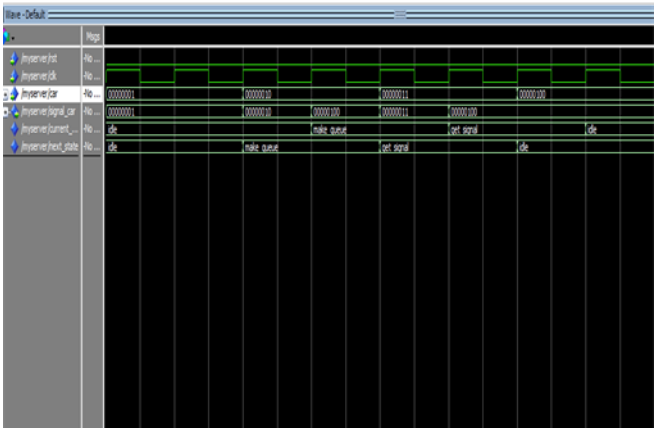

Fig. 22. Result of "Myserver" system.

Since there might no traffic light in the future, it is benefit to use it in traffic management such as to avoid accident, heavy traffic, etc. This paper also provides diagrams that explained the structure of our system. The simulation by using UPPAAL and RTOS should be able to help the reader to understand the idea of the system. Although we are well aware that our project is lacking in some parts and there are still much more improvements that can be implemented in tackling the problems in our project, but we hope that it can be used as the first step towards a solution in today's problems that we faced on the intersection for autonomous car.

## VII. Acknowledgement

We are eternally grateful to Prof. Dr. Henkler, Stefan, and Prof. Achim Rettberg whose giving inspiration, encouragement, guidance, and support from the beginning to the conclusion helped us to gain awareness and open our eyes to the importance of this subject module. We also like to express my gratitude, respect, regard, and benefits to everybody who helped me in any manner during the task's execution. we did everything in our power to obtain a better understanding and share it in this paper, and we hope the reader can benefit from it, particularly in this area.

## VIII. Affidavit

This paper has been written independently. We have not used any sources other than those indicated. All statements taken from other sources in wording or sense are cited. Furthermore, we assure that this paper has not been part of a course or examination in the same or a similar version.

## References

[1] Ahn, H. and Del Vecchio, D., 2016, April. Semi-autonomous intersection collision avoidance through job-shop scheduling. In Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control (pp. 185-194).

[2] Azimi, R., Bhatia, G., Rajkumar, R.R. and Mudalige, P., 2014, April. STIP: Spatio-temporal intersection protocols for autonomous vehicles. In 2014 ACM/IEEE international conference on cyber-physical systems (ICCPS) (pp. 1-12). IEEE

[3] Bahram, M., 2017. Interactive maneuver prediction and planning for highly automated driving functions (Doctoral dissertation, Technische Universität München).

[4] Behrisch, M., Bieker, L., Erdmann, J. and Krajzewicz, D., 2011. SUMO–simulation of urban mobility: an overview. In Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. ThinkMind.

[5] Campos, G.R., Falcone, P., Wymeersch, H., Hult, R. and Sjöberg, J., 2014, December. Cooperative receding horizon conflict resolution at traffic intersections. In 53rd IEEE Conference on Decision and Control (pp. 2932-2937). IEEE.

[6] Guney, M.A. and Raptis, I.A., 2020. Scheduling-based optimization for motion coordination of autonomous vehicles at multilane intersections. Journal of Robotics, 2020.

[7] Hafner, M.R., Cunningham, D., Caminiti, L. and Del Vecchio, D., 2013. Cooperative collision avoidance at intersections: Algorithms and experiments. IEEE Transactions on Intelligent Transportation Systems, 14(3), pp.1162-1175.

[8] Jang, K., Vinitsky, E., Chalaki, B., Remer, B., Beaver, L., Malikopoulos, A.A. and Bayen, A., 2019, April. Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles. In Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems (pp. 291-300).

[9] Kamal, M.A.S., Taguchi, S. and Yoshimura, T., 2016. Efficient driving on multilane roads under a connected vehicle environment. IEEE Transactions on Intelligent Transportation Systems, 17(9), pp.2541-2551.
[10] Keskin, M.F., Peng, B., Kulcsar, B. and Wymeersch, H., 2020. Altruistic control of connected automated vehicles in mixed-autonomy multi-lane highway traffic. IFAC-PapersOnLine, 53(2), pp.14966-14971.
[11] https://github.com/Team-Lamentis/Team-Lamentis-/tree/main/Implementation