# Cross Traffic Management For Autonomous Vehicles


Abdullah Al Forkan
*Electronic Engineering*
*Hochshule Hamm Lippstadt*
Hamm, Germany
Abdullah-al.forkan@stud.hshl.de

Md Akram
*Electronic Engineering*
*Hochshule Hamm Lippstadt*
Hamm, Germany
md.akram@stud.hshl.de

Ali Hisham Omer Ahmed
*Electronics Engineering*
*Hochshule Hamm Lippstadt*
Hamm, Germany
ali.hisham-omer-ahmed@stud.hshl.de

Dapsara Kapuge
*Electronic Engineering*
*Hochschule Hamm Lippstadt*
Lippstadt, Germany
dapsara.kapuge@stud.hshl.de



***Abstract*—At the age of modern technology autonomous vehicles technology are developing for ensuring flexibility on travel. Uses of autonomous car can save our time and money. I will definitely change the modern car industry and our daily journey. In this paper we did work about cross traffic management for autonomous cars. To ensure a safe and secure journey of autonomous cars at cross section area here we discuss and verify many approaches such as UPPAAL, FreeRTOS, and VHDL simulation. UPPAAL was utilized for modelling and verifying real-time systems to ensure correct behavior under various traffic scenarios. FreeRTOS facilitated real-time task management, enabling effective coordination among multiple autonomous vehicles. VHDL was used for the precise hardware implementation of control systems, validating their performance in realistic conditions. We also solve many exceptional scenarios such as right, forward, left and emergency car to ensure a safe and secure crossing of autonomous cars in intersection areas.**



***Index Terms*—Autonomous car, secure journey, cross section area**


## I. Introduction

Conventional methods of traffic management usually lead to congestion, delays, and irritation among motorists. Such challenges are overcome by our innovative cross-traffic system that integrates the latest technologies and smart algorithms. This keeps the traffic flowing continuously without interruptions, coordinates traffic lights, and responds to real-time traffic conditions, likely to ensure road safety, reduce travel time, and lower carbon emissions. It also makes the drive much safer for drivers at large by giving them a smooth experience at junctions.

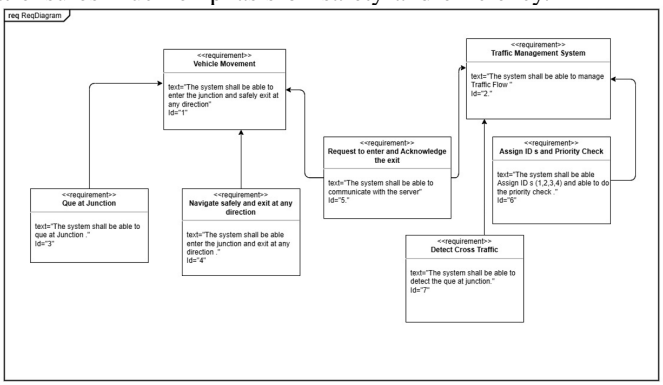
## II. Solution

Model the intersection as a queuing system where vehicles are customers and the intersection is the server. Implement a priority-based system to manage vehicle passage. Choose appropriate queue models and Integrate adaptive traffic light control algorithms to ensure efficient traffic management and reduced delays for autonomous vehicles.

## III. SysML and UML Implementation

### A. Requirement Diagram

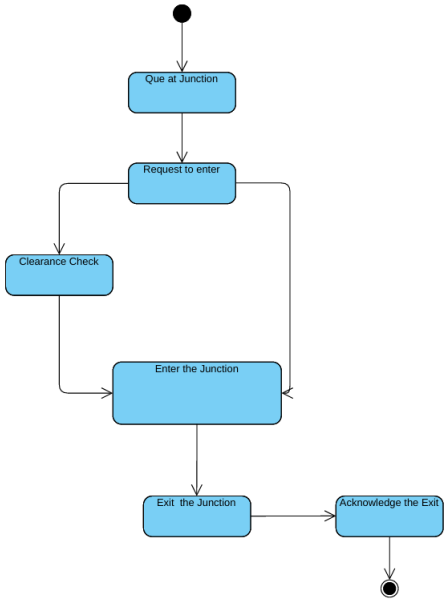
This requirement diagram shows the way in which a traffic management system is supposed to work in regulating the traffic flow through a junction. The basic requirement is the safe reaching and leaving of vehicles to and from an intersection on all the sides. Besides more specific tasks such as assigning vehicle priority IDs, priority checking, and detection of cross traffic for preventing congestion, the responsibility of managing the overall traffic flow lies with the Traffic Management System. In managing both the requests and exits of vehicles, the system needs to communicate with and interact with the server. What this means is that every requirement is important to ensure an optimized traffic management strategy that ensures much emphasis on safety and efficiency.




### B. State Machine Diagram

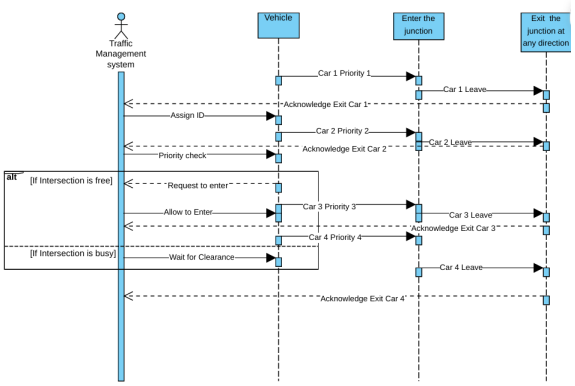
This diagram shows how a car goes through a junction while being monitored by the traffic management system,

ensuring avoidance of accidents and orderly traffic. The process starts as cars arrive and que at the intersection and then wait for their turn. vehicle sends their request to the Traffic Management System for authorization to pass the intersection. Then the system checks for clearance that the intersection is clear and safe to let the car pass. If an intersection is clear, the car can pass otherwise it waits until intersection is free. The car enters the intersection drives through it and exits in the desired direction after the clearance check. The process for that car ends when the Traffic Management System acknowledges the vehicle departure after the exit. This will ensure an effective and safe flow of traffic.



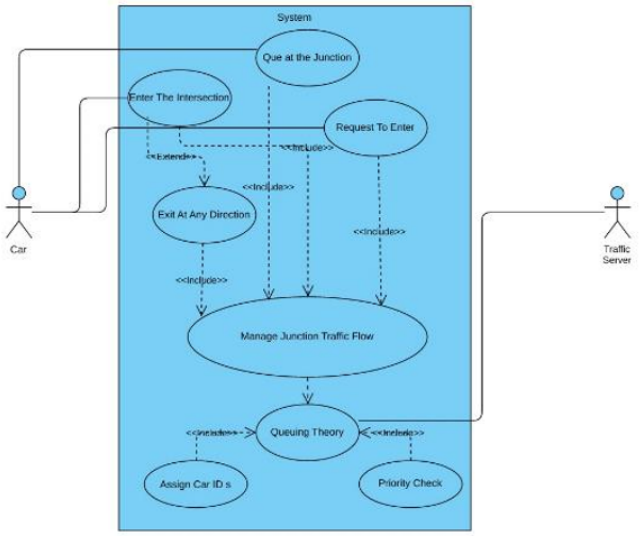

### C. Sequence Diagram

This sequence diagram shows the prioritization of vehicles by the Traffic Management System, as represented by Car 1, Car 2, Car 3, Car 4, Car IDs, and acknowledging exit techniques to manage vehicle traffic at a junction. When a car queues at an intersection, it gets a priority level and requests to enter. The system will then check the traffic at the junction. In the case of no traffic at the junction, cars are cleared on priority basis and their exits are then acknowledged by the system. If the junction is busy, cars will await clearance before approaching the junction. Through this ordered technique, by giving priority to vehicle entry and by observing efficiency and safety within the junction, it is ensured that the traffic flow goes orderly.

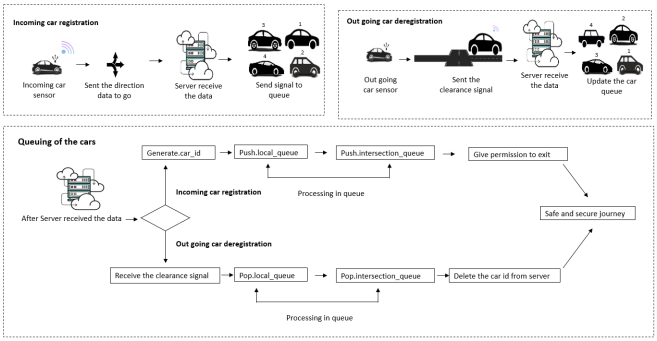



### D. User Case Diagram

In this User Case Diagram CAR will act as an ACTOR and the Traffic Server As the System. Following is the use case diagram that captures the interaction between cars and the traffic server, thus showing how the traffic management system controls vehicle traffic at a junction. The main purpose of this system is to control the flow of traffic through an interlocking network of operations. At the intersection, vehicles queue up and request clearance to pass. It utilizes queuing theory for the generation of unique IDs, maintenance of priorities of vehicles, and priority checking in case a request is forwarded. Based on priorities and flow, vehicles are granted entry into the intersection.



### E. Activity Diagram

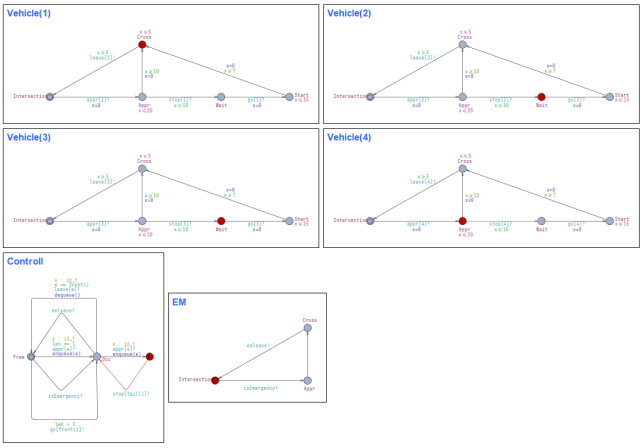



At the time whenever any car will be come to the range of intersection area they are able to exchange the direction data to the Server. So, when the server receives the direction from the car it will be generate random car id and check the availability in the intersection area according the current condition in queue update the queue. If the intersection area is not available then it will be sent the data to the car for waiting and update the car in queue. Once any car will be successfully cross the intersection are this time the car sent the date to the server that this car successfully cross the area and then the server delete the car form the queue and update the queue.

## IV. Simulation

### A. UPPAAL

In this autonomous traffic control system, 'UPAAL 5.0.0' is used to simulate the behaviors of the vehicles and the server. Three templates have been created to demonstrate the solution.




*1) Vehicle:* The vehicle template represents each vehicle state in the four-way cross-sectional road. It consists of five states Intersection, Appr, Wait, Start, and Cross. Each state denotes the current position of a vehicle. Though a clock(x) is used to monitor the transitions, this is mainly a message-based system. The communication happens between the vehicles and the server. Every time a vehicle approaches or crosses, it sends a message to the server that it is approaching (appr[id]!) or leaving (leave[id]!). In the approach or wait state a vehicle also receives a message from the server to stop or go.

*2) Emergency Vehicle:* An emergency vehicle template is created for the vehicles that follow the priority inheritance protocol. It contains three simple states named Intersection, Appr, and Cross. Emergency vehicles cross immediately, sending an 'isEmergency' signal while approaching and an 'emLeave' signal when crossed.

*3) Control Server:* The control template controls the complete functionalities of the system by creating a queue and communication system. It has three states Free, Occ, and committed state. It gets signals from the vehicles while approaching, leaving, and emergency and sends messages to the vehicles to stop and go.

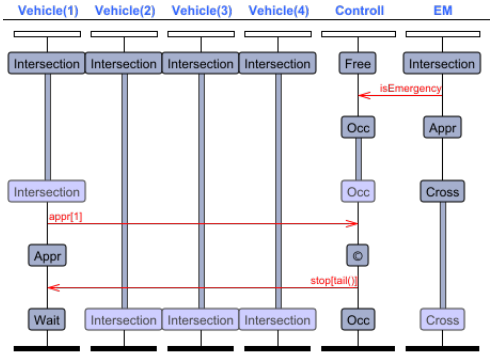
Name: Controll Parameters:

```
// Put an element at the end of the queue
void enqueue(id_t element)
{
        list[len++] = element;
}

// Remove the front element of the queue
void dequeue()
{
        int i = 1;
        len -= 1;
        while (i < len)
        {
                list[i] = list[i + 1];
                i++;
        }
        list[i] = 1;
}
```

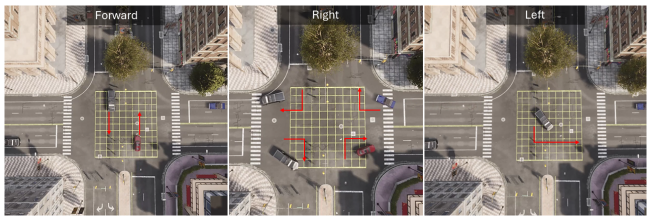
This UPPAAL code shows the enqueue and dequeue process of the control server. The Enqueue function adds any new vehicle to the list as soon as it approaches and the Dequeue function decreases 1 as soon as a vehicle leaves the road.

*4) Symbolic Simulation:* If a vehicle is going to the 'Appr' state from the initial intersection, it sends an 'Appr[id]' message to the server and the server updates its state to occupied (Occ). For the first vehicle, it can directly go to the 'Cross' state and send a 'leave[id]' message to the server, which updates the occupied state to free again. As long as the server is not occupied, any vehicle can cross without interruption and the loop continues. If another vehicle approaches when the first vehicle is in approaching or cross state, it gets a 'stop[id]' signal from the server and goes to the wait state until it receives a 'go[id]' signal. Similarly, any vehicle that approaches in this situation gets a stop signal and goes to the wait state. If the first vehicle is left, then the server becomes free again and the next front vehicle of the list receives a 'go[2]' signal to start and cross. Leaving the intersection dequeues the vehicle and creates space for the next vehicle.




This UPPAAL fig shows that if an emergency vehicle arrives at the intersection, it can approach immediately, making the server occupied, and any other car in the intersection receives a stop signal.

## B. CARLA - Maximum Parallelism

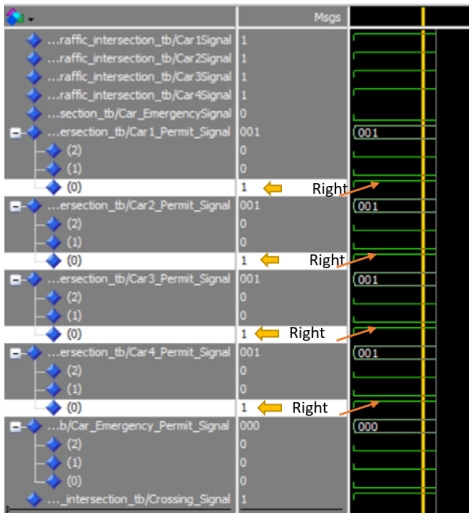



The maximum parallelism scenarios for different directions have been simulated through Carla autonomous simulator using Python. Carla is a tool for professional autonomous system testing based on Unreal engine software. It can be observed in the simulation figure that if two vehicles are going forward on the same road, such as the east and west roads can proceed together, and other vehicles are stopped. In the right scenario, if all the vehicles are going right, they can drive at the same time. Finally, for the left scenario, assuming the cross-sectional area has limited space, only one car can cross to the left at a time and other vehicles have to wait.

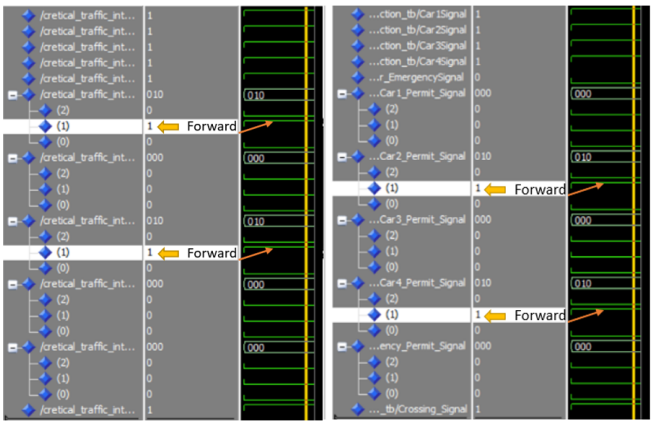
## C. VHDL

In VHDL we did work for giving the permission to the car according their direction data and the queue at this time. For that here we show some scenarios such as at the same time all cars in right direction, forward direction, left direction and a emergency car. In below we individually discuss these scenarios.

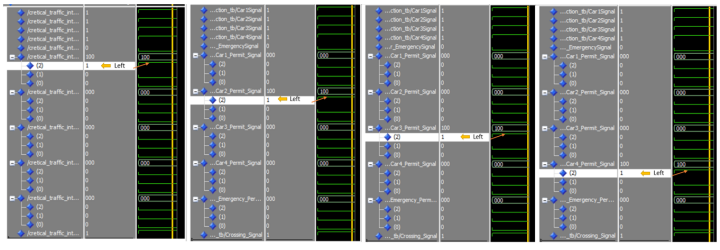
*1) Right direction:* If at the same time all car wants to go right direction then it will be no problem to give the permission to go all car at the same time. So, our serve will be giving the permission all car at the time to right direction.



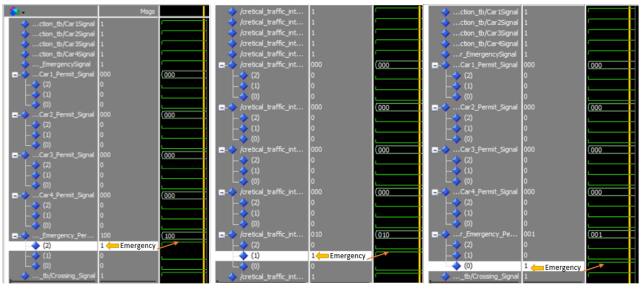

*2) Forward direction:* In the forward direction there is a possibility of accident if we give permission all car at same time so in this case to solve this problem we can give the permission firstly access two side cars to cross this area and other two side car should be wait at this time. For example, among four cars if all car wants to go forward direction this time we will give access to car 1 and car 3 to cross and this time car 2 and car 4 should be wait. After cross car 1 and car 3 then we will give the permission to car 2 and car 4.




*3) Left direction:* We can see this scenario is more complex then others because if at the same time we give access to the all cars in right direction then their will be high changes of accident. So ignore this problem we just give the access one car at a time.



*4) Emergency car:* If any emergency car such as ambulance come to the intersection area this time we are keep for others car because of the urgency and high speed of emergency car. Since at this time other car are stop this time intersection area is free and the emergency car can go any direction.



## V. FreeRTOS Simulation

### A. Introduction to FreeRTOS

FreeRTOS has proven to be one of the most robust and efficient real-time operating systems currently powering many industrial computers and small devices that we use daily.

What makes it so special is that it has been purposefully engineered to be very small in size, empowering it to support a wide variety of microcontrollers without being too demanding of resources [**?**].

### B. Project Overview

Our simulation is intended to control a simple intersection with 4 routes. It successfully manages cars approaching from four main directions (EAST, WEST, SOUTH, and NORTH).

We have mainly utilized FreeRTOS features such as tasks, queues, and prioritization to allow priority for emergency vehicles.

### C. Development Steps

*1) Research:* To utilize FreeRTOS properly, it's important to have a full understanding of the different aspects we must use. This includes the architecture we must use, how to configure FreeRTOS in accordance with it, how to leverage task creation and management functions, and becoming familiar with its interrupt handling feature.

*2) FreeRTOS Hardware-Specific Ports:* There is no universal version of FreeRTOS; as a result, it has many hardware-specific adaptations designed to utilize the full potential of the specific hardware it's custom-tailored to.

Many microcontrollers have varying architectures and memory management schemes, and a custom-made OS is the only possible path to utilize the full potential of the hardware.

In our case, we've used the POSIX adaptation of FreeRTOS. This adaptation allows us to develop our project without the use of any embedded hardware, requiring only a Linux-based development environment as both FreeRTOS and Linux comply with POSIX criteria.

### D. Implementation

*1) CMakeLists.txt:* The `CMakeLists.txt` is an essential part of our development setup, allowing us to effortlessly incorporate FreeRTOS kernel essential components into our system and automate the entire compilation process consistently.

It defines the compiler used, source files, header directory, and build rules.

*2) Configuring FreeRTOS:* To use FreeRTOS for our application, some configurations need to be defined and set. The key configurations for our build were:

- **Tickrate:** FreeRTOS uses ticks instead of seconds, so it is necessary to define the relationship between these two quantities to have precise timing for our task execution.
- **Minimum stack size:** This configuration is key to efficient memory utilization in our system. It is set in accordance with our smallest task available.
- **Total Heap Size:** Defines the amount of memory available for the entire system.
- **Tasks priorities:** Defines the number of priorities our application requires.
- **Kernel features:** Various FreeRTOS features can be selected or ignored through this setting.

*3) Main Application Code Breakdown:*

*a) Including Necessary Headers:* Our main code starts with including core FreeRTOS components, which contain functions for task management and queue handling.

*b) Defining Data Types and Constants:* These are used in the global scope to make our code more readable and maintainable. They represent our intersection traffic light states, colors, car directions, and car properties.

*c) Queue Handles:* Queues are a critical part of our code used for managing the flow of cars as they approach from different directions. They allow tasks to send and receive data.

In our case, they hold the cars, with four different directions each having a separate queue.

*d) Traffic Light States:* Traffic light states control the flow of cars through our intersection. Each direction has its own set of states indicating whether cars should wait, proceed, or prepare to stop completely.

These include red, yellow, green, and green for left turns, as the geometry of the intersection makes left turns more risky.

*e) Car Generation Task:* This task generates cars, places them in the queue, and simulates their arrival at the intersection. Emergency vehicles, with a higher priority than normal cars, spawn at a rate of 10

*f) Car Passing Task:* This task processes cars waiting to pass through the intersection. It checks the current traffic light states and allows cars to pass if the light is green for their direction and turn.

Emergency vehicles are given priority.

*g) Logging Task:* The logging task logs the state of the traffic lights, the status of each queue, and car information.

It's useful for monitoring and debugging by providing an overview of the flow of cars and the varying outputs of different tasks.

*h) Hook Functions:* Hook functions handle critical errors such as memory allocation failures and stack overflows.

The memory allocation failure hook outputs an error message and sends the system into an infinite loop to stop further task allocation.

The stack overflow hook prints an error message with the specific task that caused the issue and stops the system.

*i) Main Function:* The main function starts system operation by creating the queues and setting up tasks (traffic light task, car generation task, car passing task, logging task).

It then starts the FreeRTOS scheduler, which allocates system resources to tasks based on their priorities and traffic light states.

## VI. Conclusion

In order to provide safe and effective intersection crossing, we investigated cross traffic management for autonomous cars in this paper. Reliable traffic control and collision- mitigation approaches have been demonstrated through VHDL, FreeRTOS, and UPPAAL simulations. Real-time system modelling and verification occurred possible by UPPAAL, real-time task scheduling proved by FreeRTOS and accurate hardware implementation and validation was made possible by VHDL. By applying those approaches we can ensure the reliability and security of autonomous vehicle operations at intersections.