Autonoumous Traffic System

1st Zafirul Izzat Bin Mohamad Zaidi
Universty of Applied Sciences Hamm-Lippstadt, Lippstadt Hoffenheim
Lippstadt, Germany
zafirul-izzat-bin.mohamad-zaidi@stud.hshl.de

2nd Sheikh Muhammad Adib bin Sh Abu Bakar
Universty of Applied Sciences Hamm-Lippstadt, Lippstadt Hoffenheim
Lippstadt, Germany
sheikh-muhammad-adib.bin-sh-abu-bakar@stud.hshl.de

3rd Hadi Imran bin Md Radzi
Universty of Applied Sciences Hamm-Lippstadt, Lippstadt Hoffenheim
Lippstadt, Germany
hadi-imran.bin-md-radzi@stud.hshl.de

4th Ammar Haziq bin Mohd Halim
Universty of Applied Sciences Hamm-Lippstadt, Lippstadt Hoffenheim
Lippstadt, Germany
ammar-haziq.bin-mohd-halim@stud.hshl.de

***Abstract*—In recent years, the traffic has slowly increased due to the ever-increasing amount of cars on the road. There are several factors that affect the traffic but after careful analysis and consideration, we have found that one of the factors, which is the junction, can be further optimized to provide a smoother experience for road users. This is where the role of real time systems becomes apparently important, as we would like to introduce an autonomous solution. This paper describes in detail our documentation of our project from the beginning problem statement, our initial concept idea, our approach and finally to the implementation and conclusion.**

***Index Terms*—Autonomous traffic system**

I. Introduction

Every day, the number of automobiles on the road increases across the world. This tendency has a negative influence on our transportation system. In the worst-case scenario, a kilometre-long traffic gridlock might occur, usually during a major event. We might design a system to replace old-fashioned traffic lights that could handle traffic, especially for autonomous vehicles, in a more efficient manner, because the growing number is also being fuelled by a huge number of autonomous automobiles.

In our project, we developed an autonomous traffic system that could handle multiple autonomous cars in a junction, avoiding accidents and increasing the efficiency of traffic management in the junction.

In this article, we will go over our idea, an autonomous traffic system, in great depth. Before moving on to the main topic, the reader may come across a real-world situation with traffic systems in order to have complete knowledge of autonomous traffic systems. The next section will go through the concept and requirements of our system. Then comes the approach based on the requirement that we made. The approach is separated into two parts, which are architectural modelling and behavioural modelling. Furthermore, the reader will be exposed to the implementation section, in which a RTOS is used to meet our criterion for allowing the system to operate simultaneously. The reader will learn how our autonomous traffic system simulates in the next part. Finally, this article will come to a conclusion.

II. Concept

In this section, we will provide the user a short description on the concept of how our system idea came into fruition. Firstly, we have used the concept of resources for our system and that will be explained, A map of our junction is also provided to ease the reader for a better understanding on resources. Next we will explain about centralized and decentralized system and which option we chose for our system model. Lastly, we provide the final requirements diagram that is produced based on how we want to set up our system.

In our system, we have divided the junction into what is similar to a grid map view. Each square in the junction represents a resource that the car will use depending on which direction the car wants to go. This process is important to understand our system as we depend on the controller making critical decisions on which car is allowed to move through the junction depending on which resource is available. This is shown in “Fig. 1”.

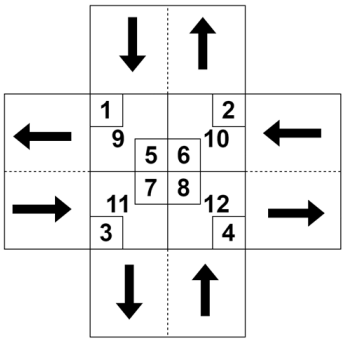

Fig. 1. Resource mapping.

There exists two different approaches for the traffic system that is possible to model our system based on said problem. The first is having a centralized system, where everything in our system revolves around one central controller or the second option is a decentralized system, where every car will communicate with each other and make decisions without the presence of a central controller. For our Autonomous Traffic System, we have chosen to utilize a centralized system architecture. This means that, in our system, there will be a traffic controller that will ultimately decide which car has the permission to pass through the junction first depending on which resource is available at the time.

The first step in solving or creating any given system would be to define the specific requirements for the system itself. This is important to save time in the future and to provide a clear direction for the implementation of the project. After careful discussion and analysis of what our system would be able to perform, we have come up with the requirements diagram as per "Fig. 2".

We have decided to name our system Autonomous Traffic System. Based on the figure, we would have two main actors that the system shall be able to operate with, the Traffic Controller and the car. We will go into detail for both, starting with the Traffic Controller. For the former, it shall be the main controller for the system and supports Car Handler and Manage Resource functions. Request Handler would then be able to accept a given car request by taking its direction and putting the car into a queue. It should also be noted that a communication protocol is required for the communication between controller and car.

Next, for Resource Manager requirement, it can access the car queue to select a car, check for the availability of the requested resources, lock the resources requested, give permission for the car to move, and unlock the resource used after the car has passed. For Car, the car will be able to give its location by utilizing a GPS system, send a request for permission to move to the controller and also give its direction on where the car wants to move in the junction.

In the successive sections, we will provide the reader with scenario-related diagrams such as Use case and Sequence diagrams to help visualize how the requirements are planned to be implemented in our Autonomous Traffic system. The diagrams mentioned will construct a sequential story for the application of said requirement diagram and how the requirements to act with each other and the actors that interact with the system.

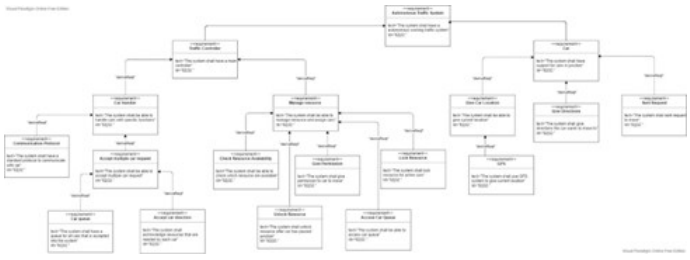

Fig. 2. Requirement diagram.

III. Approach

The model of the system, as well as the strategy for implementation, will be discussed in this section. The models may be divided into two categories: architectural modelling (discussed in the first subtopic) and behavioural modelling (detailed in the second subtopic).

A. Architectural modelling

To develop the system in a more effective and efficient way, we first plan the process. With both of the information that we gathered from said diagrams, we have a clearer vision and better understanding of the overall system. Our first approach is by modelling architectural modelling. This is important as we can now develop a more concrete system architecture with refined details. That is exactly what we did on this documentation for the reader to easily digest our overall system architecture. This is also further explained in the later subtopic section of the paper.

1) Use case diagram: We first make the use case for the traffic system, since it is the centre of our system. The use case that we have built for the traffic system is based on our main integrated scenario where the system gives or allows vehicles that come and manage the queue of the vehicles. If the car requests to use resources, the traffic system is then either permitted if the resources are available or the car has to wait until the resources are available again. We made a use case diagram to summarize the scenario above so that the reader finds it easier to distinguish the boundaries between the system and the environment. In the system, it consists of vehicle move, plan route, detect car, view map and lane assist for the car so that any unwanted incident can be avoided. The use case diagram for the traffic system is shown in "Fig. 3".

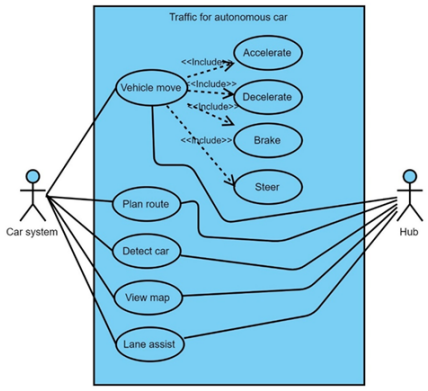

Fig. 3. Use case diagram.

2) Activity diagram handle car: Now, with the help of a use case model, more concrete diagrams can be refined. Based on the use case diagram, two activity diagrams can be constructed, which are activity for handle car and activity for manage resource. In "Fig. 4" show the activity diagram for handle car. Firstly, it will start by checking any request from the car. It will keep looping if there is no car request. After

a car is requested, it will accept the request to put the car in queue. Parallelly it goes to check any request from the car and also the car that has been put in the queue will go to the manage resource. “Fig. 4” shows the flow of the activity.

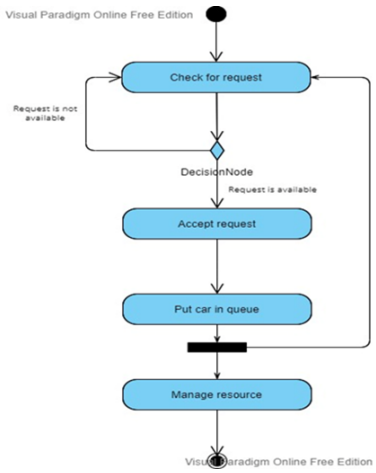

Fig. 4. Activity diagram handle car.

3) Activity diagram manage resource: For the activity of handle car,which is shown in “Fig. 5” , it will end when the car is put to manage resource. Now, the reader may know the flow of the activity of manage resource. The manage resource will access the car queue to choose a car based on the queue. After choosing the car, it will check the availability of the resources. The resources will keep checking and will only stop if all requested resources from the car are available for the car to use. Then it will lock the resources to avoid other cars to use the resources. Furthermore, the car might be able to move and after the car is leaving, the resources will unlock again so that other cars can use the resources.

4) Sequence diagram: In the following section, we continue our development process with the addition of a sequence diagram. With the sequence diagram, we can see the flow o use of the system given the context situation. This is shown i “Fig. 6”. We now have the ability to make the interface clear because, as mentioned before, the sequence diagram is used t identify any flaws and deficits of our system in the scenario In this sequence diagram, the relation is only between the ca and the traffic controller.

The traffic controller will always check any request fron the car. When the car arrives at the junction and sends a request, the traffic controller will accept the request and put the requesting car in the queue. The traffic controller then check the resource availability. After all the resources are available and can be used for the car, it will give permission and lock the resources for other cars. Then the car can use the resource and after leaving the junction, the traffic controller will unlock the resources and terminate the connection with the car.

5) System architecture: We are now at the next process stage, and we are almost near to defining the concrete system architecture. In this stage, we identify the required system components starting from the use case model until the se-

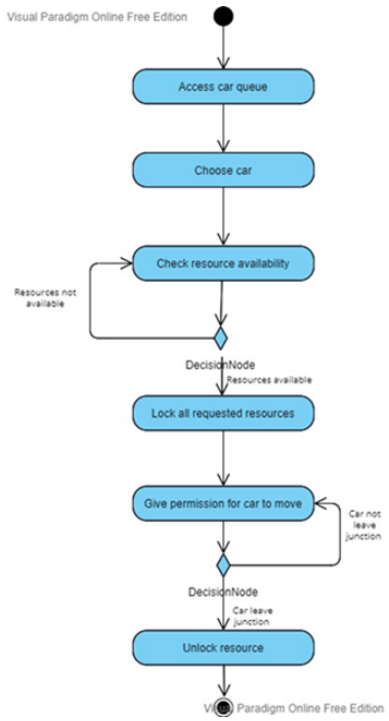

Fig. 5. Activity diagram manage resource.

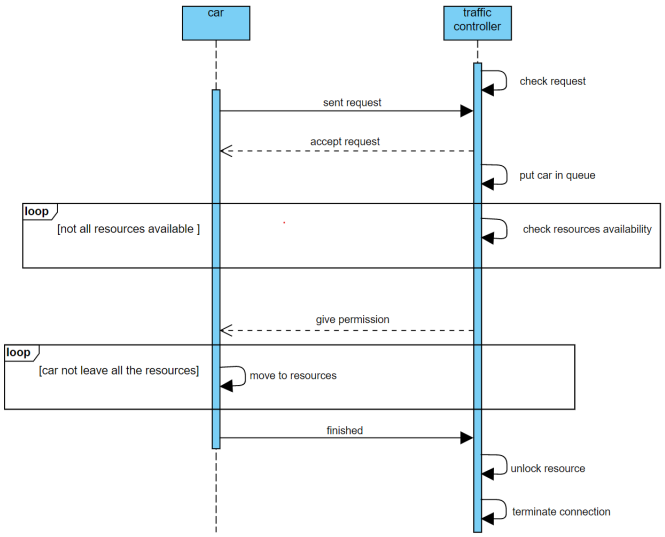

Fig. 6. Sequence diagram.

quence diagram that we have done before so that we can focus on the general structure of the system architecture. As we go through all the diagrams that are shown previously, we can see that they have the same pattern. Using that discovery, we then extract the pattern and generalize the information in order to sketch roughly our nearly concrete system architecture.

After that, we later refine again what we have sketched to produce our final concrete system architecture. The subsystem of the traffic controller is divided into four which are car queue, car controller, resources and communication. Inside the car controller, it is divided into two, which are a request handler to handle upcoming cars and resource manager to manage the usage of the resources. The final diagram produced can be seen in "Fig. 7".

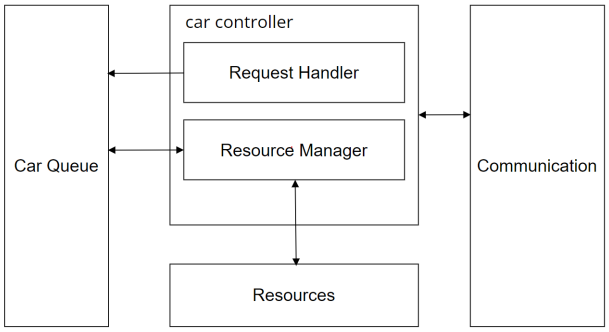

Fig. 7. System archirecture.

B. Behavioural modelling

In this section, we will describe the behavioural model of our project, which is an autonomous traffic system. This model then will be verified to ensure the system's reliability. In this project, we have two models, one with state chat and another one using VHDL to model the behaviour of our system. Those models are to make clear what will be going on in our system so that it will be easier to build a reliable and maintainable system.

1) State charts: For our state chart, we use a mealy state chart because the transition from a state to another state not only depends on the current state itself but the input from the external and internal signals of the system. Based on the system architecture that we discussed before, we have 5 components within the system which are:

- Request handler
- Car queue
- Resource
- resource manager
- communication

Based on the given components, we only model 4 state charts that represent the behaviour of the request handler, car queue, resource and resource manager. The communication is implicitly modelled in the request handler and resource manager model. For request handler, there are three states as shown in "Fig. 8". It will start with an idle state where the component is waiting for any request from a car. When there is a requesting car, a request signal will be produced. Thus, the state of the component is triggered to change its state to waiting where it tries to access the car queue by releasing wait primitive and change its state to update to be able to put the requesting car on the list whenever the queue is available. After successfully putting the car in the queue, the component will release signal primitive and back to idle state.

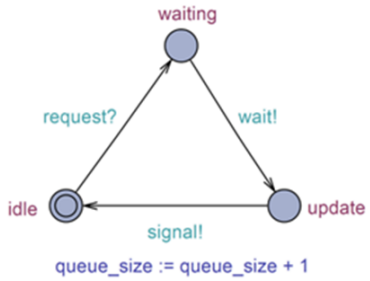

Fig. 8. Request handler component.

As shown in "Fig. 9" there are two states for the queue which is idle and busy. Idle means that the queue is available to be accessed for read and write while the busy state means that the queue is currently accessed by a critical section of another component. The state will change from idle to busy whenever it receives wait primitive and change back to idle whenever it receives signal primitive.

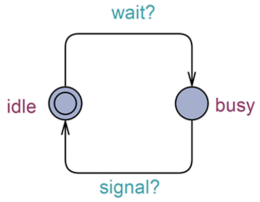

Fig. 9. Car queue component.

Based on the sequence diagram that we have discussed in the previous section, the car in the queue will be handled. The component that handles the car that is already in the queue is called resource manager as stated in the system architecture. The state chart for this component as shown in "Fig. 10" has 6 states. It will start with listening state to check the existence of a car in the queue. If one or more cars are in the queue, the component will release a wait signal to edit the queue where a car will be taken from the queue.

After that, it will release signal primitive once it successfully copies the car information and remove the car from the queue. In checking state, the component will check the availability of every requested resource. If all requested resource is available, the component will the lock all the resource for the car by releasing the lock resource signal and change it state to

ready where it will give the permission to the car to use all the resources and directly change its state to allocate where the component waits a signal from the car after the car have finished using all the resources. Once the component receives the signal it will change it state to reset where all the previously locked resource will be unlocked. After unlocking all the resource, the component will back to listening state to handle another car in the queue if available.

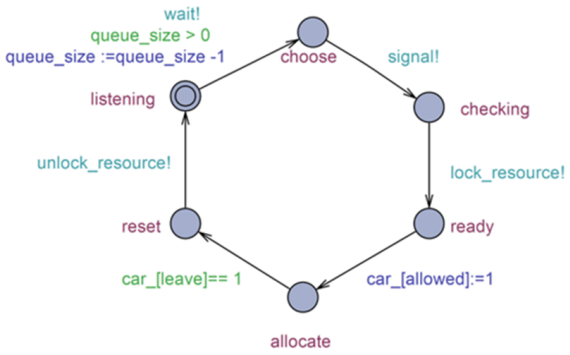

Fig. 10. Resource manager component.

"Fig. 11" shows the state chart for the last component which is the resource. It has two states which is unlocked and locked. The component will change its state from unlock to lock whenever it receives lock_resource signal and change back to the unlocked state once it received unlock_resource signal.

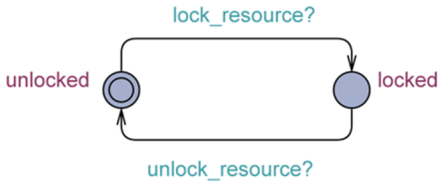

Fig. 11. Resource component.

Next, all the state charts are integrated in UPPAL for the simulation. This simulation is to verify the behaviour model of our system is free from deadlock and can be executed successfully. The model for simulation In UPPAAL can be accessed at [1].

For the simulation, a dummy state chart of the car was also modelled as shown in "Fig. 12" in order to be able to see the simulation fully work. It will start with idle and then change its state to requesting after making a request by releasing a request signal. It will wait for a permission signal from the autonomous traffic system in requesting state. It will then change its state to a moving state after receiving the permission signal and then go to leave the state after using all the resources. This is where the signal from the car is produced that later will be used by the resource manager to indicate that the car has already left all the resources. The gone state means that the car has already left the system.

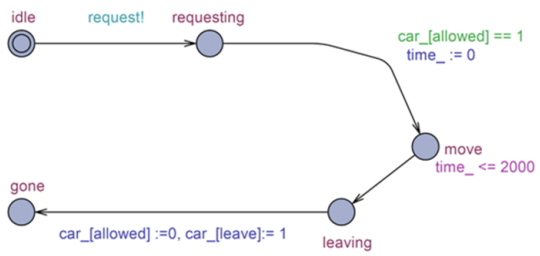

Fig. 12. Dummy car state chart.

2) VHDL: Then we move to the VHDL model. With the VHDL model, we model only one component, which is the request handler. The entity of the component is shown below:

```
entity requestHandler is
port(   rst, clk: in std_logic;
systemInput :
in std_logic_vector(1 downto 0);
data : in std_logic_vector(19 downto 0);
systemOutput :
out std_logic_vector(19 downto 0) );
end entity requestHandler;
```

When this component receives request on the system input, it will then transfer the requesting car data that consisting the ID of the car and its direction or better known as requested resource to car queue component. This data letter on will be use by resource manager to allow to car to use the requested resource.

The full VHDL model for request handler with its behavioural can be accessed at [1]. We verify the model using the test bench by giving data which are car ID and directions. We then simulate it to see the result by giving an input which is "request" and the result is as expected which is the car ID and the direction of the car as the output of the model.

By using this kind of model, we could possibly synthesise the model in the future to produce the schematics of the component.

IV. Prototype Implementation

In this section, the prototype implementation of our system using freeRTOS is described. A Real-Time Operating System, commonly known as an RTOS, is a software component that rapidly switches between tasks, giving the impression that multiple programs are being executed at the same time on a single processing core. In order to simulate our system, RTOS is the perfect operating system to be implemented in our simulation, due to the multiple tasks that are present in our system. Our system consists of a total of 6 tasks, three of which are different from each other which are addRequest, task_request_handler and resource_manager. Our system has 1 addRequest task, 1 task_request_handler task and 4 resource_manager task, resulting in 6 tasks running.

The addRequest task is simply to simulate requests from cars coming to our traffic system. Whenever the button is pushed, the addRequest will simulate incoming requests from so-called cars which are class objects that have their own attributes. The task does this by inserting new requests in an array called request_array that has the size of 100 car objects. After it has done inserting the requests, the tasks goes back to idle state where it waits for another button push to simulate another set of requests.

The task_request_handler task handles the request that was "created" by the addRequest task, and sorts the cars into a queue. It will first search the request_array if there are any request available. If one is available, it will take that car object and insert it into a queue called the car_queue. In order for it to access the car_queue, we have implemented a mutex to avoid unwanted resource manipulation. The task must first check if the car_queue is available to access via mutex. Once it has the permission to access the queue, it will then add the car object from the request array into the queue. Then, it returns the mutex after it is done accessing the car_queue and start scanning the request_array again for new requests.

The resource_manager task has the responsibility to manage the resources of the traffic to the car, depending on their requests. Each car will have their own requested resources which reflects on the direction that the car is going through the junction. The task will first scan the car_queue for any car requesting to go through the junction. If one is available, it will take the car object from the queue. Since the car_queue is a critical section, this task also is required to check whether the queue is available to access via mutex. Only when it is available can it access the queue and take the car object from the queue. The task then checks the resources that the car that was took from the queue had requested. The resources of the traffic junction are also protected by mutex. Thus, if the resources is not used by another other car, meaning that the mutex of the resource is available, the resource_manager task will then take the mutex of that resource, and "give" the resource to the car object. When the car object is done using the resource, or in other words done going through the junction, the resource will be "returned" to the resource_manager task, and the mutex will be given back for other cars to use. Then, the task scans the car_queue again.

The code of the simulation is then uploaded to an Arduino Mega as our prototype platform. The full simulation code can be accessed at [1].

V. Project Management

For project management, we use the scrum method where we divide the time that we have to develop this product into 11 cycles and each cycle takes 1 week. To make it more efficient, we have two minimum meetings in each cycle. In the first meeting, we will discuss and brainstorm together the big task that we need to complete in the cycle. As the result, the sub-tasks are created and distributed fairly among the team members. The second meeting is to verify the current progress and make improvements. Since we use the scrum method, we still help each other in completing each sub-task to ensure the product of each cycle is at its highest and finest quality.

VI. Conclusion

In this documentation, we presented what we have developed which is an autonomous traffic system with the main focus design is to tackle the problems regarding the traffic in our real life. We have explained in detail how we achieved the final product through multiple incremental steps of analysis and design during the last few months of official academic hours. We also included the link to our source code and many different diagrams to help the reader better understand what is being explained. With the help of architectural model and behaviour model, like use case, activity, sequence diagrams, state chart and VHDL. We managed to create the main components of the system. After developing the individual components of our autonomous traffic system, we generalised the pattern of interaction between the environment and the devices, and came up with the overall system architecture. Then, we focused on a specific scenario based on the state charts and verified with the help of VHDL. Finally, after verifying the state chart, we implement FeeRTOS for the implementation. The autonomous traffic system that was documented by this paper only has a constructed framework that is available for future projects, and has realised a specific scenario of the traffic system. Although we made a huge step in the right direction, we feel as though there are many more improvements that can be made in the future. Improvements that we wanted to make are further implementation of multiple special features and scenarios for our system. For example, how to deal with the cars and when the pedestrians want to cross the road. Nevertheless, we are grateful to have been guided to have some experience in developing our own product for the future. In the future, we hope that our system can be more refined and finally realised to the main world and introduce the system to the main public that not only eases the user but also helps to solve the traffic problems. We also hope to be given more opportunities like this to have a better understanding of our study course.

VII. 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.

VIII. Acknowledgement

We are heartily and most grateful to our beloved professors, Prof. Stefan Henkler and, Prof. Dr.Rettberg, Achim, whose inspiration, encouragement, guidance, and support in the beginning towards the final level enabled us to develop awareness and also open the eyes of how important traffic systems are in the real world. We would also want to offer our gratitude, respect, and appreciation to any and all persons that assisted us in any way during the completion of the task. In addition, we would like to emphasise that in this article, each member

of our group is responsible for each task, from designing the requirement diagram to writing the documentation, not to mention the implementation code. Every creative decision was discussed and agreed with all team members during our team meeting. We tried our best to make everyone feel involved in the project and make sure that the reader will get a deeper understanding after reading this article.

REFERENCES

[1] https://github.com/Adib6637/AutonomusTraficSystemHoffenheim

[2] High Integrity Systems. 2022. What is an RTOS - Real Time Operating System Information and Training. [online] Available at: ¡https://www.highintegritysystems.com/rtos/what-is-an-rtos/¿ [Accessed 19 June 2022].