Autonomous Cross Traffic Management

Adijat Ajoke Sulaimon
dept. Electronics Engineering
Lippstadt, Germany
Adijat-ajoke.sulaimon@stud.hshl.de

Amit chakma
dept. Electronics Engineering
Lippstadt, Germany
amit.chakma@stud.hshl.de

Saikot Das Joy
dept. Electronics Engineering)
Hamm, Germany
saikot-das.joy@stud.hshl.de

***Abstract*—Autonomous Vehicles require quick and clever autonomous traffic management system for safe and efficient operation and also to reduce traffic congestion. However, the rapid development in technology, vehicular communication can provide cooperative driving of intelligent vehicles with efficiency, sustainability, and safety at intersections and transportation systems in general. This project proposes a method to autonomous controlling of vehicular movements and crossing intersections using Queue theory. The objective of the project is to model a traffic management system that will ensure no collision among vehicles while cross a 4 way cross traffic intersection.**

***Index Terms*—Codes, diagrams**

I. Introduction

An intersection can be a junction or a region of the road where two or more than two roads meet. An intersection with 3 way is called T-junction or a fork. A four way intersection is called crossroad, there can also be five way, 6 way and even more. It is essential for intersections to be controlled to reduce or eliminate traffic congestion. In a junction or intersection, there is usually a queuing system that give priority to a prioritized automobile to regulate traffic congestion. In recent times, there have been a significant advancement like the autonomous vehicles which are also known as self driven vehicles. One of the biggest challenges of the autonomous vehicles is managing and mitigating traffic congestion in intersections. This traffic intersection management system for autonomous vehicles is the vehicle-to-infrastructure(V2I) system.

This Project is a simpler version of Autonomous Traffic control for cross traffic . Interested readers can find a complex but more realistic version of this project (only system design) of at the index.

We started by making the requirement diagram. Requirement diagrams are essential to enumerate the exact problems to be solved. A requirement diagram will answer question like what exactly we intend to achieve and the problems are project will solve. We also did use case diagram, state machine diagram among others. After the diagrams has been done and reviewed, we did an abstract implementation on UPPAAL using the state machine diagram. Finally, we used freeRTOS for the final implementation. The most challenging part of this project is the freeRTOS implementation. None of us knew anything about freeRTOS before this project but now we know enough to implement a traffic management system for autonomous vehicles.

II. System Design

A. Requirement diagram

In the Requirement diagram in Figure 1 we can see the main requirements of the system. As shown in the figure 1, controlling the cross traffic management system is a high level requirement as the main goal is to manage the cars in a 4 way intersection. There are 5 sub level requirements for the cross traffic management system.

Real time system - The system being real time is one of the main requirements, as we need to handle the vehicles in an autonomous system. A real time system should be reliable. As a more specific requirement we have derived the requirement to control the queue of cars in a first come first serve manner.

4 way intersection - The system should be able to handle 4 way intersection as it was one of the main requirements for the project.

Communication - The system should be able to establish stable communication to send data to the autonomous vehicles and to receive data from it. As a more specific requirement we will have vehicle to vehicle (v2v) , vehicle to intersection (v2i) and wireless communication.

Safety - The system should be safe to use and avoid any kind of collision. Without safety features the system will be very dangerous as it is one of the main goals to achieve.

B. Use case diagram

In the use case diagram Figure 3 we have primary actor cars and interaction and controller service (ICS) as secondary actors. The car should be able to connect wirelessly with ICS, pass directional signals , send completion messages , cross intersections and wait for signals from ICS. So the primary actor car has an associated relationship with connecting wirelessly , passing directional signals , passing completion messages , crossing intersections and waiting.

The ICS should be able to create a wireless network , receive directional signals from the car , make priority scheduling , receive completion messages from the car and allow the

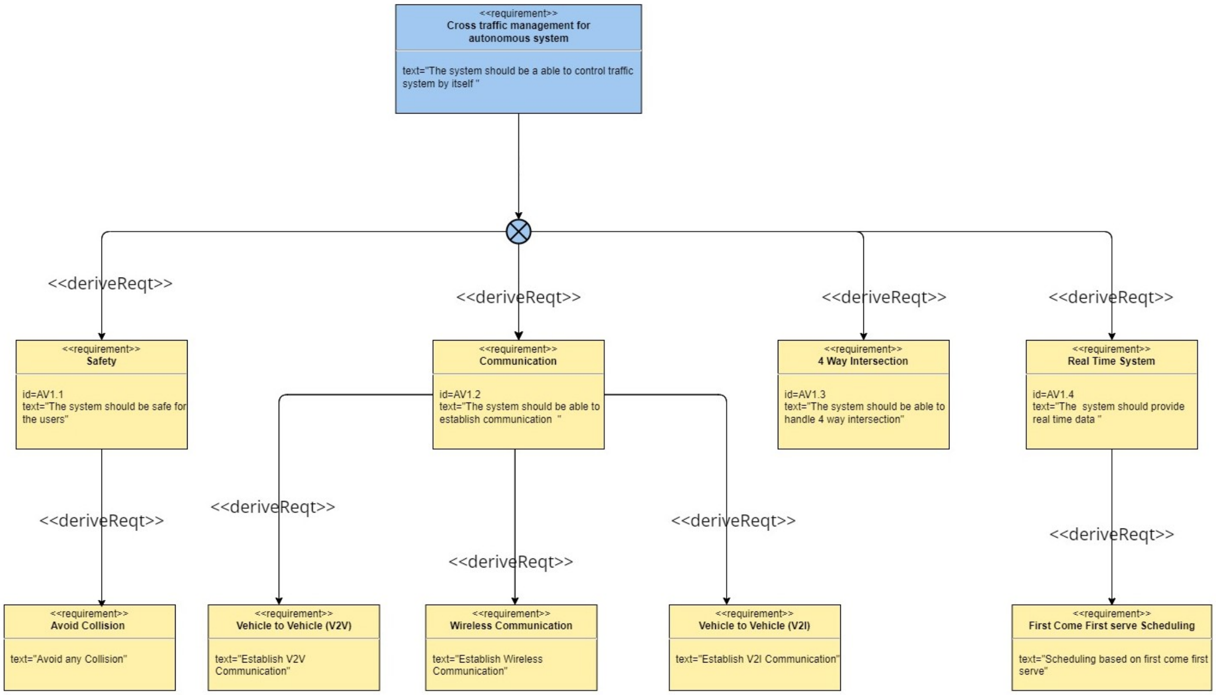

<<requirement>>
Cross traffic management for autonomous system
text="The system should be a able to control traffic system by itself "
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<requirement>>
Safety
id=AV1.1
text="The system should be safe for the users"
<<requirement>>
Communication
id=AV1.2
text="The system should be able to establish communication "
<<requirement>>
4 Way Intersection
id=AV1.3
text="The system should be able to handle 4 way intersection"
<<requirement>>
Real Time System
id=AV1.4
text="The system should provide real time data "
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<deriveReqt>>
<<requirement>>
Avoid Collision
text="Avoid any Collision"
<<requirement>>
Vehicle to Vehicle (V2V)
text="Establish V2V Communication"
<<requirement>>
Wireless Communication
text="Establish Wireless Communication"
<<requirement>>
Vehicle to Vehicle (V2I)
text="Establish V2I Communication"
<<requirement>>
First Come First serve Scheduling
text="Scheduling based on first come first serve"

Figure 1. Requirement Diagram

vehicle to cross the intersection.So the secondary actor ICS has an associated relationship with the following use cases : connect wirelessly, receiving directional signals , priority scheduling , receiving completion messages, and allowing the vehicle via message to cross.

Base use case Passing directional signal have included use case Receiving directional signal so whenever the car sends a signal the ICS receives the signal. Base use cases Passing completion messages have included use case receiving completion messages meaning whenever a car completes passing the intersection it sends a signal to let the ICS know that the crossing intersection is complete. Base use cases allowing the vehicle via message to cross have included use case crossing Intersection to let the car pass the intersection.

C. State machine diagram

The behavior of an entity is not only a direct consequence of its inputs, but it also depends on its preceding state. The past history of an entity can best be modeled by a finite state machine diagram or traditionally called automata. UML State Machine Diagrams (or sometimes referred to as state diagram, state machine or state chart) show the different states of an entity. State machine diagrams can also show how an entity responds to various events by changing from one state to another.

In Figure 4, the state behaviours of Autonomous Traffic control system is shown.

The beginning state is named as "Arrival" state which start as soon as a autonomous car arrives in the intersection. then the state moves to the "Connection state" where the car is wirelessly connected with the Intersection Controller Server(ICS). This state sends directional signal to the "Receiving" state of the ICS. Directional signal is the signal containing data of from which direction to what direction the car intends to go. Then the state changes to the "Scheduling" state of ICS where Information of incoming cars are kept in queues on first come first served basis. Then the state of ICS changes to "decision" state where cars data with earliest arrival time are picked , and the state changes to the next state named "Allowence" state. This state forwards the pass signal to the "Receiving" state of the autonomous Car. If there is no Pass/Cross signal , the car waits in the "Waiting" state otherwise the state changes to "driving" state of the car. When The car completes passing the intersection, The state is "Completion" state . from this state 'Completion' signal is passed to the "Receiving" state of ICS and the queue in ICS is updated then. This procedure continues for more and more incoming cars.

D. Activity diagram

Here we used an activity Diagram Figure 5 with swimlanes which makes the system easy to understand. We grouped all the main activities in a single diagram. From the start node to start activity Initially the vehicle/car connects to the system and sets direction . The ICS checks the database and schedules the vehicle and puts on a queue based on first come first serve basis. After a time duration the ICS allows the car to pass the intersection . The car passes the intersection , after passing the intersection the car sends a signal with final activity and the ICS updates the database.

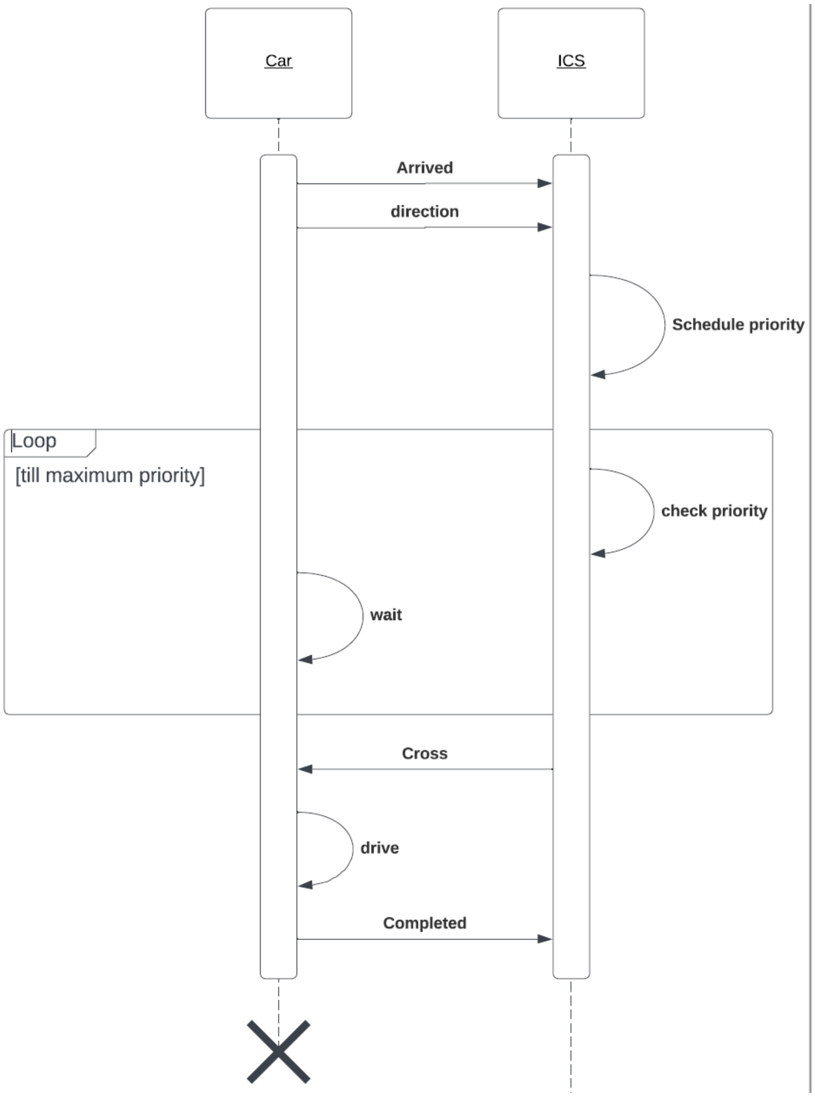


Figure 2. Sequence diagram

E. Sequence diagram

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline.

in Figure 2 , The message transfer between autonomous car and the Intersection COntroller Service(ICS) is shown.

III. IMPLEMENTATION

A. UPPAAL

Fig 6 was implemented with UPPAAL based on our state machine diagram. The initial state is the start state. When the vehicles arrives at the intersection, they wait at the arrival state and then connects to the ICS. After the ICS has received the connection message sent, the vehicle will queue in till it gets permission signal to cross the intersection. It will then drive off and message will be sent to the system that the vehicle has passed and the process is completed. P3 and P4 in the figure 6 shows the system part of the traffic management system. The start state is the initial state. The system receives direction from the vehicle and then schedule and place the vehicle in it's right lane where it will queue and wait for it's turn. The system will check the deadline and decide either to pass the vehicle or not, if the vehicle has lower deadline, it will keep waiting. If it has a higher deadline, it will be passed and the vehicle will be allowed to pass.

B. FreeRtos

We have only implemented the queueing part of the ICS(intersection controller server) in freeRTOS. We sent 10 cars to the queuing and send pass signal to the cars to cross intersection on a First Come First Served basis using the queue. We have also used esp32 micro-controller for this purpose.

```
// declare to use only 1 CPU core for the tasks
#if CONFIG_FREERTOS_UNICORE
#define ARDUINO_RUNNING_CORE 0
```

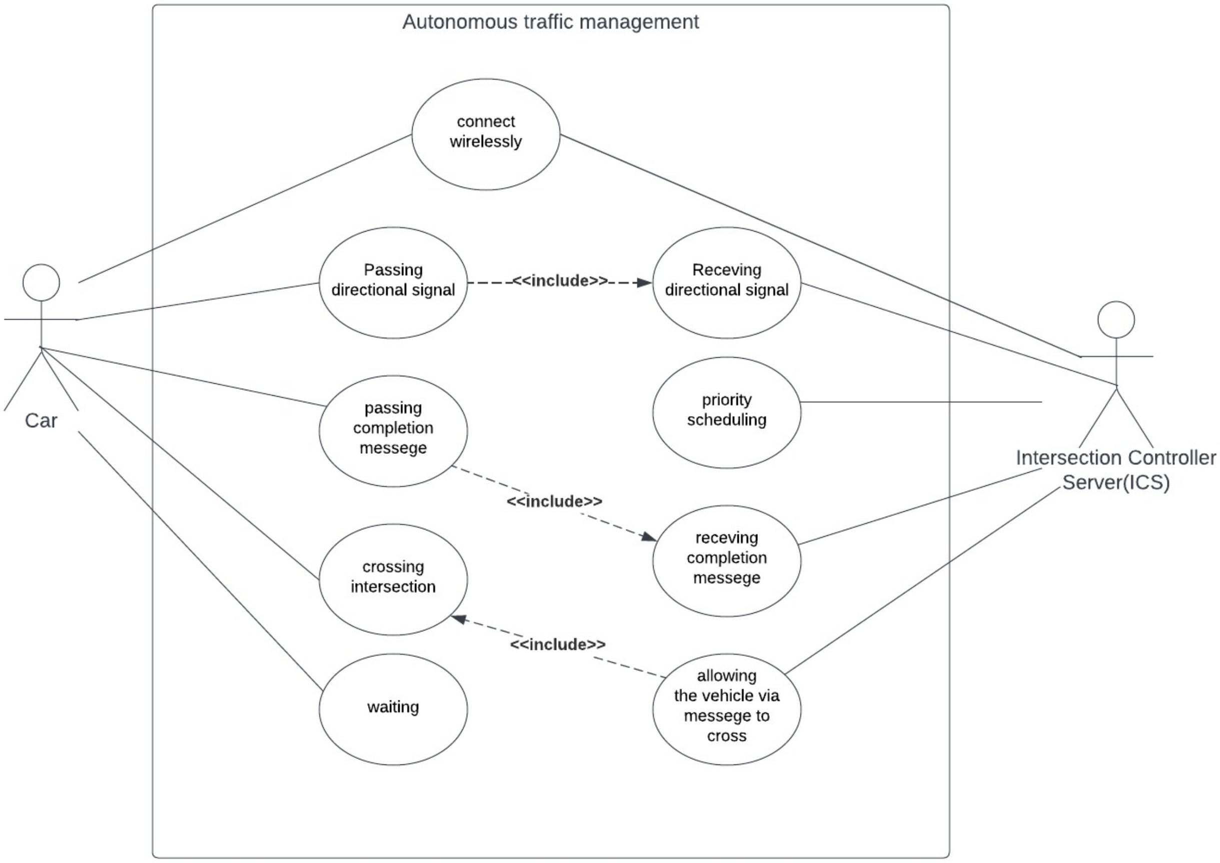

Autonomous traffic management
connect wirelessly
Passing directional signal
<<include>>
Receving directional signal
priority scheduling
passing completion messege
<<include>>
receving completion messege
crossing intersection
<<include>>
allowing the vehicle via messege to cross
waiting
Car
Intersection Controller Server(ICS)

Figure 3. Use case Diagram

```
#else
#define ARDUINO_RUNNING_CORE 1
#endif
//functions declarations
void sender_task(void *p);
void receiver_task(void *p);

//Global queue declaration so that all functions can acces the queue
QueueHandle_t Global_Queue_Handle=0;

// A variable declaration to check the car crossed the intersection or not
int completion_status=0;

void setup()
{
//set up serial monitor with 9600 baud rate
Serial.begin(9600);

// create a queue
Global_Queue_Handle= xQueueCreate(10,sizeof(char));
//create a task that sends car information to queue
xTaskCreate(sender_task,(const char*)"sender_task",1024,NULL,2,NULL);
// Create a task that receive car information from queue
xTaskCreate(receiver_task,(const char*)"receiver_task",1024,NULL,1,NULL);
}

void loop()
{
// keep the loop empty
}

// functions details
void sender_task(void *p)
{

   // declare some car with character information
   // Character start from A and continues
    char car1=’A’;
    char car2=’B’;
    char car3=’C’;
    char car4=’D’;
    char car5=’E’;
    char car6=’F’;
    char car7=’G’;
    char car8=’H’;
    char car9=’I’;
    char car10=’J’;
      // start sending car data serially with some delays
      xQueueSend(Global_Queue_Handle,&car1,1000);
      Serial.println("Car A arrived and sent to queue");
      vTaskDelay(500/portTICK_PERIOD_MS);
      xQueueSend(Global_Queue_Handle,&car2,1000);
```

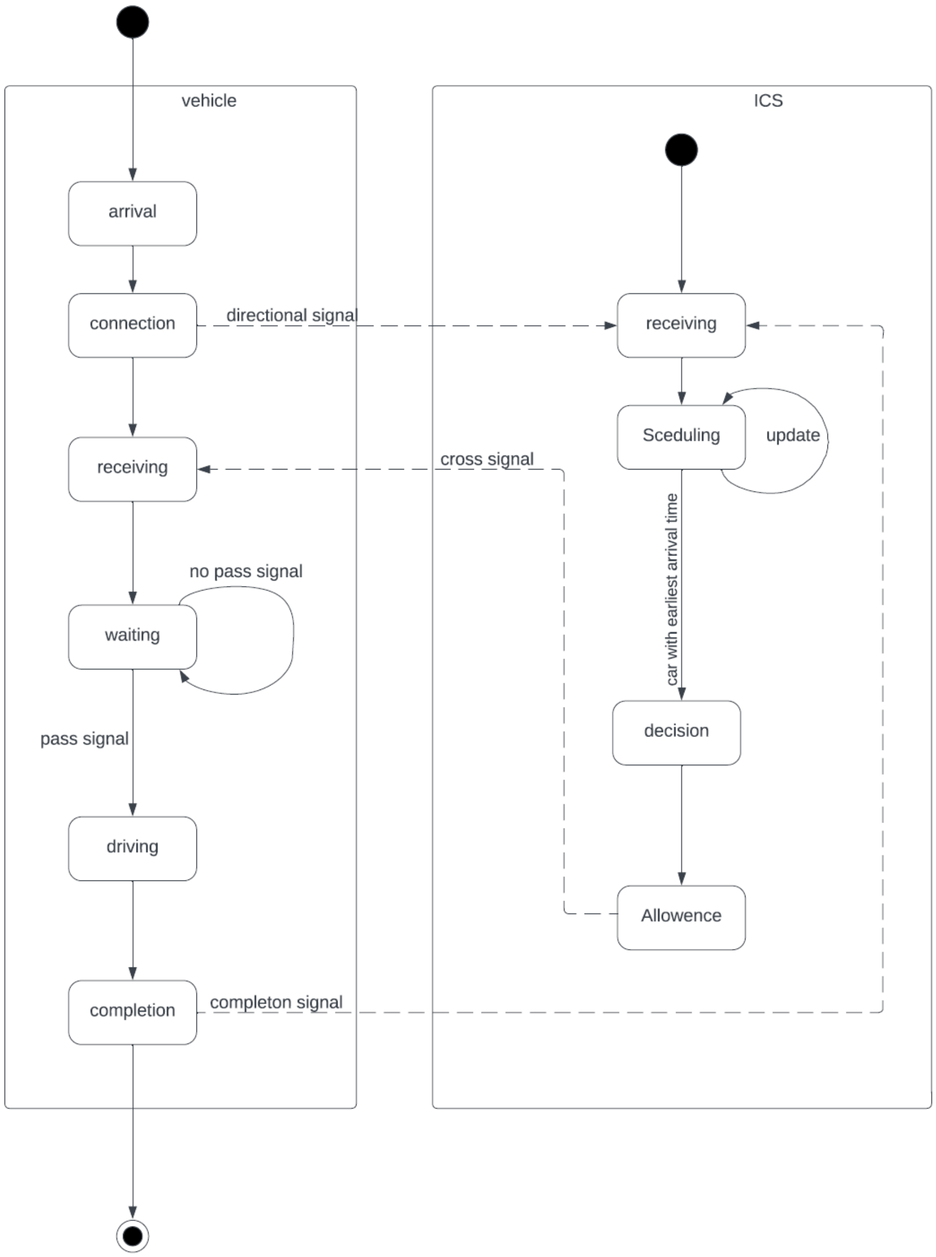


Figure 4. State Machine Diagram

```
    Serial.println("Car B arrived and sent to queue");
    vTaskDelay(1000/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car3,1000);
    Serial.println("Car C arrived and sent to queue");
    vTaskDelay(1500/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car4,1000);
    Serial.println("Car D arrived and sent to queue");
    vTaskDelay(700/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car5,1000);
    Serial.println("Car E arrived and sent to queue");
    vTaskDelay(600/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car6,1000);
    Serial.println("Car F arrived and sent to queue");
    vTaskDelay(1100/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car7,1000);
    Serial.println("Car G arrived and sent to queue");
    vTaskDelay(1000/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car8,1000);
    Serial.println("Car H arrived and sent to queue");
    vTaskDelay(1600/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car9,1000);
    Serial.println("Car I arrived and sent to queue");
    vTaskDelay(900/portTICK_PERIOD_MS);
    xQueueSend(Global_Queue_Handle,&car10,1000);
```

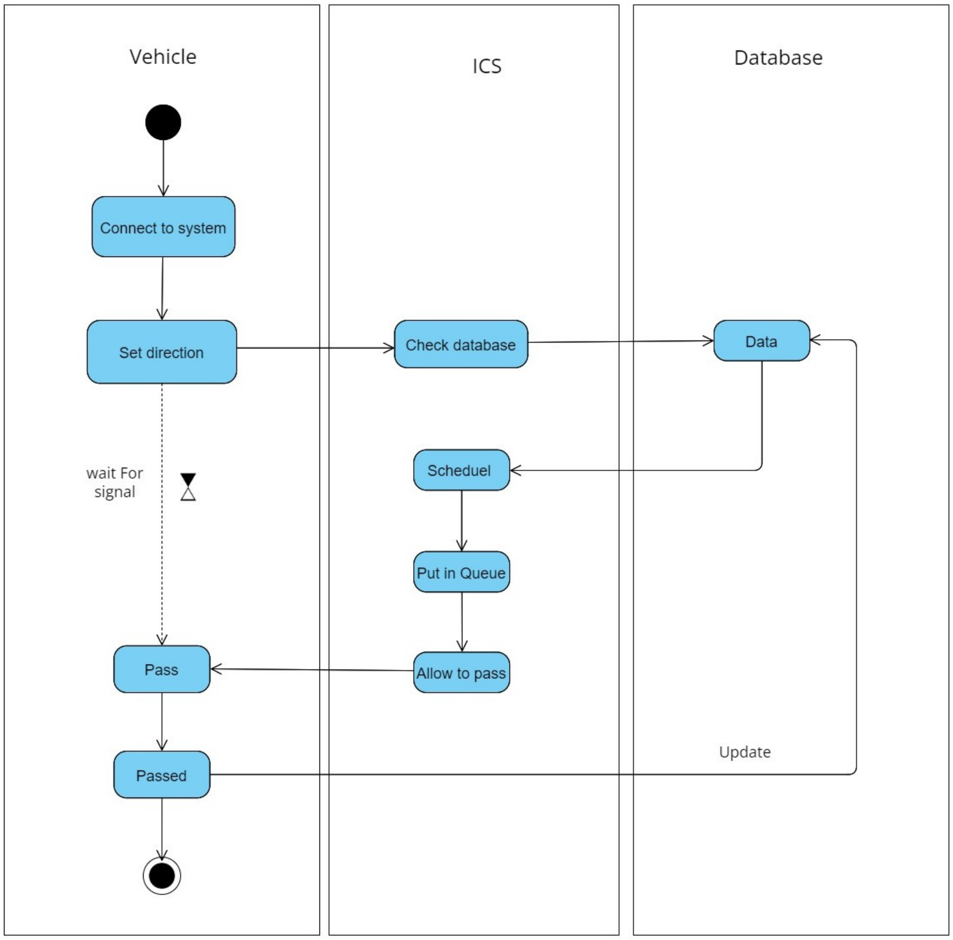


Figure 5. Activity Diagram

```
      Serial.println("Car J arrived and sent to queue");
      vTaskDelay(400/portTICK_PERIOD_MS);
}
// functions details
void receiver_task(void *p)
{
  char rx;
    while(1)
    {
      // start receiving data from queue
      if(xQueueReceive(Global_Queue_Handle,&rx,1000)==pdTRUE)
      {
      Serial.print("Car ");
      Serial.print(rx);
      Serial.print(" Received from queue");
   // wireless signal sending and receiving part is only shown in serial monitor for simplicity
      if(completion_status==0)
          {
                  Serial.print("Sending Pass signal to the Car ");
                  Serial.print(rx);
                  Serial.println("waiting to recivee completion signal from car");
                  vTaskDelay(3000/portTICK_PERIOD_MS);
                  Serial.print("Completion signal received from car ");
                  Serial.print(rx);
           }

       }
    }
}
```

C. Serial Monitor View of FreeRTOS Implementation

in Figure 7, When we uploaded the code in esp32, the serial monitor view is shown.

D. ModelSim

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY MYFIFO_1 IS
 PORT ( DATAIN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
  EN,CLK,RST : IN STD_LOGIC ;
  W : IN STD_LOGIC;
  DATAOUT : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0);
  RED : OUT STD_LOGIC);
END MYFIFO_1;

ARCHITECTURE BEHAVIORAL OF MYFIFO_1 IS

SIGNAL WPTR, RPTR : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

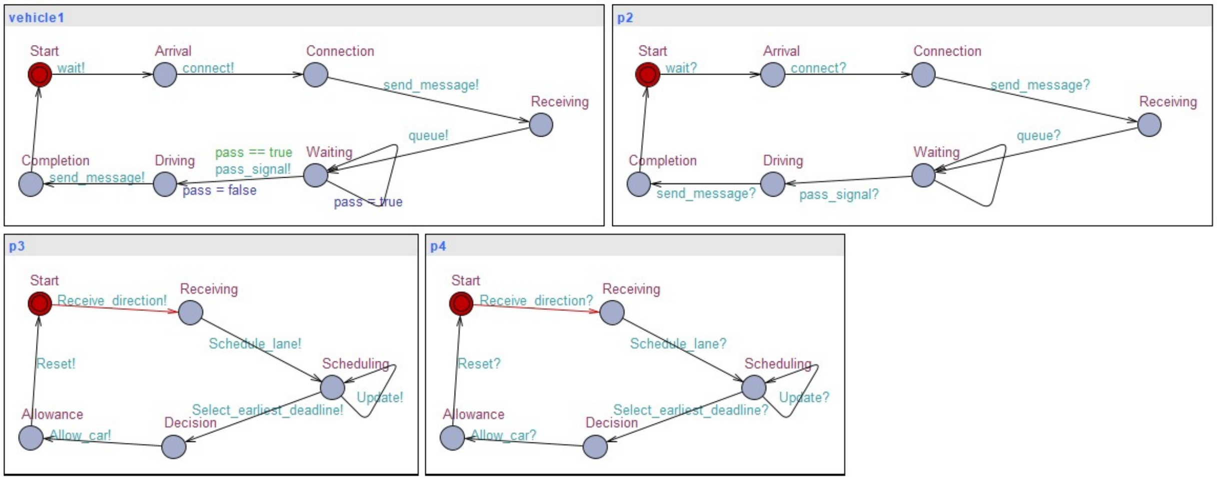
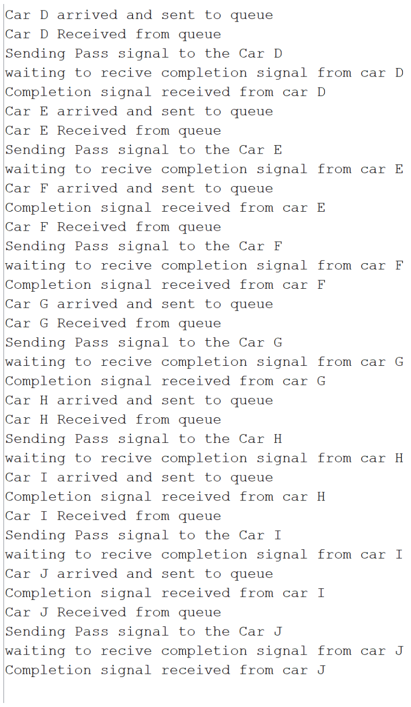


Figure 6. UPPAAL Implementation



```
Car D arrived and sent to queue
Car D Received from queue
Sending Pass signal to the Car D
waiting to recive completion signal from car D
Completion signal received from car D
Car E arrived and sent to queue
Car E Received from queue
Sending Pass signal to the Car E
waiting to recive completion signal from car E
Car F arrived and sent to queue
Completion signal received from car E
Car F Received from queue
Sending Pass signal to the Car F
waiting to recive completion signal from car F
Completion signal received from car F
Car G arrived and sent to queue
Car G Received from queue
Sending Pass signal to the Car G
waiting to recive completion signal from car G
Completion signal received from car G
Car H arrived and sent to queue
Car H Received from queue
Sending Pass signal to the Car H
waiting to recive completion signal from car H
Car I arrived and sent to queue
Completion signal received from car H
Car I Received from queue
Sending Pass signal to the Car I
waiting to recive completion signal from car I
Car J arrived and sent to queue
Completion signal received from car I
Car J Received from queue
Sending Pass signal to the Car J
waiting to recive completion signal from car J
Completion signal received from car J
```

Figure 7. Serial Monitor View of FreeRTOS Implementation

```
TYPE FIFO IS ARRAY(15 DOWNTO 00) OF STD_LOGIC_VECTOR
    (07 DOWNTO 00);
SIGNAL MEM:FIFO;

BEGIN

PROCESS(CLK,RST)
BEGIN
 IF RST = ’1’ THEN
  WPTR <= "0000" ;
  RPTR <= "0000" ;
  RED <= ’0’ ;
 ELSE IF(CLK’ EVENT AND CLK=’1’) THEN
  IF(EN = ’1’) THEN
   IF(W = ’1’) THEN               -- Write Operation
    IF(WPTR < "1111") THEN
      MEM(CONV_INTEGER(WPTR)) <= DATAIN;
     WPTR <= WPTR + 1 ;
    ELSE
     RED <= ’1’ ;   -- Memory Full Warning
    END IF ;
   ELSE                          -- Read Operation
    IF(RPTR < WPTR) THEN
     RED <= ’0’;
     DATAOUT <= MEM(CONV_INTEGER(RPTR));
     RPTR <= RPTR + 1;
    ELSE
     RED <= ’1’ ;   -- Empty Memory Warning
     DATAOUT <= "00000000" ;
    END IF ;
   END IF ;
  END iF ;
 END IF ;
END IF ;
END PROCESS ;
END BEHAVIORAL ;
```

E. *Simulation Window of Modelsim implementation*

We have created a FIFO queue memory of size 16 that means that it can hold 16 data maximum (each of which is 8 bit data) and after that it is full if data is not taken out.In the simulation window in Figure 8, we put three 8 bit data on the FIFO memory as data input (one by one) and after that, when we look at the data output one by one, we find the same data as same order we put as data input.

IV. Summary

The primary aim of this project to to model a traffic in intersection for autonomous cars using Queuing system and to decide the parameters to be considered to simulate traffic intersection using freeRTOS.

We have been able to achieve the said aim by modeling and simulating a traffic intersection for autonomous cars using queuing system. The communication method being vehicle to infrastructure and crossing intersection on a first come first served basis.

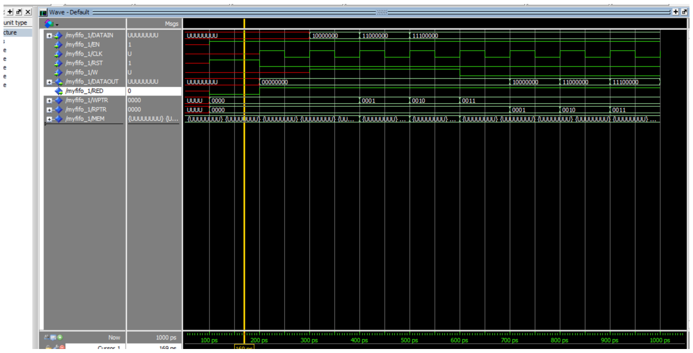


Figure 8. Wave view of FIFO Queue simulation in Modelsim

References

[1] https://www.driverseducationusa.com/resources/what-is-an-intersection/

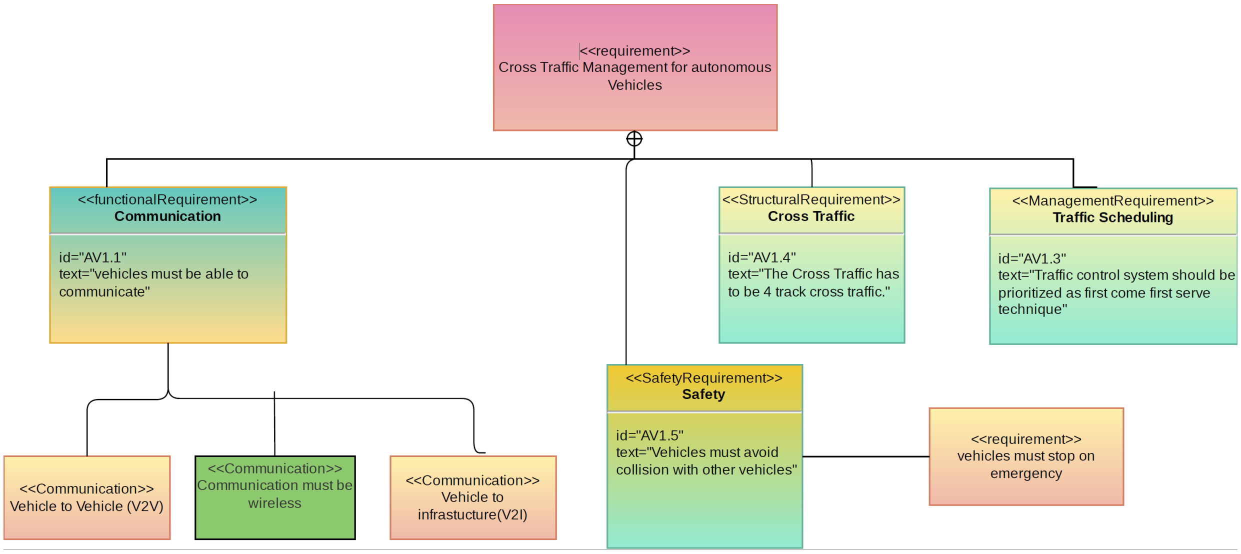
[2] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/

V. Index

A. Extended version of the system design

An extended version of the system design for autonomous intersection management project is shown below as diagram . Our team members did this at the beginning of our project , though we considered the simpler version of this in our further work as shown in this documentation paper.

You can find System Design as diagram on the next page.



<<requirement>>
Cross Traffic Management for autonomous Vehicles
<<functionalRequirement>>
Communication
id="AV1.1"
text="vehicles must be able to communicate"
<<StructuralRequirement>>
Cross Traffic
id="AV1.4"
text="The Cross Traffic has to be 4 track cross traffic."
<<ManagementRequirement>>
Traffic Scheduling
id="AV1.3"
text="Traffic control system should be prioritized as first come first serve technique"
<<SafetyRequirement>>
Safety
id="AV1.5"
text="Vehicles must avoid collision with other vehicles"
<<requirement>>
vehicles must stop on emergency
<<Communication>>
Vehicle to Vehicle (V2V)
<<Communication>>
Communication must be wireless
<<Communication>>
Vehicle to infrastucture(V2I)

Figure 9. Requirement Diagram

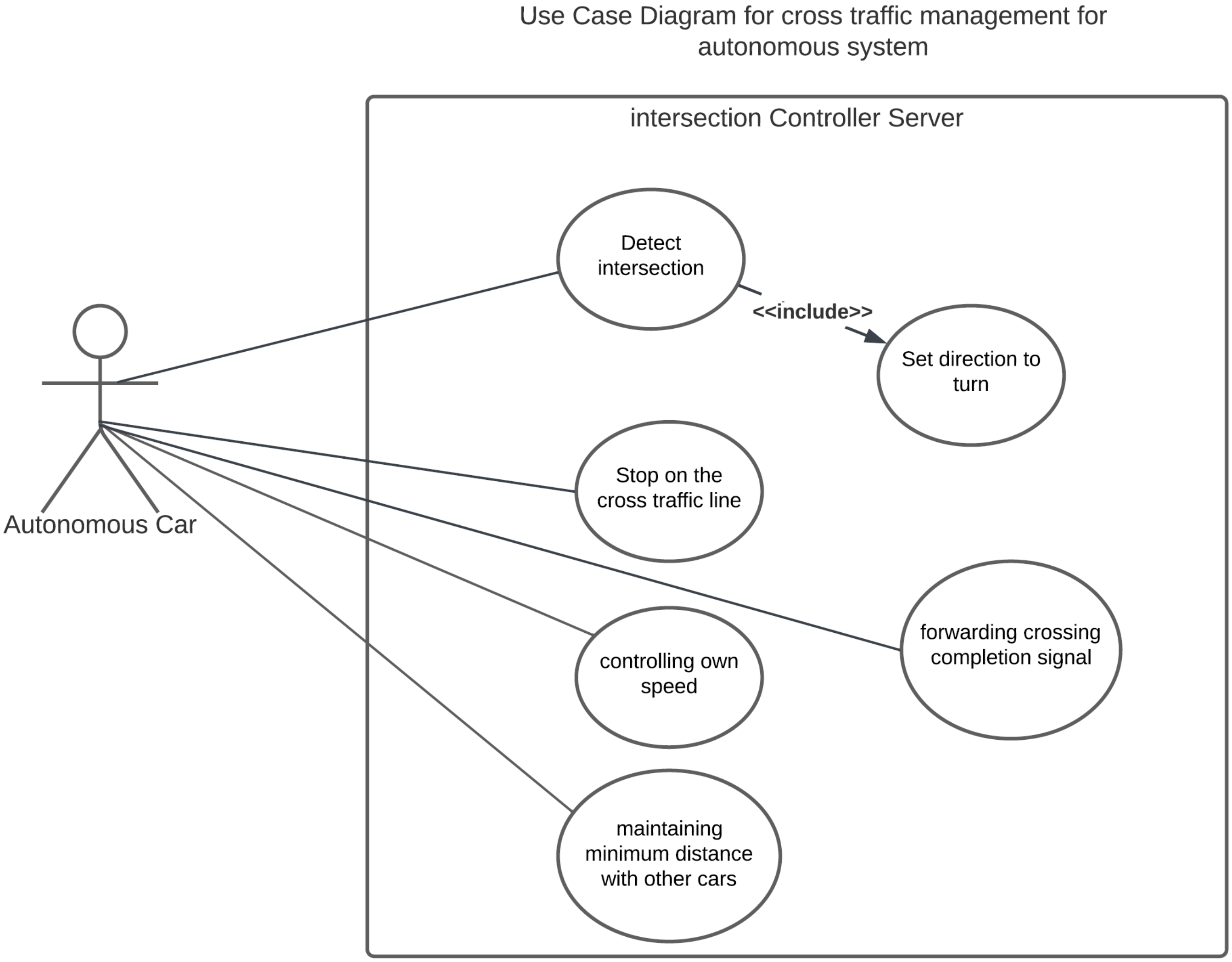

Use Case Diagram for cross traffic management for autonomous system
intersection Controller Server
Detect intersection
<<include>>
Set direction to turn
Stop on the cross traffic line
forwarding crossing completion signal
controlling own speed
maintaining minimum distance with other cars
Autonomous Car

Figure 10. Use case diagram

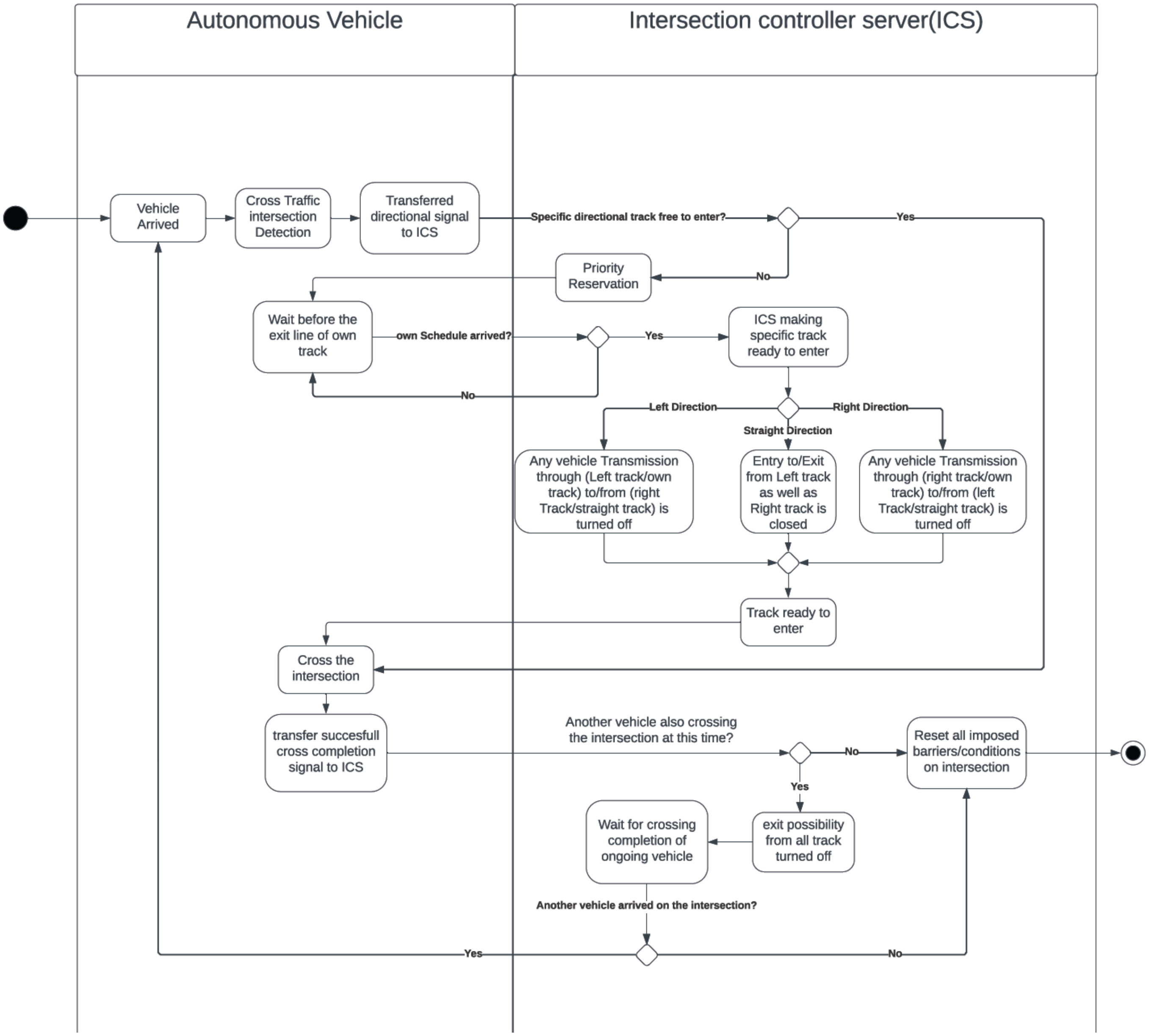

Autonomous Vehicle
Intersection controller server(ICS)
Vehicle Arrived
Cross Traffic intersection Detection
Transferred directional signal to ICS
Specific directional track free to enter?
Yes
No
Priority Reservation
Wait before the exit line of own track
own Schedule arrived?
Yes
No
ICS making specific track ready to enter
Left Direction
Straight Direction
Right Direction
Any vehicle Transmission through (Left track/own track) to/from (right Track/straight track) is turned off
Entry to/Exit from Left track as well as Right track is closed
Any vehicle Transmission through (right track/own track) to/from (left Track/straight track) is turned off
Track ready to enter
Cross the intersection
transfer succesfull cross completion signal to ICS
Another vehicle also crossing the intersection at this time?
No
Yes
Reset all imposed barriers/conditions on intersection
exit possibility from all track turned off
Wait for crossing completion of ongoing vehicle
Another vehicle arrived on the intersection?
Yes
No

Figure 11. Activity Diagram

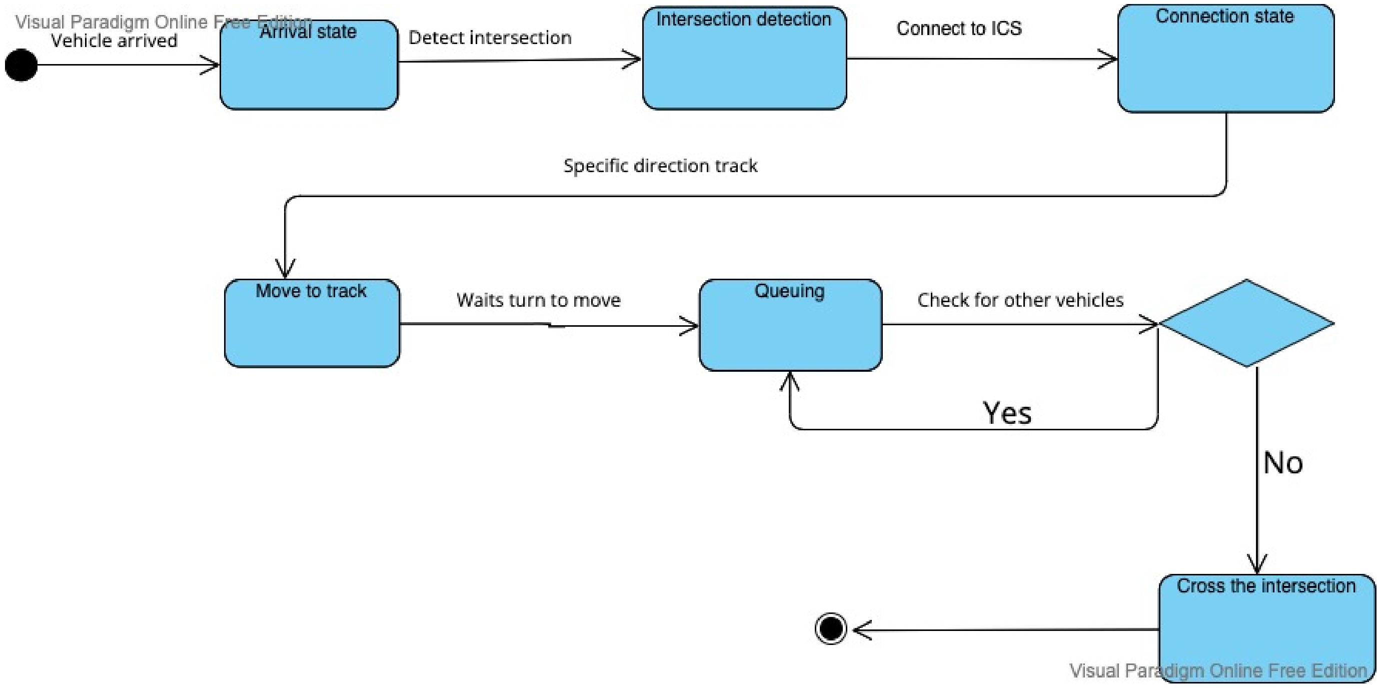


Figure 12. State Machine Diagram

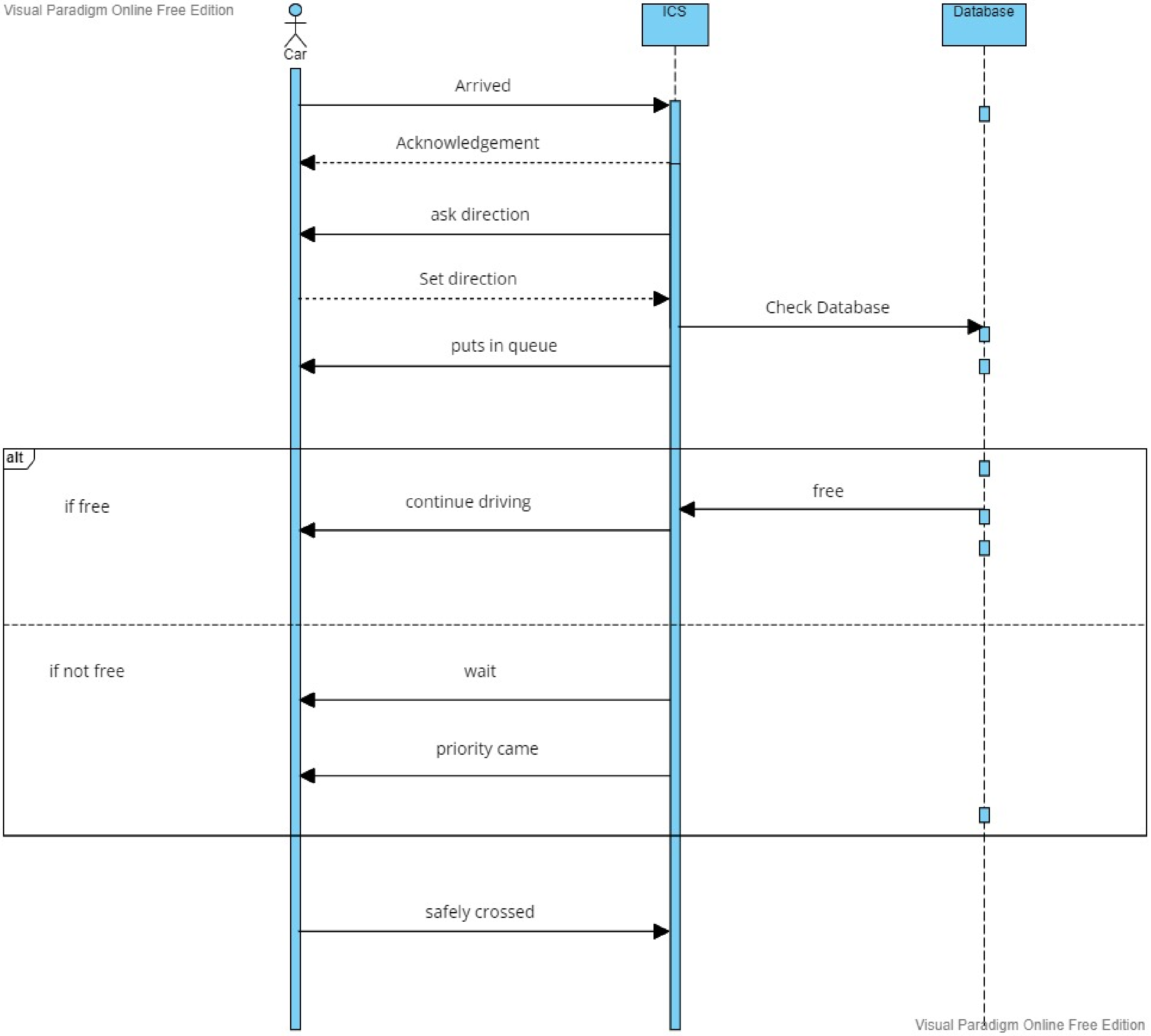


Figure 13. Sequence Diagram

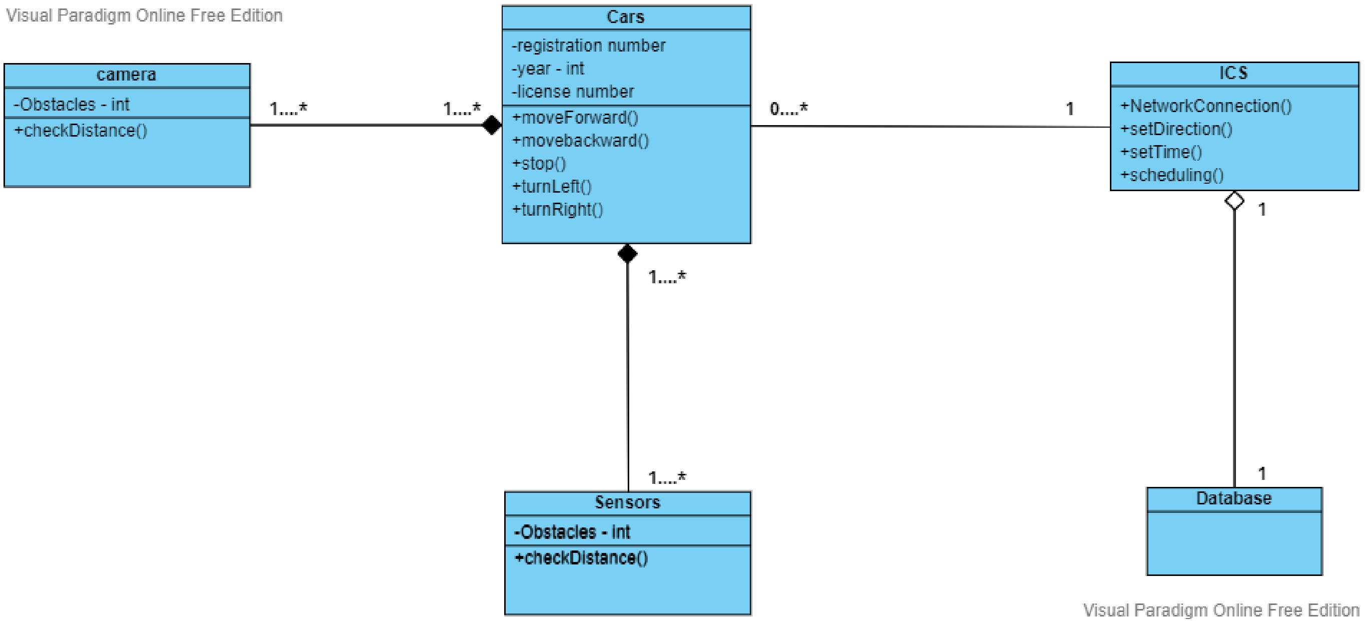


Figure 14. Class Diagram

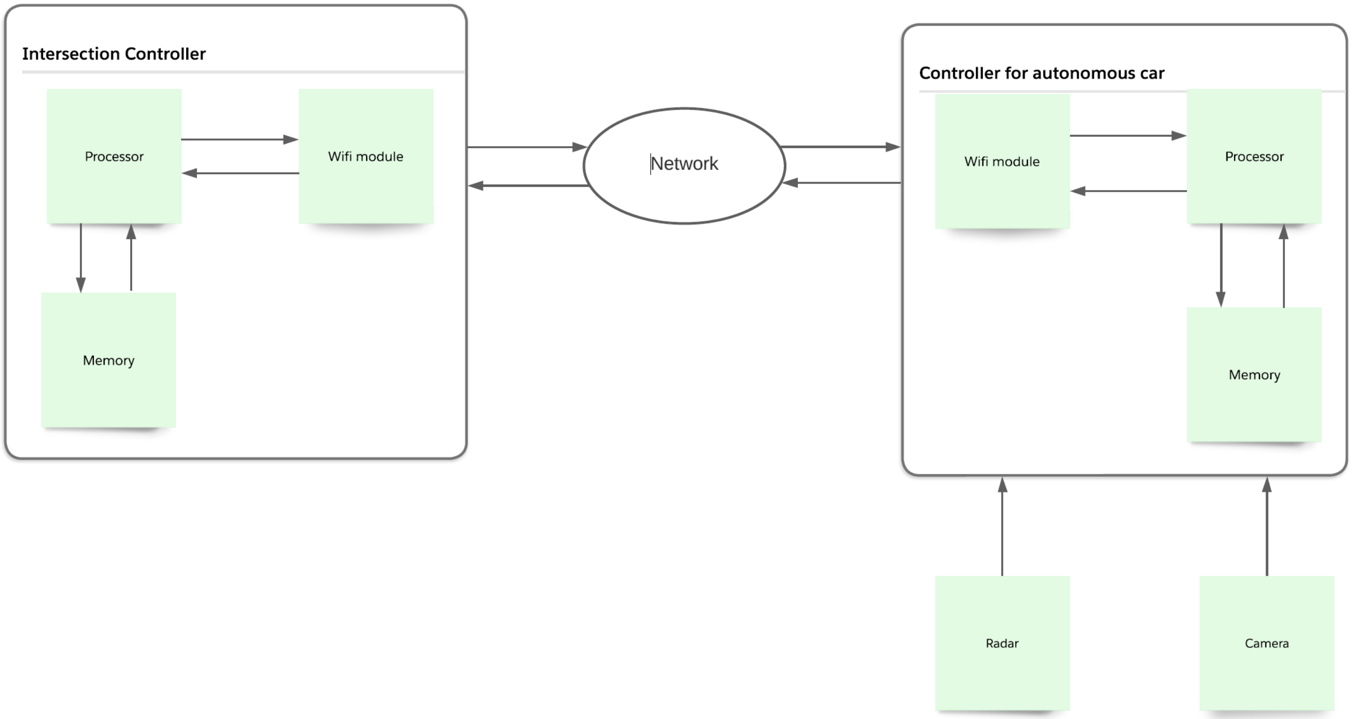


Figure 15. Overall Architecture