Smart Intersection Traffic Control

Farhad Gulizada
Matric. No:7216770
Masters in Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
Contribution Percentage: 20

Negin Amiri
Matric. No:7216441
Masters in Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
Contribution Percentage: 20

Zhao Tao
Matric. No:7216774
Masters in Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
Contribution Percentage: 20

Raghuveer Rajesh Dani
Matric. No:7216427
Masters in Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
Contribution Percentage: 20

Zaur Gurbanli
Matric. No:7216615
Masters in Embedded Systems Engineering
Fachhochschule Dortmund
Dortmund, Germany
Contribution Percentage: 20

Abstract*—The proliferation of modern vehicles has led to a significant rise in vehicle usage and congestion on roads. The surge in the popularity of autonomous cars on roads highlights the growing importance of effective traffic management. The integration of autonomous vehicles into urban environments requires a robust traffic management system to regulate the flow of traffic and ensure safe navigation through intersections. Traffic management is an integral component of autonomous networked cars, providing a framework for safe and efficient navigation of intersections. Intersections present a complex environment with a high potential for collisions and traffic management regulates the flow of vehicles to minimize risk. The system takes into account various factors such as emergency vehicles, pedestrians, and vehicle queue length to prioritize and regulate traffic flow. The implementation of robust traffic management is essential for unlocking the full potential of autonomous networked cars in urban environments.

***Index Terms*—testing, pedestrian, vehicle, microcontroller**

I. Introduction

This project presents a software-based approach to traffic management for autonomous networked vehicles in intersections. The proposed system integrates vehicle communication and sensing technologies to provide a safe and efficient traffic flow in these challenging environments. The system prioritizes the movement of emergency vehicles, pedestrians, and ordinary vehicles in a configurable manner, ensuring safe navigation and avoiding potential collisions.

The software-based solution employs a prioritization algorithm that takes into consideration various factors such as the number of vehicles in a queue, the time to reach the intersection, and the presence of emergency vehicles and pedestrians. The algorithm considers three main cases to determine the priority of each entity and regulate the flow of traffic which are further summarized in this paper.

The proposed solution demonstrates a logic-based decision-making process that prioritizes safety and efficiency in intersection traffic management. The software architecture is designed for ease of integration with existing autonomous vehicle and infrastructure systems, providing opportunities for future enhancements and scalability.

II. Literature Review

A. Traffic Flow Scenario Analysis

The project is working on how to deal with traffic management in an intersection. It is considered that the emergency cars (like ambulances, and police cars), pedestrians, and ordinary vehicles may pass the intersection, and the traffic control management system should prevent the collision. This is called the basic use case of the proposed system.

The below assumptions are considered for networked traffic control for autonomous cars:

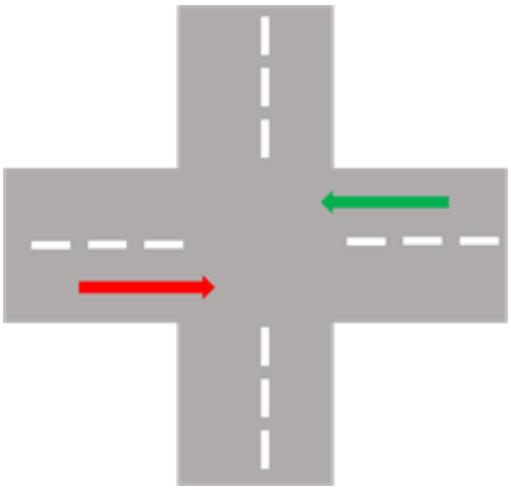
1. Each vehicle and pedestrian is equipped with a smart device to send intersection pass requests via that, receive permission to pass or not, and define the origin

and destination.

2. The traffic control system controls the traffic flow and implements the logic of decision-making.
3. The priority works like this in general:

Emergency Car>Pedestrian>Ordinary Vehicles

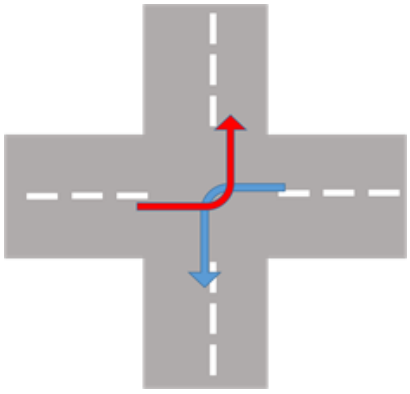
There are numerous possible concurrencies for passing the intersection. However, we focused on a few main scenarios and showed the logic that will work in general. Note that turning each car in its right direction is always feasible.



Car1/Car2	West-East
East-West	no collision

Figure 1 Traffic flow scenario 1

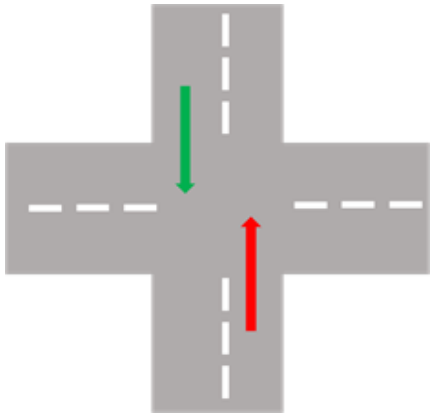
In Figure 1 the chances of collision are negligible. Both the cars, if follow lane discipline, can pass by each other without colliding with each other. Meanwhile,the pedestrian crossing lateral road is feasible



Car1/Car2	East-South
West-North	Collision–check priority

Figure 2 Traffic flow scenario 2

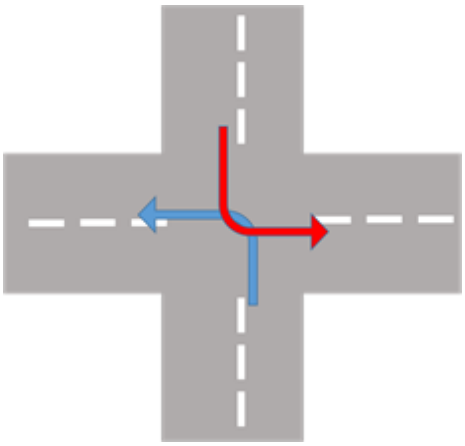
In Figure 2, it can be clearly observed that there is a high possibility of collision if proper coordination is not done in the intersecting cars. For this project, we have considered that it is a case of collision and designed a priority scheduling algorithm to manage the intersection.



Car1/Car2	North-South
South-North	no collision

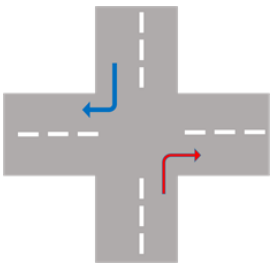
Figure 3 Traffic flow scenario 3

This case is similar to scenario 1. The chances of collision are negligible. Both the cars, if following lane discipline, can pass by each other without colliding with each other.Meanwhile,the pedestrian crossing a horizontal road is feasible



Car1/Car2	North-East
South-West	Collision–check priority

Figure 4 Traffic flow scenario 4



Car1/Car2	South-East	East-North
North-West	no collision	-
West-South	-	no collision

Figure 5 Turning right is always feasible

B. *Priority, based on the number of cars and time-to-reach*

In the intersection,for the scenario with potential of collision, the road with more cars has the highest priority in general. The prioritizing algorithm goes through three main cases.

Case 1: Less than 3 cars in a queue

- The car with less time to reach has the highest priority.

Case 2: 3 cars and more in a queue

- The more the number of cars in the queue has the highest priority.

Case 3: Equal number of cars in the queue

- The line with the lowest time-to-reach has the highest priority.

C. *Requirements*

List of the most important requirements is as below:

1. Automatically driven vehicles
2. Equipped pedestrians with a smart device for send/receive commands
3. Collision avoidance
4. The connection between cars/pedestrians and the main control system
5. Parameters to formulate and illustrate the relations
    - Time-to-reach the intersection
    - Car speed
    - Pedestrian allowed time-to-pass
    - Car type flag (emergency/ordinary car)
    - Priority values

For a better illustration, some of the diagrams are discussed in the modeling section.

III. MODELING

For modeling, Sysml is used.SysML is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametric, which is used to integrate with other engineering analysis models. It represents a subset of UML 2 with extensions needed to satisfy the requirements of the UML™ for Systems Engineering. SysML leverages the OMG XML Metadata Interchange to exchange modeling data between tools and is also intended to be compatible with the evolving ISO 10303-233 systems engineering data interchange standard.

A. *Block Diagram*

Generally the logic of the intersection traffic control system can be divided into 2 main parts which are the request handler and request executor like client and server architecture. As seen, the block diagram is divided into 2 parts as well according to the logic of the system. The handler part is responsible for handling requests from participants who may be participants, vehicles, or both. And after the handler summarizes data it dispatches it to the main part where data is processed and the final decision is made for the appropriate sequence of movement. Given that, more specifically the first part of the diagram is based on data cooperation of different sections of the system. At this point, handlers are operating for communication, pedestrians, and vehicles, especially for emergency and vehicle intensity. So handlers whose tasks are to handle the requests of pedestrians and vehicles inform the network traffic center mainly according to the course of vehicles. A communication handler is provided along with request handlers to ensure communication stability and prevent data loss between vehicles and the network traffic system.

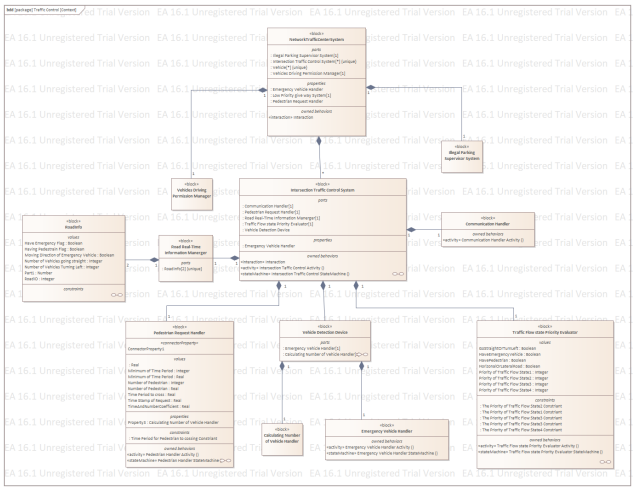


Figure 6 Block Diagram

In the second part of the diagram logic, the data coming in are handled by the network traffic system to set a priority for traffic flow depending on vehicle type whether it is an emergency vehicle or ordinary vehicle and considering the fact of a pedestrian. After the priority is identified every lane in the intersection is assigned a priority level based on which vehicles are allowed to pass in order.

B. Activity Diagram

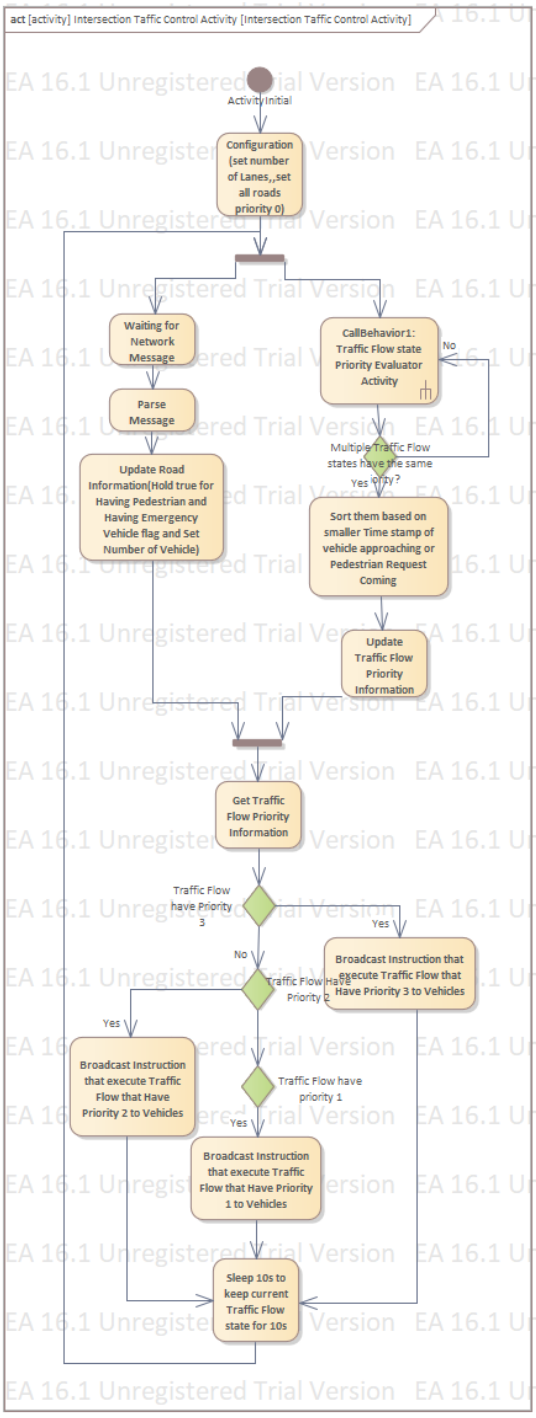


Figure 7 Activity Diagram

To achieve an efficient and smart intersection traffic control system, detecting environmental changes and rapidly reacting to changes are significant. The system has a sensor to detect vehicles(including emergency vehicles) and a related device to receive a pedestrian request. Through these methods acquire external information and then send that information to the intersection microcontroller. Meanwhile, the microcontroller asynchronously receives this environmental information and integrates these different types of information to calculate the priority for traffic flow.

Based on the above, an activity diagram to reflect the behavior of intersection control is shown in the figure. 7. To be more specific, first of all, the system simultaneously receives information and integrates information to calculate the priority of traffic flow. Then, according to the priority of traffic flow, the intersection microcontroller will consecutively execute traffic flow control from highest priority to lowest priority and there is an at least 10s time gap between different traffic flow. Microcontroller broadcasts the traffic flow state information to the vehicles approaching the intersection, so the vehicles know what movement is allowed and what is banned at the current time.

Figure. 8 below shows the process of calculating priority. Based on environmental information detected by the detective device, different traffic flows will be assigned a priority ranging from 3 to 0(3 represents the highest priority). When an emergency vehicle is detected, the priority evaluator will give priority 3 to a special traffic flow depending on the moving direction and position of the emergency vehicle. When the pedestrian handler receives a crossing request, the priority evaluator will give priority 2 to a special traffic flow depending on the position of the pedestrian. When there are no emergency vehicles and pedestrians detected, the priority evaluator will give priority 1 to a special traffic flow depending on the moving direction and position of more vehicles.

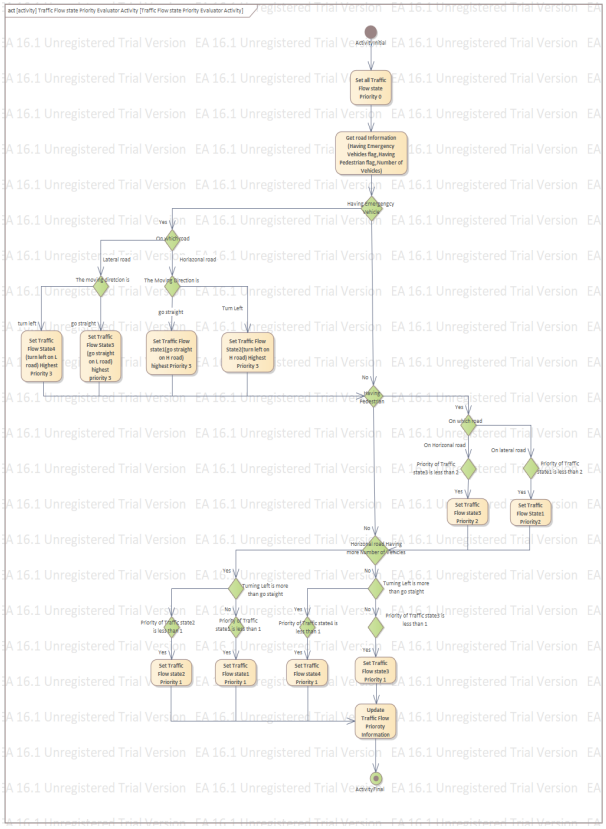


Figure 8 Priority Evaluator Activity Diagram

C. State machine

Based on analysis of traffic flow scenario, there are four different traffic flow states:

- **State1:** Going straight simultaneously in two directions on the horizontal road is allowed.
- **State2:** Turning left simultaneously in two directions on the horizontal road is allowed .
- **State3:** Going straight simultaneously in two directions on the lateral road is allowed.
- **State4:** Turning left simultaneously in two directions on the horizontal road is allowed.

There are also different actions to transit states.In the cases that an emergency vehicle with intention of going straight is detected on the horizontal road or the pedestrian request from the lateral road is received or there is no emergency vehicle and pedestrian detected and more vehicles go straight instead of turning left on the horizontal road, traffic flow will transit to state1.

In the cases that an emergency vehicle with intention of turning left is detected on the horizontal road or there is no emergency vehicle and pedestrian detected and more vehicles turn left instead of going straight on the horizontal road, traffic flow will transit to the state2.

In the cases that an emergency vehicle with intention of going straight is detected on the lateral road or the pedestrian request from the horizontal road is received or there is no emergency vehicle and pedestrian detected and more vehicles go straight instead of turning left on the lateral road, traffic flow will transit to state3.

In the cases that an emergency vehicle with intention of turning left is detected on the lateral road or there is no emergency vehicle and pedestrian detected and more vehicles turning left instead of going straight on the lateral road, traffic flow will transit to state4.

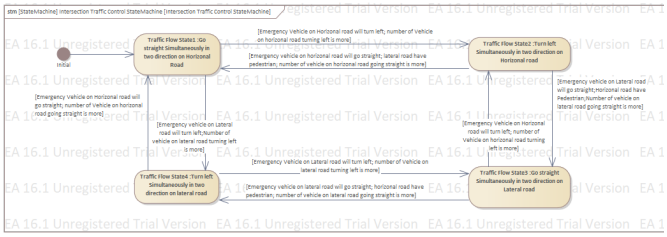


Figure 9 State Machine

D. Sequence Diagram

The sequence diagram focuses on the interaction between different parts of the system, which includes actors, control blocks, and transferred messages.

Emergency vehicles, normal vehicles, and pedestrians are the three main actors. Emergency vehicle handlers, pedestrian request handlers, road information, and the priority evaluator are the main controllers. When an emergency car is approaching, the emergency vehicle handler informs the ordinary vehicles to stop and updates the road priority

information.

When a pedestrian is coming, the pedestrian handler sends back the allowable crossing countdown to his mobile phone and updates road priority information. When an ordinary car is approaching, the same procedure is done also in that case.

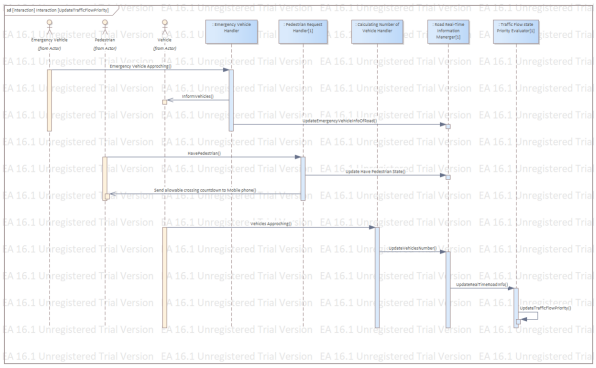


Figure 10 Sequence Diagram

E. Constraint Diagram

To more precisely map the environmental change to the system and give a reaction, a type of algorithm based on boolean calculation is designed.First of all, several variables associated with environmental change as input are defined:

- Boolean EV: HaveEmergencyVehicle(1-have,0-no).
- Boolean P: HavePedestrian(1-have,0-no).
- Boolean GoStrOrTurnL: Moving direction of Emergency or normal Vehicle(1-go straight,0-turn left).
- Boolean HOrL: situation happened in which roads(1-Horizontal road,0-Lateral road).

Output is the priority of four traffic flows:

- Priority1: priority of traffic flow1
- Priority2: priority of traffic flow2
- Priority3: priority of traffic flow3
- Priority4: priority of traffic flow4

The value range is 0-3.

Their value are given by:

$$Priority1 = 3*Ev*HorL*GoStrOrTurnL+2*!Ev*P*!HOrL+ !Ev*!P*HOrL*GoStrOrTurnL.$$

It is explained that Priority1 is equal to 3 when an emergency vehicle with intention of going straight is detected on the horizontal road,2 when the pedestrian request from the lateral road is received,1 when there is no emergency vehicle and the pedestrian is detected and more vehicles go straight instead of turning left on the horizontal road.

$$Priority2 = 3*Ev*HorL*!GoSorTurnL+ !Ev*HorL*!GoSorTurnL.$$

Priority 2 is equal to 3 when an emergency vehicle going to turn left is detected on the horizontal road,1 when there is no emergency vehicle and pedestrian detected and more vehicles turn left instead of going straight on the horizontal road.

$$Priority3 = 3*Ev*!HorL*GoSorTurnL+2*!Ev*P*HorL+ !Ev*!P*!HorL*GoSorTurnL.$$

Priority3 is equal to 3 when an emergency vehicle with intention of going straight is detected on the lateral road,2 when the pedestrian request from the horizontal road is received,1 when there is no emergency vehicle and a pedestrian is detected and more vehicles go straight instead of turning left on the lateral road.

$$Priority4 = 3*Ev*!HorL*!GoSorTurnL+ !Ev*!HorL*!GoSorTurnL.$$

Priority 4 is equal to 3 when an emergency vehicle going to turn left is detected on the lateral road,1 when there is no emergency vehicle and pedestrian detected and more vehicles turn left instead of going straight on the lateral road.

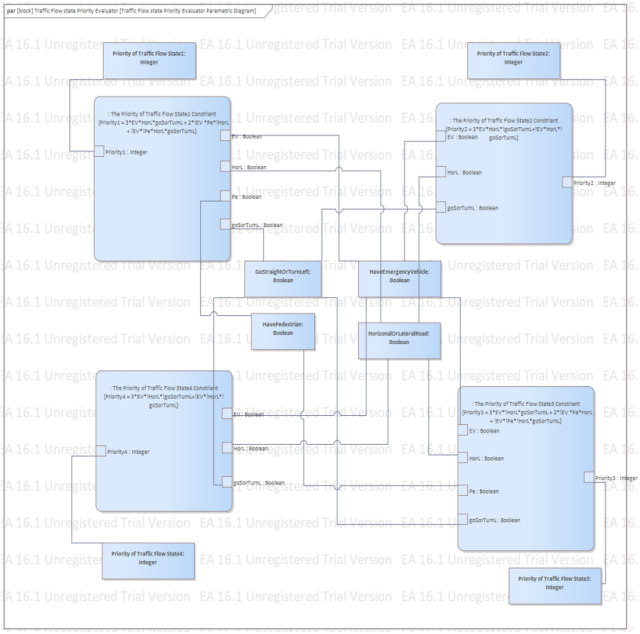


Figure 11 Parametric Constraint Diagram

IV. IMPLEMENTATION

For implementing logic, we use C++ and for visualization use Swift programming language.

A. Coding

We have a Road class which is for initializing each Road object with proper data. It contains the number of cars on the road, the remaining time to reach the intersection, whether the road has an emergency car, and the name of the road. Finally, after running the main algorithm, it gets proper priority according to all these specifications.

```
class Road{
public:
    int numberOfCars = 0;
    int remainingTime = 0;
    int hasEmergencyCar = 0;
    int priority = 0;
    string name = "";

    Road(int num, int time, int emergency, string name){
        numberOfCars = num;
        remainingTime = time;
        hasEmergencyCar = emergency;
        this->name = name;
    };
};
```

Figure 12 Road Class

After initializing roads, the main part of the code runs and estimates the priorities of each road. It starts with checking whether the road contains an emergency car or not, as we give the highest priority to that road. Since we can have several roads with emergency cars, we get indexes of all roads with emergency cars and continue our logic.

Then, we take a look at the number of cars on the road. If the value is greater than the number of lanes on the road, we consider the number of cars while prioritizing. Whichever road has the greatest number, will get high priority to cross the intersection. The last value which plays a role while giving priority is the remaining time to react to the intersection. The road with the lowest remaining time gets prioritized.

```
for (int i = 0; i < countOfRoads; i++){

    if (intersection[i]->hasEmergencyCar == 1){
        emergencyRoadIndex[getEmptyIndex(emergencyRoadIndex)] = i;
    }

    if (intersection[i]->numberOfCars >= numberOfCars){
        if (intersection[i]->numberOfCars != numberOfCars){
            for (int i = 0; i < 4; i++){
                busiestRoadIndex[i] = -1;
            }
        }
        busiestRoadIndex[getEmptyIndex(busiestRoadIndex)] = i;
        numberOfCars = intersection[i]->numberOfCars;
    }

    if (intersection[i]->remainingTime <= remainingTime){
        if (intersection[i]->remainingTime != remainingTime){
            for (int i = 0; i < 4; i++){
                nearestIndex[i] = -1;
            }
        }
        nearestIndex[getEmptyIndex(nearestIndex)] = i;
        remainingTime = intersection[i]->remainingTime;
    }
}
```

Figure 13 Prioritization Algorithm

Another crucial case to consider is that if opposite ways have subsequent priorities, there is no need to wait for one road while crossing another. So, to tackle this issue, we check the priorities of each road at the end and determine whether those types of roads have subsequent priorities.

```
void checkParallelism(Road** temp){
    int cur = temp[0]->priority;
    int size = countRoads(temp);
    for (int i = 0; i<size; i++) {
        if (i<size-1) {
            if ((findChar(temp[i]->name, "1") && findChar(temp[i+1]->name, "3")) ||
                (findChar(temp[i]->name, "3") && findChar(temp[i+1]->name, "1")) ||
                (findChar(temp[i]->name, "2") && findChar(temp[i+1]->name, "4")) ||
                (findChar(temp[i]->name, "4") && findChar(temp[i+1]->name, "2"))) {
                if (temp[i]->priority > 1) {
                    temp[i]->priority = cur;
                }
                temp[i+1]->priority = temp[i]->priority;
            }
            else{
                cur += 1;
            }
        }
        else{
            temp[i]->priority = cur;
        }
    }
}
```

Figure 14 Checking Parallelism

For better visualization, we created a desktop application in Swift programming language using the SwiftUI framework. In the first window opened we can specify the values of each road. By clicking the Initialize road button, our road object is created. It is also possible to create roads with no cars which will be eliminated while running priority algorithms.

Figure 15 Road initialization window

After the initialization of roads, and priority estimation, we get the result with car images and smooth animations. Besides that, all details of each road are indicated at the bottom of the window.

Figure 16 Priority estimation window

As can be seen from it, the road with an emergency car gets first priority, no matter how many cars are there on that road or the remaining time of them. As the third road has the highest value in the number of cars, it gets second priority. Then, the first road comes after it according to that value. As the first and third roads are parallel to each other and the cars can cross the intersection without collision, they can both get the same priority and start to move at the same time without waiting. Lastly, as the second road has the least number of cars and higher remaining time, the cars on that road cross at the end by getting the third priority.

B. Testing

Testing is a collection of methods to evaluate an application's suitability for use in accordance with a predetermined script, however, testing is not able to detect every application flaw. The basic goal of testing is to find application flaws so that they can be identified and fixed. As testing is almost the most important part of the software life cycle to successfully finish the project and build the write product we were asked to write unit and defect tests.

In this project, we were tasked with writing unit and defect tests to evaluate the application's performance. For this, we made use of the Google Testing framework for C++, which was chosen due to its ease of configuration and implementation. This framework provides a comprehensive set of tools and libraries that make it easy to write and run tests, as well as report and analyze results. The use of this testing framework was essential in ensuring that the application was thoroughly tested and that any flaws were identified and addressed in a timely manner.

Unit test

A technique for testing a unit—the smallest segment of code in a system that can be logically isolated—is unit testing. Functions, subroutines, methods, or properties are considered isolated testable objects.

Below in the picture, you can see the unit tests we designed for the project. Our goal here is to check if the functions we used in the code are working as expected. For example, in the first test case, we are testing the CharFinder function. The function looks for the char in the given string. In the second example, the RoadCounter function is tested. This function's purpose is to count roads that are not empty in the array.

```
TEST(CharFinder, UnitTest) {
    string str  = "road1";
    string el = "1";
    int result = findChar(str, el);
    EXPECT_TRUE(result);
}

TEST(RoadCounter, UnitTest) {
    Road* road1 = new Road(1, 2, 0, "road1");
    Road* EMPTY = new Road(0, 0, 0, "");

    Road* roads[4] = {EMPTY, road1, EMPTY, EMPTY};
    int expected = 1;
    int result = countRoads(roads);
    EXPECT_EQ(expected, result);
}
```

Figure 17 Unit tests

The results of the test cases are presented in the picture below. As you can, passing tests confirm that the functions written above are working properly.

```
[----------] 1 test from CharFinder
[ RUN      ] CharFinder.UnitTest
[       OK ] CharFinder.UnitTest (0 ms)
[----------] 1 test from CharFinder (7 ms total)

[----------] 1 test from RoadCounter
[ RUN      ] RoadCounter.UnitTest
[       OK ] RoadCounter.UnitTest (0 ms)
[----------] 1 test from RoadCounter (0 ms total)
```

Figure 18 Results of unit tests

Defect Test

These types of tests are similar to unit tests, but with the exemption of having negative input. In the other words, it expects the system to continue to work when negative inputs are given. The below picture demonstrates the defect tests we wrote for our project. In the first test case, the "CheckPositiveValue" function is tested by giving negative values and waiting for the system's response accordingly. For

example, while using that function we should have at least one positive value. However, if we add only negative values for testing purposes, we should receive -1. The same methodology is applied to other defect test cases.

```
TEST(CheckPositiveValue, DefectTest) {
    int sampleArray[] = { -3, -2, -1, -4 };
    int expected = -1;
    int result = checkPositive(sampleArray);
    EXPECT_EQ(expected, result);
}

TEST(CountPositive, DefectTest) {
    int sampleArray[] = { -3, -4, -5, -6 };
    int expected = 0;
    int result = countPositive(sampleArray);
    EXPECT_EQ(expected, result);
}

TEST(DefineEmptyIndex, DefectTest) {
    int sampleArray[] = { 4, 1, 2, 3 };
    int expected = 0;
    int result = getEmptyIndex(sampleArray);
    EXPECT_EQ(expected, result);
}
```

Figure 19 Defect tests

Defect testing plays an important role in ensuring that the application is robust and capable of handling unexpected scenarios. In this project, we performed thorough defect testing to verify that the system is able to handle negative input values without breaking or affecting its functionality. The results of these tests are presented in the Figure 20 below.

From the results, it is evident that we have taken the necessary precautions to ensure that the application continues to work as intended, even when faced with negative input values. This is critical in ensuring that the system remains stable and reliable, and that it is able to provide the desired functionality and performance in all conditions. Overall, the defect test results provide a comprehensive picture of the system's resilience and its ability to handle unexpected scenarios.

```
[==========] Running 5 tests from 5 test suites.
[----------] Global test environment set-up.
[----------] 1 test from CheckPositiveValue
[ RUN      ] CheckPositiveValue.DefectTest
[       OK ] CheckPositiveValue.DefectTest (0 ms)
[----------] 1 test from CheckPositiveValue (0 ms tota

[----------] 1 test from CountPositive
[ RUN      ] CountPositive.DefectTest
[       OK ] CountPositive.DefectTest (0 ms)
[----------] 1 test from CountPositive (0 ms total)

[----------] 1 test from DefineEmptyIndex
[ RUN      ] DefineEmptyIndex.DefectTest
[       OK ] DefineEmptyIndex.DefectTest (0 ms)
[----------] 1 test from DefineEmptyIndex (0 ms total)
```

Figure 20 Results of defect tests

C. *Scheduling*

Based on this smart traffic control system, there are three main periodical Tasks:

1. Receive traffic condition Message
2. Calculate traffic flow state priority
3. Execute traffic control

$$WCET_H(P) := \max_{i2Inputs} \; \max_{h2States(H)} ET_H(P, i, h)$$

Consider all possible program inputs — *Consider all possible initial states of the hardware*

As Figure shows how to calculate Worst Case Execution Time(WECT), measuring the execution time for all inputs and all hardware states is not feasible in practice. Generally, it needs approximation. To be more specific, first of all, the reconstruction of a control-flow graph from the binary, secondly, the determination of invariants for the values in registers and memory, thirdly, the determination of invariants on the control flow by determining loop bounds, identifying infeasible paths, then Determination bound on execution times of program fragments, finally, determination a worst-case path and an upper bound on the WCET.In this case, after determining the loop bound, based on the upper loop bound, using Milis() function calculate the time interval from task begin to task finish. Then, we can get an approximate WECT.

Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)
Task1_RecvMessage	Periodic	☑ Yes	0	300	-	300	101.354
Task2_CalculatePriority	Periodic	☑ Yes	0	400	-	400	51.124
Task3_executionControl	Periodic	☑ Yes	0	650	-	650	299.457

Figure 23 Task parameter table

EDF scheduling algorithm can be applied in this case. The simulation of scheduling is shown below. It is schedulable

and the utilization of the CPU is 0.9451.

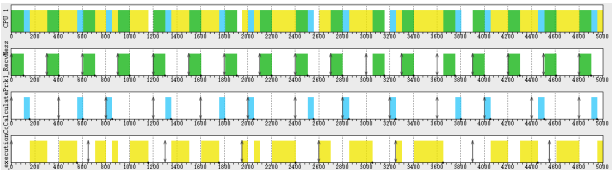


Figure 21 Scheduling simulation based on the EDF

RMS scheduling algorithm is also able to be applied in this case. The simulation of scheduling is shown below. It is schedulable and the utilization of the CPU is 0.8740.

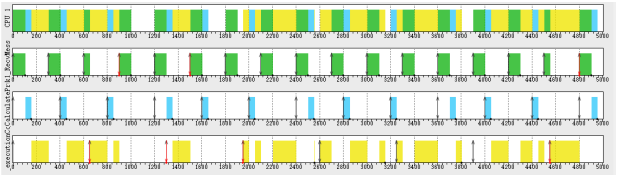


Figure 22 scheduling simulation based on RMS

Overall, EDF applied, in this case, has a higher CPU utilization than RMS, so the EDF algorithm is optimal.

V. CONCLUSION

In conclusion, the proposed system for traffic management in intersections provides a promising solution for the safe and efficient navigation of autonomous networked vehicles. The integration of vehicle communication and sensing technologies along with a configurable prioritization mechanism for emergency vehicles, pedestrians, and ordinary vehicles, ensures a robust and reliable system. The use of SysML for modeling and EDF and RMS scheduling algorithms for simulation provides a solid foundation for further development and deployment of the system. The simulation results show that the EDF algorithm is recommended for this application as it results in higher CPU utilization compared to the RMS algorithm. Overall, this project offers a valuable contribution to the field of traffic management for autonomous vehicles.

VI. ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our professor, Dr. Henkler, for his unwavering support and guidance throughout this project. His expertise and insight have been invaluable in helping us to complete this project successfully. We are truly grateful for his patience and encouragement, and for providing us with the opportunity to learn and grow as individuals. This project would not have been possible without his constant support and encouragement. Thank you, Dr. Henkler, for everything.

REFERENCES

[1] Z. Wang, Min Li, L. Tang, J. Huang and W. Shen, “Intersection Traffic Control Algorithm based on Vehicle Location Signal and its Application” International Journal of Communication Networks and Distributed Systems, January 2020.

[2] R. Ball, N. Dulay, “Enhancing Traffic Intersection Control with Intelligent Objects”, Imperial College January 2010.

[3] O. Barzilai, N. Voloch, A. Hasgall, O. Lavi Steiner and N. Ahituv, “Traffic control in a smart intersection by an algorithm with social priorities,” Contemporary Engineering Sciences, Vol. 11, 2018, no. 31, 1499 - 1511 HIKARI Ltd.

[4] A. Agrawal and R.Paulus, “Smart Intersection Design for Traffic, Pedestrian and Emergency Transit Clearance using Fuzzy Inference System”, International Journal of Advanced Computer Science and Applications, Vol. 12, No. 3, 2021.

[5] O. Grembek, A. Kurzhanskiy, A. Medury, P. Varaiya and M. Yu the University of California, Berkeley, “An Intelligent Intersection”, arXiv:1803.00471v1 [cs.CY] 25 Feb 2018.

[6] Ma Q, Zhang S and Zhou Q., “Development of a conflict-free unsignalized intersection organization method for multiple connected and autonomous vehicles”, PLoS ONE16(3): e0249170, March 30, 2021.