COLLISION PREDICTION AND SMART TRAFFIC FLOW OPTIMIZATION FOR AUTONOMOUS CARS, USING RADIO COMMUNICATIONS AND DIRECTX COMPUTE SHADERS

Cosmin – Constantin MIHAI[1], Ciprian LUPU[2]

This paper describes the implementation of a high-performance communications and control system used to coordinate car traffic in a smart intersection, so that autonomous vehicles run through the crossroads with the highest and safest possible speed while avoiding collisions. Communications between cars and the central intersection's computer is realized through radio transmissions, and the decision-making algorithms are implemented using High Level Shading Language (HLSL), a programming language used to create scripts which run on graphics processing units, powered by DirectX. The smart traffic management system provides benefits in terms of infrastructure capacity and environmental impact.

Keywords: Computational modeling, real time system control, optimization, traffic management, Kalman filter, central processing unit, smart cities

1. Introduction

Traffic in high density settlements is a difficult problem to solve, and has become a leading source of economic damage, environmental harm caused by dangerous byproducts of combustion engines, noise pollution and impact on wildlife and green spaces, and it is generally regarded as the main cause of decreasing living comfort in cities [1] [2] [3]. The primary issue at hand is solving conflicts between congruent roads. Current mainstream technology relies on driver awareness and some limited flow control systems (such as traffic lights and signs) to prevent accidents [1] [4] and ensure safety. These traffic control systems are often inflexible and will increase the time a car spends in traffic. Another problem is that human drivers inevitably fail to respect some signals [1], due to various reasons (haste, tired drivers, poor positioning of signals etc.), leading to jams and accidents.

Automated cars can fix this issue up to a point [5] [6]: they have better chances at detecting simple obstacles and avoid direct impacts, but they have no overall knowledge of what is happening in the intersections, as sensors are limited in range and computers do not have the ability to perceive the overall picture.

[1] PhD Dept. of Automatic Control and Compute Science, University POLITEHNICA of Bucharest, Romania, e-mail: cosmin.constantin.mihai@outlook.com

[2] Prof. Dept.of Automatic Control and Compute Science, University POLITEHNICA of Bucharest, Romania, e-mail: ciprian.lupu@acse.pub.ro

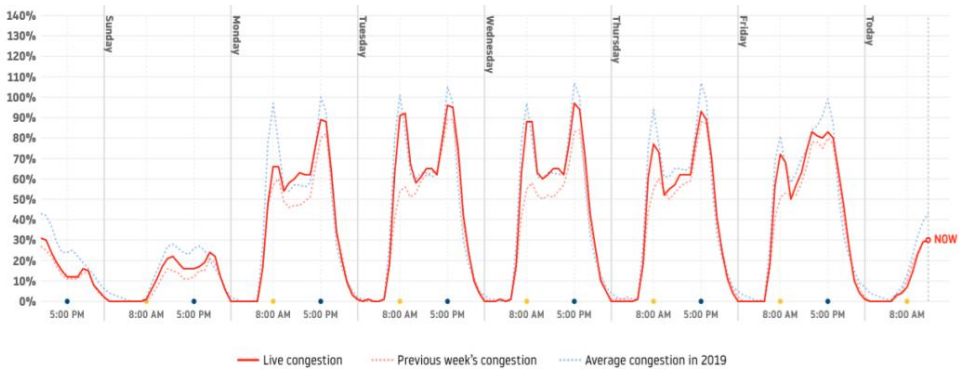

Fig. 1. Traffic index in Bucharest over one week, in 2020. Traffic peeks twice a day, in the morning and in the evening, but the daily average is still high throughout the day [2].

Therefore, centralized systems equipped with greater processing power must be used to solve the problem of visibility and predictability. State of the art literature mentions several prediction and management systems, such as systems based on reserving a slot in the intersection [5] (thus limiting the number of cars running through conflict points to only that which the infrastructure allows), prioritizing the bigger incoming group [3] [7] or distributing traffic along alternative routes [1] [3] [8]. This article proposes the implementation of a centralized traffic management system that will adjust car speeds so that flows in intersections do not require full stop of one or more lanes to prevent crashes. The novelty of the approach is the use of graphics processing units (GPU) to offload the massive calculations needed to optimize the flow management of heavy traffic. To showcase this approach, a case study will be presented in which the system achieves collision avoidance in a crossroad, allowing cars to flow freely through the intersection. Each car sends a package describing its current state and capabilities: minimum and maximum speed, current speed, compass bearing (to determine its direction) and location markers, GPS position. Using this information, the system will compute the trajectory of the cars and detect whether an impact is likely to occur. If an impact is detected, the system will act and compute a new speed reference for each car, as to avoid any impacts, and cars to flow safely through conflict points. The calculus is offloaded on the system's GPU, which is designed to scale horizontally very well [9] [10] [11], allowing the system to support increasing pressure from heavy traffic.

2. Implementation overview

The proposed solution will use a hierarchical control system approach, where decision making is decoupled from the main control system and entrusted to the supervisor system, which will be a computer equipped with a GPU capable of running compute shaders and with a radio module which will act as the

communications interface with the cars. The central computer can be positioned either in each intersection or can be configured to cover a larger area, depending on the particularities of each implementation. As road infrastructure is heavily dependent on the geographical constraints of each location, each time such a system is implemented, an analysis will be done to determine how to position each traffic supervisor. For settlements with simple infrastructure, it may be enough to have a single supervisor deal with the entire area and cover all the intersections, while in bigger cities it may be necessary to have a supervisor for each big intersection and its surrounding area. Cars flow seamlessly from one supervisor to another.

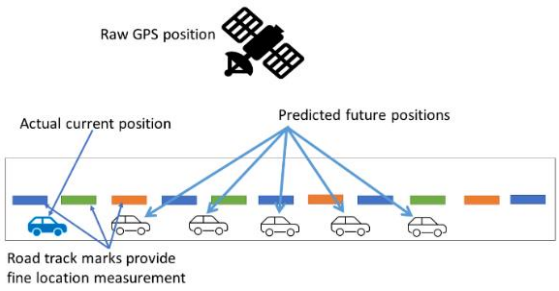

Fig. 2. Overview of the prediction system. The system will use Global Position System (GPS) and road track marks to provide an accurate measurement of the car position. The blue car is the actual current position of the vehicle, while the white shapes represent predicted future positions.

Each car will be equipped with the following devices which provide the necessary information for the system to work.

1. Speed meter: a device which will compute the current speed. This can be achieved in various ways, such as using GPS delta, or using an odometer.
2. A compass, which will provide the heading angle in the 0-360 degrees domain.
3. A roadside marker reader [6] [12], which will be used to read various markings on the road. Common vehicle to infrastructure communication protocols can be used, such as heterogeneous networks (HetNet) [13]. Cars must be able to read those markers under any weather conditions.
4. Other location services, such as mobile telephony networks and GPS.

Using the information received from each car, the system will perform the following operations to optimize traffic flow and improve safety:

1. Update the current estimated position of each car, using the last known position, reported speed, GPS location and road track marks.

2. Compute the next N future positions, given the information above.
3. Check if any of the above future positions match between cars. If there is a match, it means there is a likely collision in the future.
4. If collisions are found, find the optimal speed at which the collision is avoided.
5. Update the process state and send the new commands to each car.

Collision detection will be performed using Kalman prediction [14], which results in an approximate position of the car in relation to the reference system, and some margin of error which will represent the uncertainty in the car position. This margin will create a hit box around the estimated position of the car. An impact is considered imminent if 2 or more cars have overlapping hit boxes. A centralized system to detect future collisions has a significant advantage over on-board collision detections systems (such as Autonomous Emergency Breaking – AEB [15]) as it is able to predict collisions in situations in which AEB lacks enough information, such as when another car is coming from the side way, or when weather phenomena prevents on-board car systems from functioning properly. The system can complement car AEB and other on-board car safety systems.

3. Radio communications

Radio communications will be used to share information between the cars in traffic and the control system mainframe. Each car will relay the information described in section 2. The server-side application will perform the following operations in a loop:

- Check the health of the communications and GPU device pointers. If any of the handles have been closed due to device malfunction, recreate them. If recovery is not possible, the system will stop working.
- Read from the input streams, to get data sent by each car.
- Dispatch the information to the predictor component of the system.
- Send the results of the predictor to each car.

Cars will write their information to the on-board transmitter and will receive commands on the same stream. To handle traffic numbers that can easily reach into thousands of cars, 5G [16] will be used to transfer data between cars and the management infrastructure, using TCP/IP. The protocol will support sufficient bandwidth and low latency for real time remote control of all vehicles. As a resilience policy, if the communications stop working for a period, then the whole system will default to a lower level of automatization, involving classical control lights and signals. As the system has advanced knowledge of car positions of a few seconds, there is a grace period in which communications can recover,

which depends on each implementation particularities. When communications recover from various errors, the system will be able to come back online automatically.

4. Collision detection

Collision detection is achieved through a multi sampling technique, in which the estimated position of each car after a set amount of time is computed using DirectX compute shaders. Shaders are scripts used to program GPUs. Each sampled future position is computed in parallel to the others, taking advantage of the parallelized hardware of GPUs.

Typically, shaders run at specific moments in the image creation pipeline [17]. Compute shaders, however, can be run at any point, even independently of the graphics pipeline, and can be used to compute general purpose mathematics, in parallel on core groups [9] [11]. Each core gets assigned a thread to execute a single instruction sequence. The exact number of threads each compute shader group runs is defined at shader compilation time. The application calling the shader defines the number of groups to perform the command. In DirectX, this is achieved using the *numthreads* attribute of the compute shader entry function, and the parameters passed to the Dispatch function [18]. A Kalman predictor is used to approximate the positions of the cars in relation to a reference point after a specific time. The predictor must consider some uncertainties in the position of the car, as the exact position in a future point will be unknown at prediction time. The basic function for getting the instant position is the movement formula, defined as

$$S = vt \tag{1}$$

Where v is the speed, t is the time span, S is the distance traveled.

The general Kalman predictor works in two stages. In the first stage, it predicts the estimated state and the estimated error [14]:

$$\begin{aligned} x_k^- &= F x_{k-1}^+ + B u_{k-1} \\ P_k &= F P_{k-1}^+ F^T + Q \end{aligned} \tag{2}$$

In the second stage, the algorithm updates the state:

$$\begin{aligned} y_k &= z_k - H x_k \\ K_k &= P_k^- H^T (R + H P_k^- H^T)^{-1} \\ x_k^+ &= x_k^- + H_k y \\ P_k^+ &= (I - K_k H) P_k^- \end{aligned} \tag{3}$$

Where x is the state that needs estimation, P_k is the estimated error, k is the time moment, F is the transition matrix, B is the control input matrix applied

to the control vector u, z is the measurements vector, $H = I_6$ is the matrix of measurements, y is the residual of the measurement, K is the Kalman gain, P^+ is the updated prediction error, x^+ is the updated state.

The evolution of the state x is defined by

$$x_k = Fx_{k-1} + Bu_k + w_{k-1} \tag{4}$$

Where w is the process noise vector, with zero-mean gaussian covariance Q.

$$z_k = Hx_k + v_k \tag{5}$$

In (5), the measurement vector H is the measurement matrix, and v is the measurement vector and has the covariance matrix R.

The direction of movement is given by the compass angle from each car. Therefore, delta between 2 points can be computed using the following formulae.

$$\text{compasRadians} = \frac{\pi}{180} CompassAngle \tag{6}$$

Given this is a real time system, a strict sampling period is used. Therefore, t can be computed as multiple of the sampling period.

$$t = CN \tag{7}$$

In (7), C is a constant sampling period, and N is the number of the current sample. Combining this with (6) and (1) will result in the final formula for predicting future positions of the car in a 2D space. In most cases, 2D will be enough.

$$\begin{aligned} X_n &= S\sin(compassRadians) + X_0 \\ Y_n &= S\cos(compassRadians) + Y_o \end{aligned} \tag{8}$$

The speed of the car will be received as a parameter, and time will be a multiple of the sampling period. Each car will also have a collision (hit box) area around it in the form of a circle which represents the uncertainty in the Kalman filter. If 2 hit boxes touch or intersect each other, then a collision is detected.

To handle 3D coordinate system, we can expand the previous formulae and assume the acceleration can be defined as a 3D vector, as shown in (9).

$$x = [s(t), v(t)]' \tag{9}$$

Where:

$s(t) = [s_x(t), s_y(t), s_z(t)]'$ is the positional vector defined in 3D (latitude, longitude, and altitude – altitude is necessary to deal with advanced infrastructure such as bridges and passageways),

$v(t) = [v_x(t), v_y(t), v_z(t)]'$ is the velocity vector defined in 3D.
Therefore, the state at time moment k can be estimated knowing the state at $k-1$.

$$x_k = \begin{bmatrix} s_k \\ v_k \end{bmatrix} = \begin{bmatrix} s_{k-1} + v_{k-1}\Delta t + \frac{1}{2}a_{k-1}\Delta t^2 \\ v_{k-1} + a_{k-1}\Delta t \end{bmatrix} \quad (10)$$

Where a is the acceleration of the car. This can be rearranged as

$$x_k = \begin{bmatrix} I_3 & I_3\Delta t \\ O_3 & I_3 \end{bmatrix} x_{k-1} + \begin{bmatrix} \frac{1}{2}I_3\Delta t \\ I_3\Delta t \end{bmatrix} a_{k-1} \quad (11)$$

It can be assumed that the acceleration of the car has some measurement noise, denoted by $e_k = N(0, I_3 q_e^2)$. Applying the covariance relationship, the covariance matrix of the process noise Q can be defined as

$$Q = \begin{bmatrix} \frac{1}{4}I_3\Delta t^4 & O_3 \\ O_3 & I_3\Delta t^2 \end{bmatrix} q_e^2 \quad (12)$$

The position of the car can be estimated through any positioning systems. It will be assumed that the positioning system used will provide velocity and position measurement, which will be degraded by a measurement noise:

$$z_k = \begin{bmatrix} s_k \\ v_k \end{bmatrix} + N(0, R) \quad (13)$$

The compute dispatch will predict the car position at various time intervals, depending on the current information. At each sampling point, the position of the car will be adjusted in the algorithm, through a combination of measurements, such as global positioning systems, which will provide a rough estimation of the position and roadside markers, which combined with GPS will provide an accurate position of the car. In this use case, the following compute group indices will be used [9] [18]:

- SV_DispatchThreadID is the combined index within the entire Dispatch call. The X component will be the multiple of sampling periods. Each thread will multiply the sampling period with its group thread index to determine Δt. For example, if the sampling period is 1 second, then the thread with SV_GroupThreadID.X = 2 will compute position after 2 seconds.
- SV_GroupThreadID is the index of a thread within a group. The X component identifies cars within the input car state array. The threads within the group with SV_GroupID.X = 1 will compute the future positions of the 2nd car (indexing starts at 0).

- SV_GroupID is the index of a group. The X component determines the output index of each thread. The output array of future car position contains the flatten array of positions of every single car.

The input parameters of the compute shader will be an array of objects defining the car state. The output will be the predicted positions of the cars. The 2 arrays will be shared between all thread groups. The data will be in 1D format, Y and Z will always be 1, both for Dispatch call and for *numthreads*.

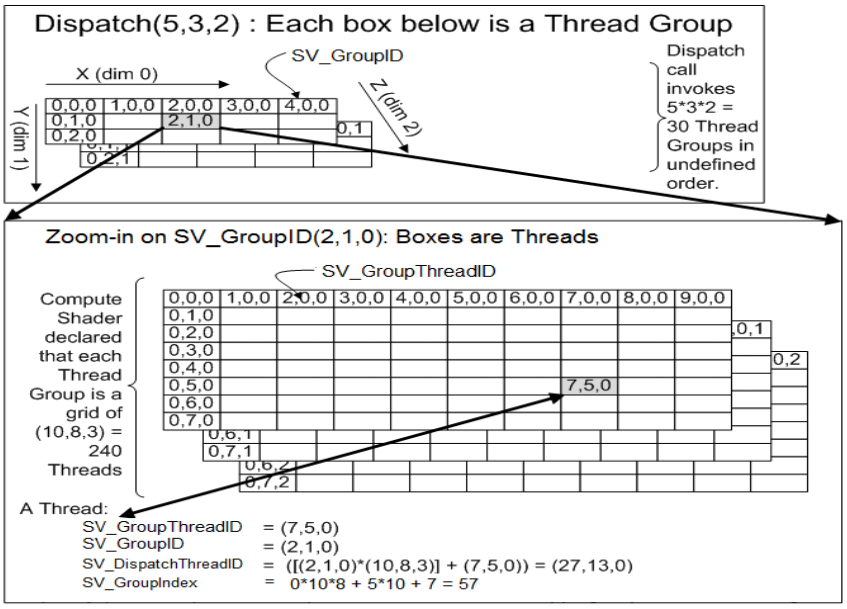

Fig. 3. The relation between the number of threads in a group defined by a compute shader and the number of groups dispatched by the application [18]. Each group and thread can be identified by a 3D vector, with X, Y and Z components.

The output array will contain the centers of the hit boxes used to determine collisions. The collision detection itself will use a classical formula for determining if 2 circles overlap or touch and match the same timestamp.

$$d \leq r_1 + r_2 \tag{14}$$

Where d is the distance between the centers of the circles, and r_1 is the radius of the first circle and r_2 is the radius of the second circle. The uncertainty in the Kalman filter can be estimated by increasing the radius (and therefore the hit box) of each car. The uncertainty is kept low through the usage of the roadside trackers. If these become unavailable, uncertainty will increase. In such cases, the system will not be able to accurately predict impacts and should default to a more conservative approach to traffic management, like how current traffic signals work. Mathematically, the prediction function can be described as shown in (15)

$$C(S_n) = P_{n*m} \tag{15}$$

Where:
S_n is a vector containing the metadata of each car (id, bearing, current speed, location)
n is the number cars.
P_{n*m} is the output matrix containing the next m instant locations.
The impact detection function can be described as

$$L(P_{n*m}) = \begin{cases} 0 & when\ no\ impacts\ found \\ 1 & when\ 1\ or\ more\ impacts \end{cases} \tag{16}$$

Speed adjustments will be determined using a constrained optimization technique. The constraint function will be the very same function which predicts impacts. In this case study, a Monte Carlo technique was used [19]. The goal of the optimization is to find the best speeds at which the vehicles will not collide, while avoiding full stops as much as possible.

$$\begin{aligned} &P_{n*m}{}^{*} = \max\left[C(Id_n, B_n, S_n^{*}, L_n)\right] \\ &L(P_{n*m}{}^{*}) = 0 \end{aligned} \tag{17}$$

As shown in (17) at each iterative pass in the optimization algorithm, the impact detection function will be applied to the computed reference speeds for each car. If no impacts are detected, then the computed reference speeds will be dispatched to each car. Additional constraints can be added to support speed limits and other legal restrictions on the road segment. In some situations, it may be impossible to prevent crashes without stopping a car. In such cases, the system will try to minimize the number of cars that get a full stop. The cars that entered the supervised area first will be given priority and a higher speed.

5. Case study

The system was tested using two experimental cars and a track as shown in Fig. 4. The application was developed in such way that it allows both controlling the physical cars and simulate the flow in a virtual model. It was developed in C++ as it interacts naturally with DirectX. The compute shaders have been developed in the native DirectX HLSL. The development environment is Visual Studio 2019, using the latest C++ toolchains. MATLAB and Excel have been used to plot figures and double check the results. At startup, the application initializes the DirectX device pointer and its context, compiles the shader code from source file and allocated data buffers to exchange data with the shader. Then the following instructions are executed in a loop:

- Get the data and write it to the buffers. Create data views for inspecting the results.

- Dispatch the compute shader.
- Dispatch results to cars.

The test cars, shown in Fig. 4, are based on Arduino and are equipped with several sensors which gather all the necessary data for the experiments: speed, bearing, positioning. They are also capable of following the road track markers, behaving like line follower robots simulating how an autonomous vehicle would behave in a real-life scenario.

Fig. 4. Test car based on Arduino (left), contains sensors for detecting track markers, radio communications module for interacting with the system, and a compass for determining movement bearing. On the right, example of track markers.

The 2 scenarios are represented in Fig. 5. In the safe scenario, shown in Fig. 5 left side, the two cars have intersecting paths, but the distance between them stays well above the danger limit. At any sampling time, they are far enough from each other. Fig. 6 shows the distance between the two cars over time, as computed by the simulator stage.

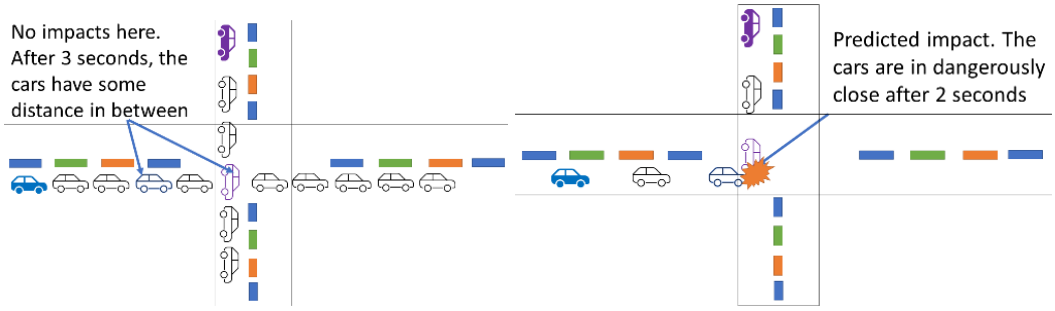

Fig. 5. On the left side, simulation of cars with intersecting trajectory but no collision. When the purple car does get in front of the blue car, the blue car is yet to enter the intersection, as shown with the colored highlighted predicted positions. On the right side, the simulation example of how two cars would collide in the middle of the crossroads. Those figures illustrate how the system perceives the movement of cars, by sampling their trajectory at specific time intervals, how it detects possible collisions. Each hollow car shape represents the sampled position of a car at after a time interval.

This can be verified with a simple MATLAB script, plotting the distance between the 2 cars, as seen in the figure bellow. The system correctly detects distance between the 2 cars as safe, and impacts are highly unlikely. In this situation, the system will take no action since the cars can safely pass through the conflict area of the intersection with no risk.

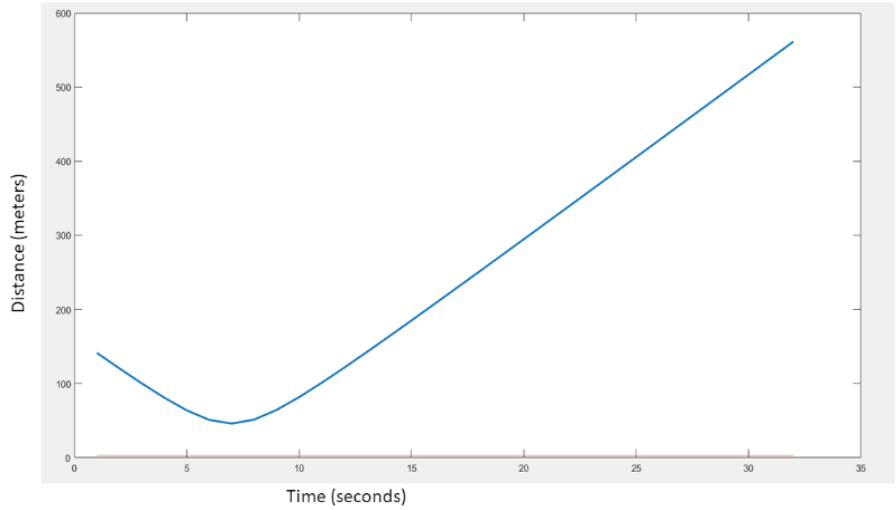

Fig. 6. The distance over time between the 2 cars in the "no impact" scenario. While the cars get close, the distance does not drop below 50m, well above the orange limit of 1m.

In the second test scenario shown in Fig. 5 right side, the 2 cars will cross path and will come in dangerously close to each other. The difference between the 2 scenarios is the speed. In the first scenario, the cars travel at different speeds, so they never pass close to each other. In the second scenario, the cars run at the same speed, and will therefore crash into each other after about 6 seconds, as the projected distance will be well within the crash margin, at about 1m. In the second scenario the system will have to adjust the speed of the of the cars to avoid impact. The trajectory of the cars is the same as in the first case.

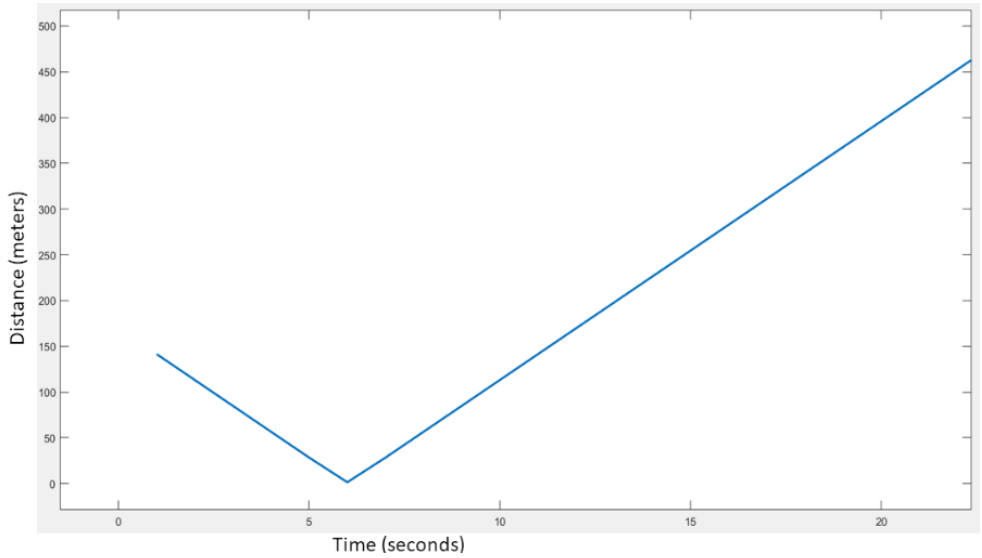

Fig. 7. The distance over time between 2 cars that are set to collide after 6 seconds. As the distance drops towards 0, an impact is imminent.

As seen in Fig. 7, the system correctly detects the possible impact between the 2 cars. The system will need to act and compute new speed references for the cars. The function L presented in (17) will return 1, and the system will have to compute new reference speeds for both cars. The system can complement traditional car on-board AEB implementations, as it has information about the traffic, position and direction of other cars which would now be available to on-board car systems, thus increasing the efficiency of existing safety features.

As it can optimize speed to avoid impacts, the system allows existing infrastructure to support heavier traffic. Since cars will require to cold start less often, this will reduce pollution, having a positive impact on the environment [1].

In comparison to a classical signaling system for a crossroads junction with 1 lane for each direction, the system allows up to 4 cars in the intersection, while classical traffic management systems, such as a red light, can only allow 2 cars at maximum. This is achieved by not stopping the cars with intersecting paths, but by adjusting speeds so that they narrowly pass (but not crash) by each other.

6. Conclusions

This paper presented research and results on a novel approach to traffic management for automated cars, using a combination of various on-board car sensors and a traffic supervisor equipped with a graphics processing unit to predict the movement of cars and avoid impacts when going through intersections, by computing the future positions of each car using a Kalman filter.

The novelty of this research was the use of graphics processing units, compute shaders and DirectX to perform the calculations needed to detect collisions and perform speed optimizations. This way, the system scales horizontally with the numbers of cars in traffic, a single computer system can handle thousands of cars, as GPUs are designed to perform mathematical operations through hardware accelerated parallel computations.

5G will be used to handle the data transmission between the traffic supervisor and each car. The technology allows low latencies and appropriate bandwidth to handle data transfers between the supervisor and thousands of cars in traffic. Low uncertainty in the Kalman filter is achieved through fine grained positioning systems, combining GPS and roadside car trackers. Fallbacks to lower levels of signaling automation can be used in case of system malfunctions, communications outages, or bad weather. Although the case study has only captured the situation of a single lane crossroad intersection going in straight line at 90 degrees angle, the calculations can be adapted to suit any kind of intersection and account for turns, complex infrastructure such as bridges and passageways.

By increasing the capacity traffic capacity of existing infrastructure, the proposed system has a positive impact on the environment and helps increase living conditions and standards in crowded metropolitan areas.

Acknowledgment

The work has been funded by the Operational Programme Human Capital of the Ministry of European Funds through the Financial Agreement 51675/09.07.2019, SMIS code 125125.

R E F E R E N C E S

[1]. *R. W. Caves*, "Encyclopedia of the City", 2004.
[2]. *TomTom*, "Bucharest traffic", TomTom, Bucharest, 2020.
[3]. *U.S Department of Transportation*, "Congestion: A National Issue", 2008.
[4]. *European Commision*, "Road Safety statistics", 18 09 2020. [Online]. Available: https://ec.europa.eu/transport/media/news/2020-06-11-road-safety-statistics-2019_en.
[5]. *K. Dresner and P. Stone*, "Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism", 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004.
[6]. *F. Secuianu, C. Mihai, A. Vulpe and C. Lupu*, "Implementation of an autonomous mobile platform based on computer vision", 18th International Carpathian Control Conference (ICCC), 2017.
[7]. *V. Constantinescu and M. Pătrașcu*, "PERFORMANCE ANALYSIS OF GENETIC ALGORITHMSFOR ROUTE COMPUTATION APPLIED TOEMERGENCY VEHICLESIN UNCERTAIN TRAFFIC", U.P.B. Sci. Bull., Series C, 2020.

[8]. *S. Irani and V. J. Leung*, “Scheduling with Conflicts, and Applications to Traffic Signal Control”, Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 1996.
[9]. *Microsoft Corporation*, “Compute Shader Overview”, 31 05 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d11/direct3d-11-advanced-stages-compute-shader.
[10]. *nVidia*, “NVIDIA TURING GPU ARCHITECTURE”, 2019.
[11]. *C.-C. Mihai, C. Lupu, D. Secuianu and A. Vulpe*, “Implementing high performance system simulators using modern graphics rendering devices: Implementing system control algorithms on graphics hardware”, 14th International Conference on Engineering of Modern Electric Systems (EMES), 2017.
[12]. *X. LIANG, M. SHEN, G. D. and G. CHEN*, “REAL-TIME MOVING TARGET TRACKING ALGORITHM OF UAV/UGV HETEROGENEOUS COLLABORATIVE SYSTEM IN COMPLEX BACKGROUND”, U.P.B. Sci. Bull., Series C, 2019.
[13]. *S. P.K., S. R., N. S.K., G. K.Z. and N. S*, “Seamless V2I Communication in HetNet: State-of-the-Art and Future Research Directions in Connected Vehicles in the Internet of Things”, pp. 37-83.
[14]. *D. Popescu, A. Gharbki, D. Stefanoiu and P. Borne*, “Process Control Design for Industrial Applications”, 2017.
[15]. *Euro NCAP*, “AEB Car-to-Car” Euro NCAP, 2020.
[16]. *5G PPP Architecture Working Group*,"View on 5G Architecture 2016.
[17]. *Microsoft Corporation*, "Pipeline Stages (Direct3D 10)”, 31 05 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-pipeline-stages.
[18]. *Microsoft*, “D3D11DeviceContext: Dispatch method”, 01 09 2020. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/d3d11/nf-d3d11-id3d11devicecontext-dispatch.
[19]. *R. Cools and D. Nuyens*, “Monte Carlo and Quasi-Monte Carlo Methods”, Leuven, Belgium, 2014.