Design and implementation of destination tracking and velocity control as an addition to a boids-inspired traffic simulation

Thijmen Limbeek
University of Twente
EEMCS faculty
Enschede, the Netherlands

***Abstract*—With the ever-growing number of personal vehicles sold, our roads will inevitably get to a point where they can not smoothly accommodate all individual participants in traffic. A possible result is an increase in traffic jams, accidents, chaos and stress on the roads.**
The common denominator in the problems that cause slow-down of traffic flow is often the unpredictability of human input. Instead of adapting the roads to the drivers, an alternative solution to aid in vehicle throughput is automated vehicle platooning. There are of course many different ways to implement automated vehicle platooning, but the focus of this paper lies on an adaptation of the boids model inspired by the work of C. Reynolds [1]. This paper is an extension of previous work, where a simulation environment was created using openGL compute shaders that run in parallel on a GPU for increased performance [2]. This simulation environment will extend the used control algorithm by including vehicle speed adaptation and making vehicles follow a path to their destination. The addition of destination tracking and velocity matching was found to have had a contribution to preventing crashes in situations where they were previously unavoidable and allows vehicles to track a path towards a destination sufficiently smoothly.

I. Introduction

Anyone who has driven a car in the last few years during rush hour can tell you that our roads are busy. Since personal vehicles are becoming more and more abundant, as well as larger every year, it is safe to say the roads will get even busier and more congested. Eventually, we will get to a point where our current roads are unable to support the number of drivers in a smooth way, which has cascading effects to the entire road network. Views of whether or not roads should be made bigger to accommodate drivers is a topic with many differing opinions, but ultimately, it is not the best solution because increasing the number of lanes or the size of the roads is inefficient use of space, makes places less habitable for people and ecosystems and costs unrealistic amounts of money and time.
So instead of focusing on giving room to human error, the issue can be approached from a technological point of view. Automated driving is a topic that can be implemented in many possible ways. Cooperative vehicle platooning is an interesting concept that allows vehicles to form very tight-knit clusters due to advanced automated vehicle perception and closely coordinated communication with each other, reducing congestion. It also has the potential benefit of lower energy consumption and emissions [3].

In essence, vehicle platooning is similar to flocking behaviour found in nature: a group of entities travels together in a cluster to reach a common goal without colliding into each other. In the 1980s, C. Reynolds came up with a set of rules that describe this flocking behavior for use in computer simulation [1]:

- *Cohesion*: striving towards the centre of the group
- *Alignment*: attempting to match the group's velocity
- *Separation*: collision avoidance, maintaining appropriate distance

The intention of Reynold's publication was to provide rules for flocking, and did not include any algorithm proposals. In publication [4], three approaches to an algorithm are described that could be used for flocking in confined space, such as roads. The paper makes a distinction between three types of agents: α agents, which describe the moving flocking agents themselves, β agents, which describe the effect of obstacles (walls, road debris, etc.) and γ agents, describing "group objectives" such as migration of a group towards a destination for example.
The classic form of the boids algorithm is a reactive process that does not rely on anticipation or prediction. Publications [5] and [6] describe more complex form of cooperative vehicle clustering that make heavy use of inter-vehicle communication/intention messaging alongside the vehicle's own "senses". This is a concept that could be explored further in the context of boids-inspired autonomous vehicle simulation as well.

Previous research was conducted to simulate automated vehicle platooning using the boids model proposed by C. Reynolds. Where the boids model typically describes flock dynamics in free space (either three or two dimensional), this simulation environment adapted it to work in a two

dimensional confined space. Next to this, the calculations of this simulation are performed using openGL compute shaders that make use of the parallel computation that GPUs can perform, resulting in fast execution times [2]. The simulation environment does model roads, although relatively basic "uninterrupted" ones. It does not support more realistic situations such as highway exits or junctions to name a few.

The focus of this paper lies on designing and implementing a destination tracking system and a basic speed adaptation system based on autonomous decisions, as well as supporting modifications to the previous simulation environment in an attempt to make it more capable. Future work might include road network simulations, path-finding for destinations, or intention-messaging to other vehicles.

The main question that this paper will answer is the following:

- *How can the addition of destination-based navigation and velocity control to a flocking-based autonomous vehicle test environment be designed in a way to ensure safety and stability?*

In order to find an answer to this question, the following sub-questions will be evaluated:

- *What modifications are required to the simulation environment for destination-based navigation and velocity control?*
- *What kind of experiments are required to measure the performance of destination based navigation and velocity control?*

In section II-A, the reader will be introduced to a few key concepts that help in understanding the subject matter. Then, the improvements and additions to the simulation that are required for destination tracking and velocity adaptation to work will be discussed in section II-B. After this, the concepts behind the chosen methods for destination tracking and velocity adaptation will be explained in sections II-C and II-D respectively. The methods for validating and improving the system will be highlighted in III-A and III-B, and the results will be shown and discussed in sections III-D and III-C.

II. Design

In the publication for the previous iteration of the simulation environment [2], it was found that the algorithm implementation works well for simpler traffic situations, but in some instances, crashes occurred that could be avoided with active braking. The purpose of the previous research was to investigate whether a form of boids algorithm could even function at all in tight spatial constraints such as roads. This is why the test setups consisted of uninterrupted stretches of roads without more intricate traffic scenarios like junctions or exits. The goal of this paper is to provide the groundwork for a more realistic traffic situation by including the ability to set a destination for a vehicle and velocity control. The following section will inform the reader about prerequisite terminology and theory. It will also discuss improvements and supporting modifications to the previous iteration of the simulation environment.

A. Key concepts and prerequisite knowledge

C. Reynolds described a set of behavioral rules for computer simulation of herds, schools or flocks that when applied is typically called a "boids" algoritm. The term boid comes from bird like objects, or bird-oids and describe individual entities moving in a group. The following section will describe the behavioral rules as implemented in [2]. The first of these behavioral rules is cohesion, which is the group's tendency to steer towards the centre of the group. In essence, each individual boid will experience a force towards the approximate centre of the group. This force is proportional to the distance to the centre: the further away, the higher the force. Typically, a "detection range" is implemented for the cohesion force such that multiple flocking groups can exist in one space without forming a single group comprised of all vehicles in space. The cohesion factor can be mathematically described by equation 1, in which $\vec{q}$ represents the position of another vehicle within the cohesion detection range, and $\vec{p}$ represents the position of the "current" vehicle that is calculating the cohesion contribution.

$$\vec{f_c} = \vec{q} - \vec{p} \tag{1}$$

The second behavioral rule is alignment, which causes vehicles to attempt to match their velocity to that of the group. In this context, when the term velocity is used, it is meant to represent a vector that implies both direction and speed, so the alignment factor is also responsible for steering with the group and along with road. Equation 2 represents the alignment factor mathematically, where $\vec{v_i}$ is the velocity vector of another vehiclethat is being checked, and $\vec{v_j'}$ is that same vehicle's requested velocity from the previous time step.

$$\vec{f_a} = \frac{\vec{v_j} + \vec{v_j'}}{2} \tag{2}$$

The last of the three core rules is the separation factor, which can best be described by working on a "steer to avoid" principle. In the context of autonomous vehicles, this means that a hypothetical vehicle uses its sensors to be aware of its surroundings and avoids obstacles on the road by applying a force away from them when needed. Mathematically, the separation factor is described by equation 3, in which r represents the desired separation distance, d represents the inter-vehicle distance for which to calculate the separation component, and $\vec{q}$ and $\vec{p}$ are the same vehicle locations as in the cohesion factor.

$$\vec{f_s} = -e^{r-d} \cdot \frac{\vec{q} - \vec{p}}{|\vec{q} - \vec{p}|} \tag{3}$$

The resulting velocity vector is the sum of total separation, alignment and cohesion contributions times their respective weights.

Other terms used for simulation entities are (α) agents and actors for the vehicles or animals, and β agents for obstacles, such as walls or hindrances on the road.

The simulation environment makes use of openGL compute shaders. The enviroment was created due to the fact that most traffic simulators typically take a long time to perform a simulation, making it difficult to perform parameter sweeps or gather data in a time-efficient way. A compute shader is a script/set of instructions to be run on a GPU instead of a computer's CPU. It allows for parallel deployment of a collection of calculations at a much larger scale to what a typical CPU can handle in terms of parallelism. Variable storage is saved on a memory buffer, which is required to be pre-allocated. To put the performance into perspective, when v-sync in the simulation was turned off, meaning the framerate was uncapped, a single pass of the algorithm of about 1300 frames at a timestep of 0.1 seconds per frame takes about the same time as extracting the data collected in the pass and plotting results.

The system will be designed based on a model of a 2019 Toyota Corolla LE, with a length of 4.9 meters and a width of 1.8 meters. The system evaluation will take place at a speed of $2m/s$ or $7.2km/h$ for consistency with the previous iteration of the simulation environment.

B. Supporting modifications and improvements

1) Improved Collision Detection: Collision detection is an important part of a traffic simulation environment, as it can serve as a metric to immediately rule out certain strategies (both parameters and additions to the algorithm) during the design and optimization phase. While familiarizing with the simulation framework, it became apparent that the collision detection method works, but it is not as accurate. The implementation assumes that every vehicle's hit detection shape is a circle with a fixed radius, which is a typical occurence in boid simulations. If the centre point of another vehicle comes within this radius, a flag is enabled in the buffer that the vehicles have collided. This is of course a simplified view of reality, but it has the advantage that very few calculations are required.
The problem is that sometimes a crash is flagged while there is still ample space between vehicles. The reason for this, is that this radius has to assume the worst case scenario, which is when two vehicles make contact with each other at their corners in a rear-end collision.

The simple crash detection method that is used in the previous iteration of the simulation environment was chosen for a reason: it is simple to implement, requires very little computation power and works for any kind of polygon. This works fine for simpler traffic situations, but with the addition of destinations and speed adaptation, the simulation starts to more closely resemble a realistic traffic situation. Since the vehicle flock will become more dynamic, using a circle as a bounding box will be too inaccurate because this will likely report more false-positive crashes.

In simulation and game development, there are two popular ways that can be applied in the context of boid simulation for realizing collision detection: Axis-aligned bounding boxes and the separating axis theorem. Axis aligned bounding boxes are a middle ground between distance based collision detection and the separating axis theorem in terms of required computation power. In some situations, it can be more accurate than distance based collision detection, but it has the significant drawback of requiring the bounding boxes of the vehicles to be perpendicular to the x and y axis [7]. Essentially, this would mean that it would only serve as a better crash detection alternative on stretches of road that are parallel to either the x or y axis.

The Separating Axis Theorem (SAT) is a more viable alternative that allows accurate collision detection under any angle, and is the preferred choice in the context of this paper. This theorem requires that the shapes to be used in crash detection are convex: from any point in the shape, you can draw a line to any other point without going outside of the shape. Since the vehicles are modelled as rectangles, their shapes are convex, and the SAT can be implemented. Like mentioned before, while the SAT is the most accurate out of the previously discussed collision detection algorithms, it comes at the highest computational cost.
The first step in SAT crash detection is to calculate the normal for every edge of the shape. Figure 1 displays a schematic drawing for a situation where the SAT will report no collision. For visualization purposes, one can draw a line parallel to the normal, which is the blue line at the bottom of figure 1. On this parallel line, the minimum and maximum coordinates of the vehicles are projected with respect to the blue line. These minima and maxima are compared, and if the "Line segments" between the minimum and maximum of each shape overlap, it means there is no separating axis can be drawn between these two lines on this normal projection. The black arrow at the bottom of figure 1 shows that there is no overlap between the vehicle's minima and maxima, indicating that a separating axis can be drawn, so there is no collision. These steps are usually repeated for all normal projections, and if it is possible to draw a separating axis on a normal projection, the vehicles are not colliding. In this context this is slightly different, since the vehicles' shapes are symmetrical both length-wise and width-wise, meaning that only two normal projections are necessary as long as they are adjacent, because the other two will be the same as their respective edge on the opposing side of the vehicle. This is why the separating axis theorem is prefered over AABBs. In this context, only 2 axes have to be checked, which means that the difference in computations required is not very big while the performance gain is significant.

The advantage of the separating axis theorem is that it can be very accurate in whether a crash has happened or not. Next to this, it offers the possibility to calculate a more accurate distance between vehicles as opposed to centre point to centre point: it makes it possible to calculate the minium distance between the vehicles.

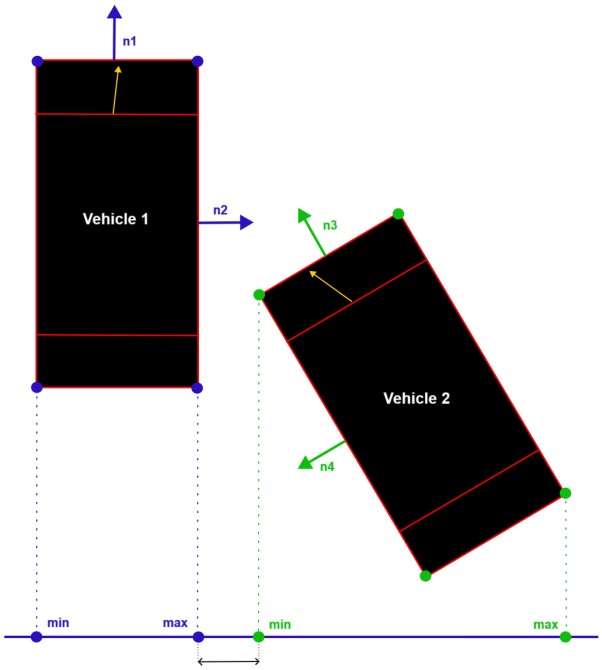

Fig. 1. Illustration of SAT collision detection

2) Cohesion: Typically, the boids algorithm is implemented in free-space, possibly with obstacles. Since a traffic environment is a confined space, the importance of the cohesion factor can be questioned: the edges of the road and the other vehicles on the road keep a vehicle in place, so a driving force to keep with the group might be redundant, or even counter-productive in some situations. Either the cohesion should be turned off temporarily for vehicles attempting to leave the group, or it should be turned off/kept to a low importance weight. Since the provided code base as a results of the research in [2] assigns a very low weight to cohesion already with no apparent drawbacks in vehicles' close proximity to each other, the cohesion factor will be left enabled in the algorithm code with the option to quickly disable it if needed in the future.

C. Destinations

In order to design and implement destination tracking for the simulation, a system has to be created that lends itself well to GPU computing. There are many ways of destination tracking that could be used, but in essence, a driving force towards the final destination should be exerted along the vehicle's path. An interesting option is to implement Bézier curve path generation for smooth continuous track planning towards the final destination [8]. While this will create an ideal path for the vehicle to follow, it is computationally expensive in case the route has to be recalculated, and can take up significant amounts of buffer space. Another option is vector field navigation, where a vector field map is created that guides a vehicle towards it final destination. This approach can provide dynamic, smooth motion towards a vehicle's final destination, but is difficult to adapt at larger scale. Since each vehicle requires its own vector field map, an excessive amount of buffer space will be taken up, slowing down the simulation and requiring high amounts of GPU memory to function at all in full scale traffic situations.

The chosen destination tracking approach is a form of waypoint navigation, that makes use of a list of pre-defined location markers. A system like this is simple, does not take up large amounts of buffer space and is not computationally expensive. These properties are ideal for implementation in a compute shader. The most notable drawback for this form of navigation is that it makes use of larger discrete steps between location markers, which means that the navigation will likely not be as smooth as the aforementioned alternatives.

The location markers are tracked using a mixed version of the separation factor and γ agent as described in [4]. The separation factor is a desired velocity component that varies in strength with distance to the vehicle flock. The driving force towards the location markers should be constant however, such that vehicles do not experience inconsistent acceleration forces. The reason for this, is that inconsistent acceleration forces could cause traffic congestion and an uncomfortable experience for people inside the vehicle. The implementation of this constant force towards the next point is described in formula 4:

$$\vec{f_d} = \frac{\vec{q} - \vec{p}}{|\vec{q} - \vec{p}|} \cdot dist \tag{4}$$

In which $\vec{f_d}$ represents the destination factor, $\vec{q}$ represents the coordinates of the next point in the coordinate track, $\vec{p}$ represents the centre point coordinates of the vehicle and $dist$ is the desired distance for the striving point. This equation turns the vector that points towards the next location in the destination track into a unit vector, and multiplies it with a fixed distance to ensure that a constant force is applied to the vehicle. Essentially, the exponential component that varies with distance is replaced with a constantly present point in front of the vehicle toward which it should strive.
When the vehicle finds itself within a certain range of a location marker, it adds that marker to a list with visited locations. During every simulation step, the vehicle checks if it already visited the location. If it did, the new target will be the next location in the destination track.

To support the destination tracking system, a road marker simulation object is added. These road markers are stylized in the simulation as dashed lines, and act the same as regular walls to vehicles that have to keep following the road, apart from not being able to collide with the vehicle. This means that when a vehicle has to exit the road, it will be able to do

so without issue, since the separation force exerted by the road exit is simply ignored in this case. Other vehicles will keep driving though, and if they happen to be accidentally forced over the road marker, the force that the road markers apply will help get the vehicle back on the main road.

Since compute shader buffer sizes have to be fixed during the entirety of the openGL context run time, the buffer that contains the location markers for the destination tracks for each vehicle has to be pre-allocated. The location markers are stored in a csv file that contains a specific track corresponding to a vehicle in rows, and x and y coordinates in alternating columns. During the import phase of the location markers, the python driver code keeps track of the maximum number of location markers out of any individual destination track. Then, an N by L buffer block is created where N represents the number of vehicles, and L represents the maximum number of location markers. If a vehicle has no specific destination or if it has a limited amount of location markers in its route, the rest of that vehicle's buffer entry is filled with coordinates out of road bounds. The shader code that performs the destination algorithm will simply ignore further entries for that vehicle and the vehicle will follow the road as normal after this point.

D. Velocity control

Velocity control is a crucial component to add to the previous iteration of the simulation environment. Without any form of active braking and solely relying on the separation force to slow a vehicle down, destination tracking will not be able to function to its full potential, and crashes will happen that are entirely preventable. Two options were considered for velocity control as a means to avoid crashing. The first of which is a force-based approach, and the second a scaling based approach.

To better understand what choice was made, it is important to understand the structure of the algorithm compute shader. The shader consists of four parts: vehicle-to-vehicle force evaluation, vehicle-to-wall force evaluation, vehicle-to-road marker force evaluation and destination tracking forces. In between these stages, the velocity vectors are normalized to the desired speed of the vehicles. In any case, due to the current structure of the compute shader, the velocity control decision and the corresponding action have to be applied in the "evaluation loop" where the discovery that led to the decision was made. For a force-based approach this poses a problem. As an example: if the need arises to brake in the vehicle-to-vehicle stage, the resulting velocity vector will be calculated such that a crash can be prevented. This does not necessarily mean that the final velocity vector will fully reflect this change: if there is no risk of collision towards the other kind of simulation entities found in the subsequent evaluation loops, there will still be a separation and alignment force applied to the vehicles, making it likely that vehicles will not brake with the force required.

It is possible to re-structure the algorithm compute shader, but due to the fact that the majority of GPUs are unable to stream and synchronize real-time data back to the CPU without complex tools, debugging becomes difficult. Instead, the more simple, more compatible method to the current compute shader structure is chosen: a scaling factor that is applied to the algorithm's output velocity vector.

The scaling method works by first running the boids algoritm as normal, and then performing a series of checks to confirm if another vehicle is in front of it using a "crash field of view". During the evaluation loops mentioned before, the distance and angle of the closest other simulation entity are kept track of. After each loop, the lowest distance is compared to the proportional braking distance d_p and emergency brake distance d_e, and the angle is compared to the pre-set value that corresponds to the crash field of view. If the conditions for a braking stage are met, the required braking force of either of those stages is applied. Distance calculations are performed from centre point to centre point of vehicles, and a description of how the angle is calculated between vehicles can be found in appendix section V-A.

The advantage of this method is that the scaling factor can be calculated anywhere and is applied only at the end, which means that the contributions from all of the four stages are dampened at once.

For the proportional braking stage d_p, the scaling factor is calculated using equation 5:

$$scaling = \frac{d_{min}}{f_p} \tag{5}$$

In which d_{min} is the minimum distance to another simulation entity, and f_p is the weight given to the proportional braking. This method of scaling allows for a less jittery braking phase, since less braking force is applied when the vehicle is further away from the obstacle. This stage is meant to serve as a bridge between what the separation factor can manage by itself and emergency braking. In the emergency braking stage, the velocity vector is set to near-zero. The actual deceleration is clamped to the maximum deceleration force of the car on which the vehicles are based, so this will not result in physics-defying brake behavior.

E. Algorithm

The following section will describe the sequence of execution for the design of the control algorithm as sketched in this paper and provide commentary on changes and new additions to the simulation environment designed in [2]. This section will only discuss the computational side of the algorithm, since the rendering shaders are non-relevant in this context. The final algorithm is split up into multiple compute shaders. For algorithm initialization, two pre-calculation shaders are used:

- A pre-calculation shader to calculate location and wall normal information and stores them in the GPU buffer. This shader remains unchanged from its inclusion in [2].
- A pre-calculation shader for streaming location and road marker normal information into the GPU buffer.

As these pre-calculation shaders are solely meant for initialization, they are only executed once. The road markers and their

information buffer are a new addition, but these two compute shaders are very similar in nature. Separation of these shaders allows for more structured shader programming and allows for easier debugging.
The algorithm itself is performed in the following shaders:

- Distance calculation: this shader is responsible for calculating distances and angles between all simulation entities and stores them in the distance buffer. Collision detection by means of the separating axis theorem is also performed, and if a collision occurred, the simulation state buffer entry for collisions of those vehicles are flagged as positive.
- Algorithm: this shader performs a round of cohesion, alignment and separation calculations for vehicles on vehicles and vehicles on walls, and scales the velocity request vectors based on if a crash is imminent. If the destination track buffer has a valid entry for the invocation ID of a specific vehicle, the final set of calculations of the algorithm shader applies force towards the vehicle's destination if further conditions are met.
- Vehicle movement: as the name suggests, this shader executes movement of the vehicles and stores the new vehicle positions and rotation in the buffers. Movement is restricted to the physical limits of the vehicles, so it prevents unrealistic acceleration or deceleration and enforces a maximum steering angle. This shader also remains unchanged apart from minor bugfixes.

III. Experiments and results

The following section will describe the experiments that were used to improve the performance of the control algorithm and verify its performance. The sub-research question: *What kind of experiments are required to measure the performance of destination based navigation and velocity control?* will therefore also be answered in this section.

A. Velocity adaptation validation and parameter sweep

In order to make sure that the velocity adaptation is working as intended, a test setup is created where vehicles drive towards a y-junction. The most favorable situation is where the braking force and emergency distance are as small as they possibly can be while being safe, or in other words, the lowest jitter in vehicle velocity. This way, passengers will not feel uncomfortable with the high acceleration and deceleration forces. Figure 2 shows the vehicles positioned in a grid structure. The green triangles in front of the vehicles show the field of view of a vehicle in which a braking operation will trigger if another vehicle's centre point is located within the triangle. Half of the vehicles will follow the track of red dots to the left and the other half will follow the track to the right. Which vehicle gets which track is chosen in such a way that the paths of the vehicles are somewhat intertwined. This will test the vehicles' ability to break to avoid crashing, as well as the vehicles' ability to follow a location marker track.

The performance measures for velocity adaptation are the following:

- Velocity standard deviation (σ_v): the average standard deviation of the grid of vehicles as shown in figure 2 will be used to gauge ride quality. Higher standard deviations are less desirable, and indicate more jitter in vehicle speed or even crashes. For the final decision on parameters, optimization will be performed based on the minimum σ_v.
- Collisions: For a set of parameters to be usable, a crash occurring is out of the question.
- Time spent (frames): Using just the standard deviation and collisions could result in situations where indeed no crashes have happened with a smooth ride quality, but it is possible that these parameters are not up to standard when it comes to travel time. A hard maximum time of 1300 frames is set for the full grid of vehicles to reach their goal. This time was empirically determined to be more than enough for the vehicles to reach their final destination, without being excessive. The final destinations are either of the two upper red dots in figure 2

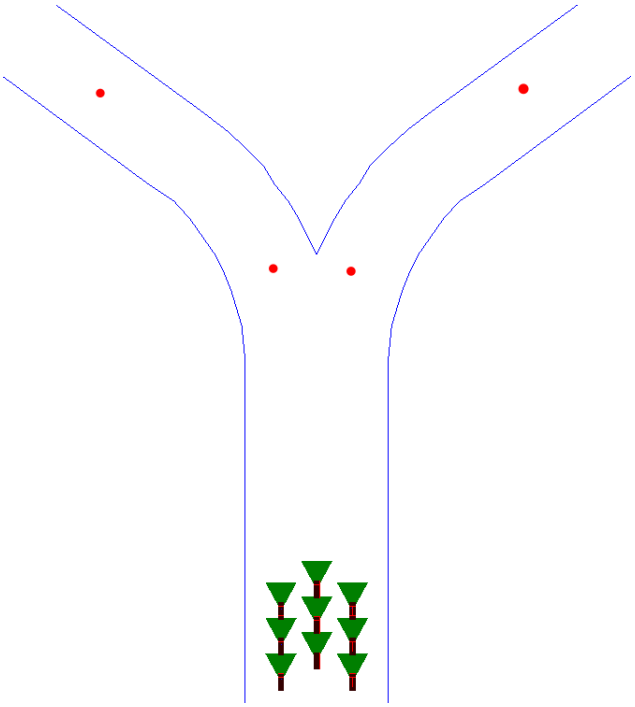

Fig. 2. Experiment setup for speed adaptation validation

The first step in finding ideal parameters is finding the minimum safe braking distance. During testing, it was found that the emergency braking stage has a significant impact on crash prevention. To find a minimum safe distance, a sweep was set up with the following parameters:

- f_s: The weight of the separation component.
  Range: 2.0 to 3.0. Step size: 1.0

- d_p: Proportional braking distance.
  Range: 7.0 to 10. Step size: 0.5
- f_p: Proportional braking force scaling factor.
  Range: 5.0 to 15.0. Step size: 1.0
- d_e: Emergency brake distance.
  Range: 4.0 to 7.0. Step size: 0.5

These ranges for these parameters were determined empirically, and cover a reasonably wide range of possible combinations (980 in total). They were also chosen to be consistent and kept in line with the speed ($2m/s$) and parameters determined in [2]. It should be noted that the safety result of 140 simulations in total for each braking distance serves as an indication of where to start.

To reproduce the experiments, the map files *y.wls*, *y.loc* and *y.spn* should be used. The four parameters mentioned before vary with the sweep. Other notable paramters are:

- Cohesion weight: 0.1
- Cohesion distance: $15m$
- Separation distance: $10m$
- Alignment weight: 6
- Destination tracking pointer distance: $20m$

These parameters remain consistent for all experiments.

B. Destination system validation

For the destination tracking, two types of more challenging road environments will be tested: a y-junction and a highway exit. The destination tracking points will be plotted together with the path of the vehicle(s) to assess if the vehicles' actual paths are close enough to the pre-defined track of points. Both of these situations will be evaluated for a single vehicle, as well as for multiple vehicles on the road to simulate quiet and more busy roads. What is expected for a functioning system is that the vehicle's path will come within an acceptable range of the location marker and move on to the next one, because the markers are supposed to act more as a guideline rather than the exact track a vehicle should take. If all vehicles have to cross the exact location marker coordinates, they will have to line up to do so which creates congestion and will possibly cause dangerous traffic situations. There is a high likelihood that the path will not be completely direct between points, due to the forces exerted by other simulation entitites. Even for single vehicle scenarios, the walls will most likely slightly throw off the ideal curve.

The test setup for the y-junction can be observed in figure 2. The red dots indicate the (approximate) location markers as used in the simulation. A path to the left and a path to the right is defined. The markers and road segments are mirrored in the middle, so for the singular vehicle test, the vehicle can be placed to the left, right and in the middle of the road leading to the y-junction and the resulting vehicle path should be very similar if not the same depending on whether the vehicle goes left or right. For the test with multiple vehicles, half of the vehicles will be assigned to the left path and half of the vehicles will be assigned to the right path randomly.

The test setup for the highway exit can be observed in figure 3. Once again, the red dots indicate the approximate location markers as used in the simulation. The dashed line is a road marker that exerts a force on the vehicles that will continue along the road, acting as an artificial wall segment. Vehicles that are supposed to take the exit will ignore this force. In the single vehicle scenario, the path of the vehicle should show a smooth line close to the location markers. In the multiple vehicle scenario, the paths will likely deviate a little from the ideal line between the location markers, and the vehicles will take longer to reach the final destination, but depending on the maximum amount of vehicles that will fit through the slightly narrowed curve section, they should be able to follow a path close to the location markers without issue.

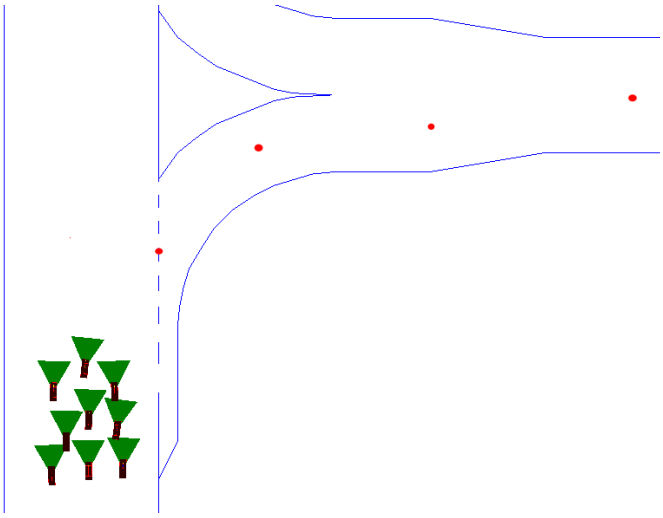

Fig. 3. Test environment 2 for destination system validation

The performance measures for the destination tracking system are the following:

- The traces of the vehicles' paths and their proximity to, and fluidity through location markers.
- Travel time.

C. Results: Velocity adaptation

Figure 4 shows the result of the first parameter sweep described in section III-A with the intention of finding a baseline for the minimum emergency brake distance.

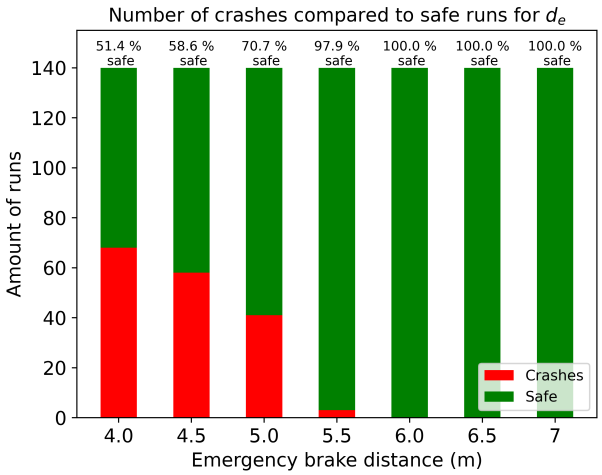

Fig. 4. Crashes vs safe runs for varying d_e

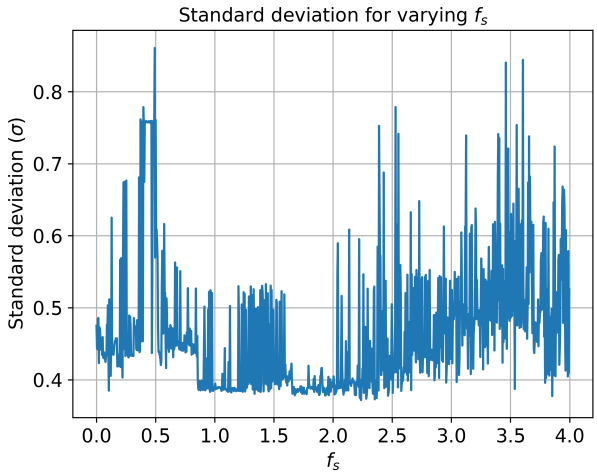

Fig. 5. Optimization plot for separation weight

Figure 4 shows a bar graph that compares the amount of crashes versus safe, crash-free runs in the original wide parameter sweep for which the ranges are described in section III-A. The total number of simulation runs is 980. While this graph should not be used for definite conclusions on what minimum brake distance to use at the given speed of $2m/s$, it can be used as an indication for a starting point of a higher resolution sweep. In the graph, there is a clearly observable downward trend in crashes when the emergency brake distance d_e becomes larger. Using this graph, the sweeping range for figure 8 was determined. With this same sweep, it is found that the following variables give the lowest average standard deviation when only simulations with a minimum brake distance of 5.5 are taken into account that contain no crashes and no simulations with vehicles that haven't been able to reach their goal before the simulation ended:

- Separation component weight f_s: 2.0
- Proportional braking distance d_p: 9.5
- Proportional braking force scaling factor f_p: 5.0
- Emergency braking distance d_e: 6.5

Even though a minimum brake distance of 5.5 m does result in a few crashes in figure 4, preferably the brake distance should be as small as possible. There could be a better standard deviation value with no crashes anywhere in between 5.5 and 6, so values above 5.5 are still taken into consideration the optimization process.

An optimization process was performed for these variables, resulting in the following plots:

Figures 5 and 8 show the clearest ranges of what their respective parameters should approximately be. For the separation weight f_s, for which the plot is observable in figure 5, the most stable behavior also happens to bear the lowest standard deviation. This behaviour is observed between a value of approximately 1.6 to 2.0. The higher peaks are caused by runs that contained one or more crashed vehicles. For this optimization sweep, crash events are more or less evenly distributed throughout the parameter range, with a notable absence between a weight of approximately 0.8 and 2.4

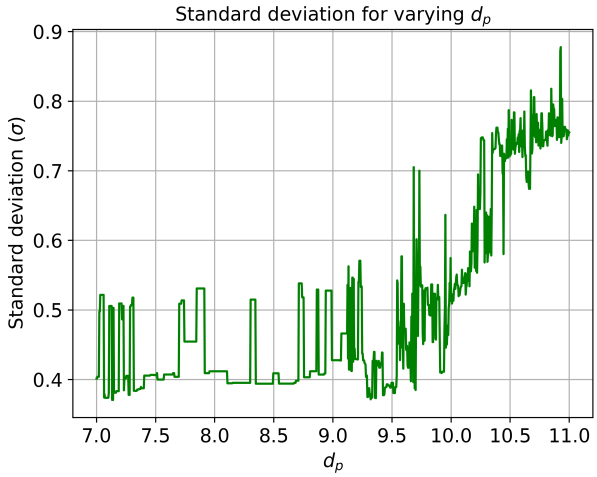

Fig. 6. Optimization plot for proportional braking distance

For optimal usage of proportional braking, it's range should be big enough to bridge the gap between emergency braking and obstacle avoidance by means of the separation factor. It is meant as a transition phase after all. Looking at the plot in figure 6, the lowest range of standard deviation that meets this criteria, as well as being in a "stable" range of

standard deviation, the ideal option would be a value within the range 8.0 to 9.0. In this optimization sweep, 186 crashes occured, with all but 3 of them being for d_p values greater than 10.1. The most plausible explanation for this is that the the proportional braking and separation forces overlap. The scaling factor will be applied to the entire "requested" velocity vector, thus also scaling the alignment with flock members. This likely results in a "chain of collisions" of sorts due to the fact that the average standard deviation is so high for larger d_p.

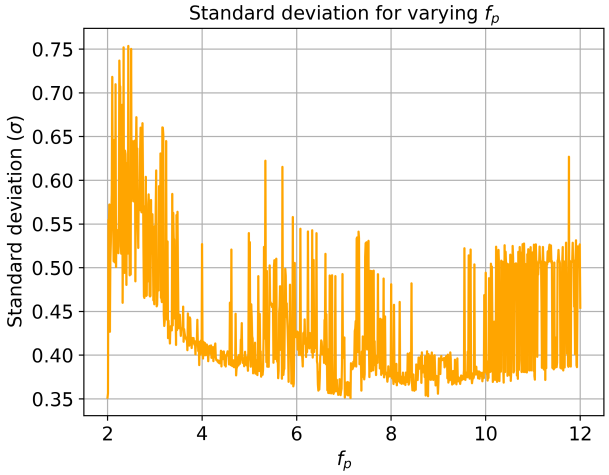

Fig. 7. Optimization plot for proportional braking weight

The proportional braking weight is at its most stable with the lowest standard deviation in the range 8.5 to 9.5. Intuitively, having the highest standard deviation for lower braking forces like figure 7 displays is not the most logical observation because braking is not applied with as high of a force. This is explained by the fact that when the braking force in the proportional braking phase is too low, the vehicle will have to over-rely on emergency braking, causing a jittery vehicle speed, or the vehicle will simply crash. Data collected during the optimization sweep shows that out of the 50 crashes in the 1000 iterations performed, half of them take place in the f_p range 2 to 3.27, which coincides with the large peaks at the beginning of the plot. If a vehicle crashes somewhere halfway in the run, the mean value will lie somewhere around $1m/s$, and then the standard deviation quickly becomes larger, which also explains the higher standard deviation in this parameter range.

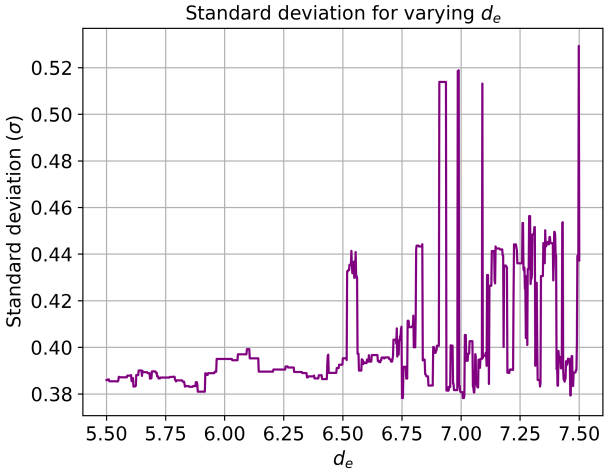

Fig. 8. Optimization plot for emergency braking distance

Figure 8 shows the optimization plot for the emergency brake distance d_e. Stable behavior with low standard deviation is observed below a distance d_e of approximately 6.5 m when keeping f_s, d_p and f_p at their "optimum" that was obtained before. The lower standard deviation in this region makes sense: waiting longer for braking with the highest force possible and instead relying on proportional braking first should provide a less jittery ride, which will result in a lower standard deviation. No crashes happened during this sweep, which can also be seen in the plot by the absence of high peaks (0.6) like those found in the other plots.

A final, high resolution sweep is performed in the stable ranges as determined in this section:

- f_s: 1.6 to 2.0 Step size: 0.05
- d_p: 8.0 to 8.75 Step size: 0.05
- f_p: 8.0 to 9.0 Step size: 0.05
- d_e: 5.75 to 6.5 Step size: 0.05

The sweep in these stable variable ranges resulted in the following parameters in search of the lowest standard deviation:

- f_s: 1.6
- d_p: 8.5
- f_p: 8.75
- d_e: 5.9

These parameters result in a velocity standard deviation σ_v of 0.3798 in the y-junction test environment.

D. Results: destination tracking

In order to gauge the performance of the destination tracking system, the test setups as described in section III-B are used in combination with the ideal parameters that were extracted from the stable ranges in figures 5, 6, 7 and 8. The following plots show the traces of the vehicles compared to the location markers, as an evaluation of the algorithm's ability to smoothly follow a track of markers.

It is important to note that for the plots in figure 9, 10, 11 and 12, the origin of the vehicles is the lowest y coordinate in the plot: the vehicles are driving upwards. The plot in figure 9 shows the trace of a single vehicle going to the left on the y-junction road as portrayed in figure 2.

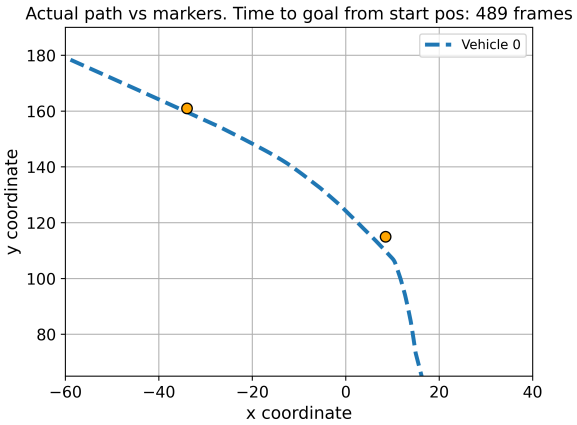

Fig. 9. Trace of a single vehicle on the left side of y junction

In figure 9, the dashed trace represents the left path of the y junction test simulation, which the vehicle took in the simulation. The orange dots with black outline represent the location markers. This will be the case for figures 10, 11 and 12 as well.

The path that the vehicle took came within sufficient proximity of the location markers, indicating that the destination tracking works reasonably well. Around coordinate (10.0, 105.0), the vehicle takes a relatively sharp right. There are two likely reasons for this, which both have to do with the map layout. The first of which was discovered during testing, which is that the boids algorithm does not bode very well in confined space with curves with large, ”mismatched” discrete wall segments. What this essentially means, is that in order to create a (sharp) curve that works well with vehicles, relatively short line segments have to be drawn as the walls that have to be roughly matched to a line segment on the other side of the road. The ”v” shape in the centre of the y junction also causes some strange behavior, since those line segments are very close to each other. The line segments will exert a force in opposing directions. Once the vehicle has progressed a certain amount on either of the two lanes, the line segment that exerts the unwanted force is outside of the detection range. The transitional point where this happens will likely not be very smooth.

The second possible reason for this sharp bend is that the placement of location markers is relatively far apart. The large spacing between location markers was done on purpose however, to make sure that it is not required to calculate an exact line for the vehicles to take.

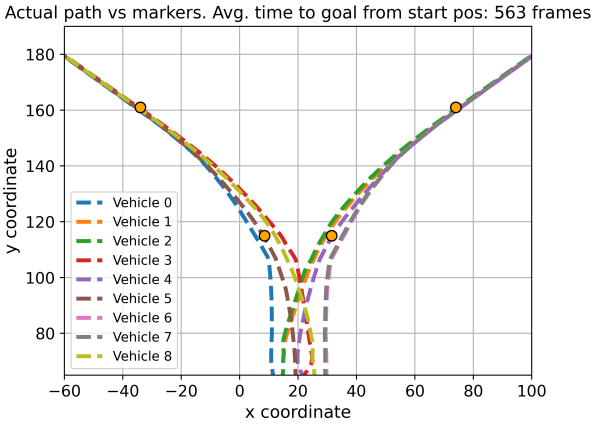

Fig. 10. Traces of 9 vehicles for both sides of y junction

Figure 10 represents both sides of the y junction test road and shows the traces of nine vehicles in total. A noteworthy observation is that for vehicles that started on the opposing side of the road to where they were going, the ”sharp bend” near the first location marker is much less noticeable than for the single vehicle, and for the vehicles that start in the centre or at the same side of the road as the side they are going. This is especially clear for vehicle 2 (green trace) and vehicle 8 (yellow trace). There is a sharper bend present for vehicle 2 and 8 as well, but it takes place just below y = 80.0. The initial positioning and earlier bend does line vehicle 2 and 8 up better for a clean curve through the location markers. Since the road is relatively small, vehicles start to follow each other in a line formation (caused by alignment) through the smaller corridors rather than forming clusters again. The alignment between vehicles is likely also one of the causes for the slightly smoother lines. Compared to the simulation displayed in figure 9, the average vehicle takes 74 frames of extra time when roads are busy, which translates to 7.4 seconds extra over a distance of 100 meters. At a speed of $2m/s$, this is deemed an acceptable difference.

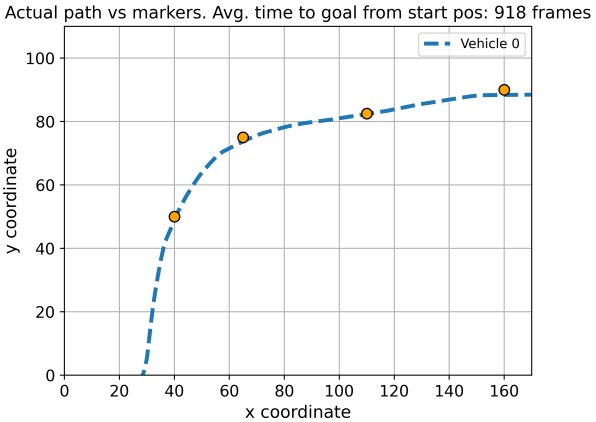

Fig. 11. Trace of a single vehicle taking the road exit

The trace in figure 11 shows a smooth line through the location markers where the contours of the road are clearly visible in the vehicles path. An example of this is the stretch between the location marker at approximately (110.0, 82.5) and the location marker at (160.0, 90.0). Cross-referencing this with the map layout as displayed in figure 3, there is a section of road that starts to narrow towards the centre of the smaller road section. There is a sharper bend present in the vehicle's trace at approximately (55.0, 70.0). The most likely explanation for this is that this location is also close to a sharp corner that consists of relatively large discrete curve line segments (especially on the outer edge of the curve).

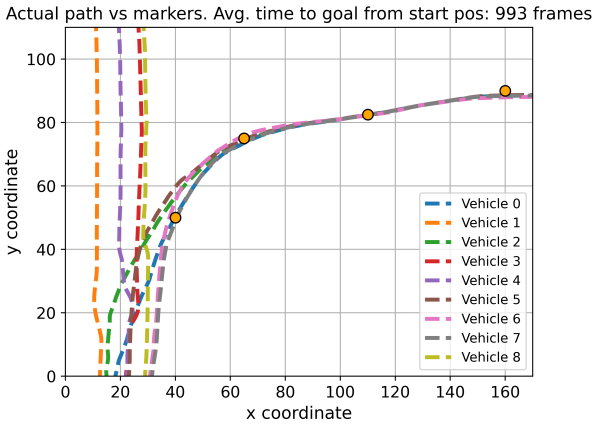

Fig. 12. Traces of 4 vehicles taking the road exit with the rest on the main road

The final test for the destination tracking system is the highway exit scnenario with a busy road. The first observation is that there is a difference of 75 frames between the average of four vehicles taking the exit and the time it took for a single vehicle to get to the final location marker. This translates to a real life time loss of 7.5 seconds over a distance of 200 meters. This indicates that this traffic situation has better traffic flow compared to the y junction. Two of the vehicles leaving the flock on the highway initially came from the left side of the flock, and two of them came from the right side. In terms of angle, the same observation can be drawn as for the y junction scenario with multiple vehicles, which is that after the second half of the vehicle traces, a line formation has formed and the location markers are followed quite smoothly. The vehicles leaving the flock, especially vehicle 2 and 0 reach a state where the angle towards the location marker is starting to become too large, with one or more vehicles too close to safely turn towards the exit. What the vehicles do in this case, is slow down until there is enough room around them to make a safe turn. This is clearly visible by the trace of vehicle 1, which corrects its trajectory around the turning location of vehicle 2. With human drivers, behavior like this will likely cause congestions or accidents, but since in this context, the vehicles are automated, this is not as big of an issue. While not visible in the plot, the vehicles were later observed to catch up and form a flock again.

E. Future work

While the addition of the speed adaptation, improved crash detection and destination tracking has made the simulation environment more realistic, there are still points of improvement. To name a few, future renditions of this simulation environment could add path-finding for destinations, intention messaging for smoother traffic flow, variation in vehicle shape (model a semi-truck and trailer for example) and improved vehicle detection. The simulation has also not been verified with higher speeds. This would likely require performing a new sweep of variables like performed in the context of this paper, together with the parameters that were decided on in [2]. The base for more accurate distance measurements between vehicles is present with the introduction of the separating axis theorem, but could not be fully implemented.

IV. Conclusion

The addition of waypoint based navigation and velocity vector scaling as a means of velocity control to a boids algorithm provides a control strategy for simple every-day road situations in an efficient simulation environment making use of compute shaders. The system was found to be capable of safely guiding individual vehicles towards a destination with limited speed fluctuation, providing an acceptable ride quality in reasonable time. Stable behavior can be achieved for a decent range of control parameters. Compared to the standard boids algorithm, the control strategy provided in this paper is able to overcome more challenging scenarios in confined space. These include, but are not limited to models of a y-junction and a highway exit.

The performance of the velocity control system was able to be evaluated by using the average standard deviation of a vehicle flock as a measure of speed fluctuation, while also measuring the time it took to reach a goal without a single crash occurring. The destination control system was evaluated by comparing the vehicle's trajectories to the layout of location markers, and judging the traces' accuracy and smoothness. By implementing a more sophisticated crash detection algorithm, the system's performance was able to be evaluated more carefully, resulting in fewer false-positive crash detection reports. This opened up a broader range of control parameter values to use and improve the system with.

While the addition of destination tracking and velocity control brings the boids inspired simulation environment as described in [2] closer to a more realistic traffic simulation environment, it is still a basis. For one, it is important to evaluate the functionality of the control strategy at higher speeds. While the road scenarios are designed to resemble more realistic traffic situations, they are loosely based on sattelite images and could be improved upon by creating a tool for smoother curve creation, or following road design guidelines publicised by government agencies.

REFERENCES

[1] C. W. Reynolds. (1987) Flocks, herds and schools: A distributed behavioral model. [Online]. Available: https://dl.acm.org/doi/10.1145/37402.37406

[2] J. Blondel. (2021) Distributed control algorithm for cooperative autonomous driving vehicles inspired by flocking behaviour. [Online]. Available: https://essay.utwente.nl/85648/

[3] H. C. et al., "Truck platooning reshapes greenhouse gas emissions of the integrated vehicle-road infrastructure system," *Nature Communications 14, article 4495*, 2023, available: https://www.nature.com/articles/s41467-023-40116-0.

[4] R. Olfati-Saber. (2006) Flocking for multi-agent dynamic systems: algorithms and theory. [Online]. Available: https://ieeexplore.ieee.org/document/1605401

[5] C. F. Caruntu, C. Pascal, L. Ferariu, and C. R. Comsa, "Trajectory optimization through connected cooperative control for multiple-vehicle flocking," *2020 Mediterranean Conference on Control and Automation*, pp. 915–920, 2020, available: https://ieeexplore.ieee.org/document/9182964.

[6] C. F. Caruntu, L. Ferariu, C. Pascal, N. Cleju, and C. R. Comsa, "Connected cooperative control for multiple-lane automated vehicle flocking on highway scenarios," *2019 23rd International Conference on System Theory, Control and Computing*, pp. 791–796, 2019, available: https://ieeexplore.ieee.org/document/8885496.

[7] P. C. et al., "Collision detection using axis aligned bounding boxes," *Simulations, Serious Games and Their Applications*, pp. 1–14, 2014, available: https://link.springer.com/chapter/10.1007/978-981-4560-32-0$_1$.

[8] J. wung Choi, R. E. Curry, and G. H. Elkaim, "Continuous curvature path generation based on bezier curves for autonomous vehicles," *IAENG International Journal of Applied Mathematics, 40:2*, pp. 91–101, 2010, available: https://users.soe.ucsc.edu/~elkaim/Documents/IAENG_JAM_Choi.pdf.

V. APPENDIX

A. Angle calculations velocity control

For the angle calculation, the first step is to take the dot product between the vehicle's relative rotation and the vector between the centre points of vehicle i and vehicle j. After this, the expression is normalized and then the arc cosine is taken. Equation 6 demonstrates this:

$$angle = cos^{-1}\left(\frac{\begin{bmatrix} cos(-rot_i + \frac{\pi}{2}) \\ sin(-rot_i + \frac{\pi}{2}) \end{bmatrix} \cdot \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix}}{abs\left(\begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix}\right)}\right) \tag{6}$$

In which rot_i is the rotation of vehicle i in world space, and $(x, y)_i$ and $(x, y)_j$ are the coordinates in world space of vehicle i and j respectively. The reason why the cosine and sine contain the negative rotation in world space and a factor of $\frac{\pi}{2}$ has to do with how the coordinate frame is defined in the simulation environment: usually, the rotation direction is positive when going counter-clockwise, and an angle of zero is defined when pointing parallel to the positive x axis. In the simulation environment, the clockwise direction is defined as positive and a rotation of 0 degrees is when the vehicle is pointing parallel to the positive y axis.