# Distributed Control Algorithm for Cooperative Autonomous Driving Vehicles Inspired by Flocking Behaviour

Joppe Blondel
*Faculty EEMCS, University of Twente*
Enschede, the Netherlands
j.blondel@student.utwente.nl

***Abstract*—Control strategies for cooperatively autonomous driving vehicles are becoming more and more complex to ensure safe operation while, when taking inspiration from nature, more simple mathematical descriptions exist which describe behaviour of animals moving in a flocking or herding manner. In the eighties Reynolds came up with a set of heuristic rules [1] which form a model for flocking behaviour of birds and his work has been an inspiration for constructing a control strategy based on flocking behaviour for self-driving vehicles. Previous research relied on more simple kinematic models of vehicles and thus using it directly in real vehicles would be difficult. This research provides an algorithm inspired by flocking behaviour as found in swarms of birds and a bridge between said algorithm and a vehicle. A dedicated simulation environment which uses parallelism on a GPU is written and is used to determine the performance of the proposed algorithm. It is found that the proposed algorithm performs relatively well under the tested situations and is found that a wide range of parameters could be used which result in stable behaviour.**

***Index Terms*—Cooperative Autonomous Driving, Flocking behaviour, Self-driving cars, Boids**

## I. Introduction

Traffic jams, car crashes and environmental impact of present-day traffic are things that are inseparable of the view of modern traffic. Where were empty roads and content drivers are now congested traffic arteries during the more and more extended rush hours. Within most of the car accidents, the driver was responsible.

To eliminate the human factors within traffic congestion and car crashes one could eliminate the human driver itself. Research into self-driving cars has provided us with autonomous driving cars which can navigate themselves within modern-day traffic by imitating the human driver. Adding communication to self-driving vehicles will lead to anticipation of abrupt actions and movements and thus will probably lead to less congestion and more safety. Cooperative Autonomous Driving (CAD) is an area in which a lot of active research is done.

Since algorithms for CAD must be safe and avoid crashes in any cases, the rules on which these algorithms rely are growing more and more complex. Instead of adding more and more rules to ensure safe operation, one can look completely the other way and take inspiration from natural phenomena. Within nature, there are a vast amount of animals which travel in large groups, herds or flocks without any collisions. Research has shown that to simulate such flocking behaviour, one does not need a complex set of rules [1]. This research is driven by the question if such behaviour can be mirrored in modern traffic.

The goal of this research is to see if these set of rules can be used in self-driven traffic and if so, how well such an algorithm can perform in different situations with strict spatial constraints (vehicles must not drive out of designated areas). Within this research, an attempt will be made to implement such an algorithm and it will be evaluated in terms of efficiency and safety.
Back in 1986, Reynolds had shown interest in flocking behaviour and provided an approach for simulating flocks with a simple set of heuristic rules in which the animals are called `boids` (bird-oids): *cohesion (or flock centring)*, *alignment (or velocity matching)* and *separation (or collision avoidance)* [1]. In the same paper, two ways of obstacle avoidance are mentioned:

- Avoidance based on a 'force field'
- Avoidance based on 'steer-to-avoid'

Since Reynolds formulation of his rules did not specify any mathematical algorithms, multiple interpretations of the algorithm are around. [2] poses a method which uses graph theory and is elaborated on in [3]. This paper provides us with 3 algorithms and gives a formal description of flocks which can be used to analyze flocking algorithms.

Within most research boids are considered as single points or circular disks. [4] takes another approach and poses a constructive method to design a flocking control algorithm for mobile agents with an elliptical shape and mentions ships as a possible application. Research in flocking behaviour used in control algorithms for aircraft is done by Crowther in *Rule-based guidance for flight vehicle flocking* [5].

Within the field of (cooperative) autonomous driving itself little research is done. Kakaria's *Boids On Wheels* [6] shows a proof of concept study on simulating traffic with vehicles acting as boids and *Flocking Algorithm for Multiple Nonholonomic Cars* of Hayashi and Namerikawa [7] provides an algorithm which enables multiple nonholonomic cars to travel as a flock on a highway. More research on flocking control on nonholonomic agents is done in [8]. Recent research on the usage of flocking behaviour in CAD [9] shows that

Reynolds's rules suffice as a control scheme for autonomous vehicles (which bring in a set of constraints with them such as maximum velocities and controlling a vehicle indirectly by steering angle) if, and only if, a balance is formed between the different 'forces' of each rule which occurs when a flock is formed. In general, these three rules alone are too simplistic to act as a control mechanism. The addition of goal-driven behaviour decreases the amount of collisions but does not completely diminish them.

Even though research is done into the application of flocking behaviour in the control of self-driven vehicles, evaluation of such algorithms on every-day traffic situations such as crossings, round-a-bouts' and highways has yet to be done. Without additions and alterations, especially within the coupling between the flocking algorithm and the control of a vehicle, the relatively simple Boids algorithm will not be feasible to be used in self-driven traffic. This research will provide the needed coupling between the flocking algorithm and the vehicle control and will build upon [9] with the addition of obstacle avoidance. Furthermore it will provide an efficient simulation environment using parallelism on the GPU for smaller execution times. In this paper the following questions will be answered.

- How well does a flocking behaviour inspired control algorithm for autonomous driving vehicles work with strict spatial constraints?

To answer this question the following sub-questions must be answered:

- How can we measure the performance of a flocking behaviour inspired control algorithm for autonomous driving vehicles?
- How can we add obstacle avoidance to a flocking inspired control algorithm for autonomous driving vehicles while keeping reasonable stable behaviour?

The outline of the paper is as follow: Some background on the original Boids algorithm and a mathematical description of the kinematics of a vehicle are provided in Section II. The algorithm and the coupling with the kinematics are described in Section III. The choice for writing a specific simulation environment and the description of the environment is presented in Section IV. Section V will consist of a few steps for evaluating the proposed algorithm. Finally concluding remarks, encountered difficulties and directions for future work are provided in Section VI.

## II. Preliminaries

This section will give the reader information which is necessary to read the following sections of the paper. Assumptions made or specific interpretations of certain information are stated.

### A. Boids

Modeling of the flocking behaviour of birds and other animals has kept people of multiple disciplines busy for a long time. Reynolds provided in 1986 a basic set of rules which can be used for such flocking [1]. Within his paper each object which moves according to his rules are called *boids* which is used in this paper as well. Reynolds stated the three rules as follow:

- Flock Centering. This rule makes a boid want to steer to the center of nearby flockmates. The farther a boid is from this center, the stronger the urge to go towards it.
- Velocity Matching. Each boid will try to match its velocity with its flockmates.
- Collision Avoidance. This rule will add the urge to steer away from an imminent impact and thus avoid collisions with other flockmates.

To combine these rules Reynolds proposes that each rule gives an *acceleration request* which are in its most simplistic form averaged to get the direction of the movement of the boid.

Since the rules stated by Reynolds are qualitatively described, multiple interpretations of these rules can differ strongly. Within this research the rules are interpreted as follow (rules are depicted in Figure 1 as well):

- Flock Centering (from now on called *cohesion*) will give an acceleration towards the center of all other boids within a certain radius of perception. The farther away the boid is from the center, the higher the acceleration will be (in a linear fashion).
- Velocity Matching (from now on called *alignment*) will give an acceleration by taking the average from all velocities (direction and speed) of nearby boids (within the same radius of perception as with cohesion). No distinction is made between the boids within the radius of perception.
- Collision Avoidance (from now on called *separation*) will give an acceleration per boid within a smaller separation radius in the direction pointing away from the other boid. The magnitude of the acceleration is dependent from the distance between the boids.

The accelerations for each rule are combined by taking a weighted average and the result is used as the acceleration of the boid itself.

### B. Vehicle kinematics

Within other researches boids are simulated as points which are able to accelerate in all directions or as simplified vehicles which have constraints to resemble real life vehicles ([7]). However, in a try to simulate vehicles as realistic as possible, the kinematics of a car are elaborated upon. Most road vehicles have fixed rear axis and are steered by rotating the wheels on the front axis (Ackermann steering [10]). This section will provide a mathematical description of the motion of such a vehicle.

As basis a vehicle of length $l$ is taken with the center of mass at the center of the vehicle itself (see Figure 2). Changing direction is done by rotating the front wheels around their center point (which results in a rotation of $\varphi$ of a virtual wheel in the center of the front axis). The angle between the roll axis of the vehicle and the global y axis is called $\theta$, the velocity of

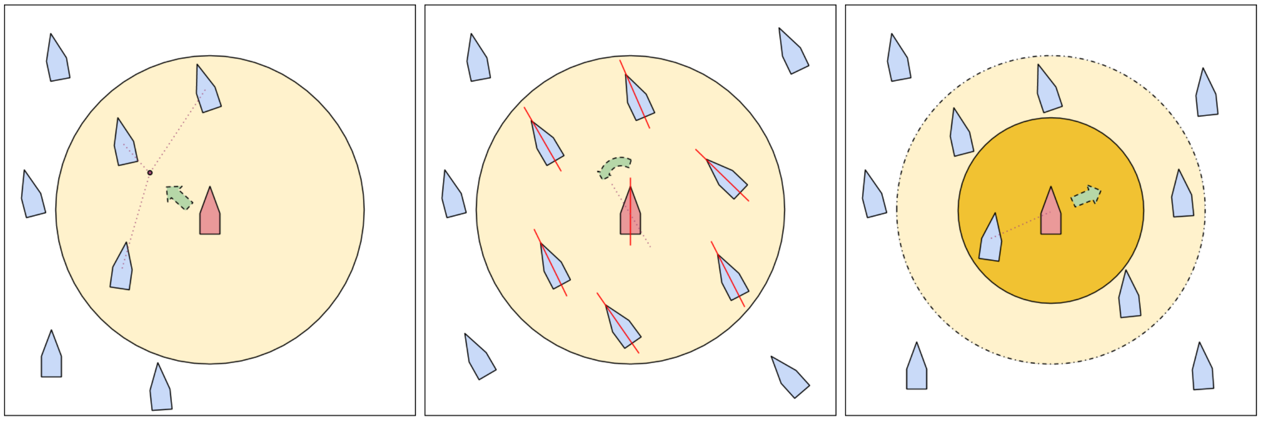

(a) Cohesion (b) Alignment (c) Separation

Fig. 1: Visual representation of the rules for flocking behaviour

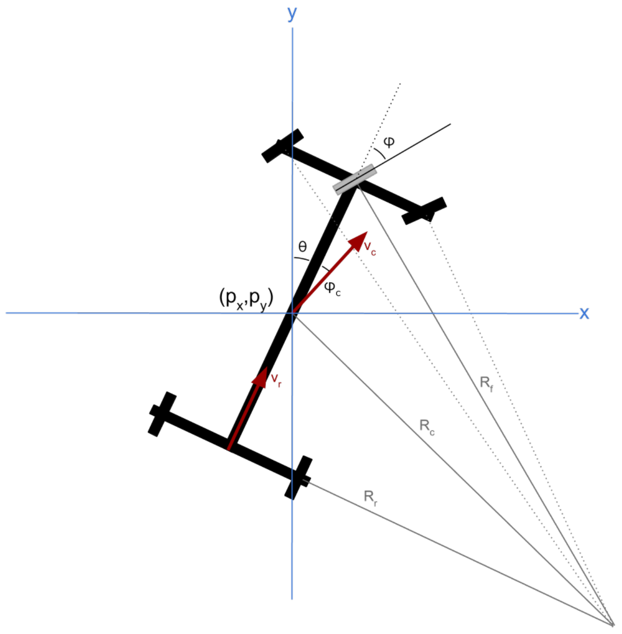


Fig. 2: Vehicle kinematics

the car is given as vector $\boldsymbol{v_c}$ at the center of mass and the speed of the driving car is given by $v_r$. Needed for this research are the relations between $\varphi$, $v_r$, $\boldsymbol{v_c}$ and $\theta$ which can be used to define the motion of a vehicle:

$$\begin{cases} \dot{\theta} = \omega_c \\ \dot{\boldsymbol{p}} = \boldsymbol{v_c} \end{cases} \tag{1}$$

where $\boldsymbol{p}$ is the position of the vehicle, $\theta$ is the angle between the global y axis and the vehicle and $\omega_c$ is the rotational velocity of the car around its center of rotation. Assumed is that the steering angle $\varphi$ cannot exceed $\varphi_{max}$ and the speed of the car $v_r$ cannot exceed $v_{max}$.

$$\varphi_c = \arctan\left(\frac{\tan(\varphi)}{2}\right) \tag{2}$$

gives the relation between the angle of the velocity of the vehicle and the steering angle. The relation between the speed and the magnitude of the velocity of the vehicle is given below.

$$v_c = \frac{\tan(\varphi)}{2\sin(\varphi_c)} \cdot v_r \tag{3}$$

$$\boldsymbol{v_c} = v_c \cdot \begin{pmatrix} \cos\left(\frac{1}{2}\pi - \theta - \varphi_c\right) \\ \sin\left(\frac{1}{2}\pi - \theta - \varphi_c\right) \end{pmatrix} \tag{4}$$

Using $v_c$ and $\varphi_c$ the velocity of the vehicle itself can be calculated as stated in Eq. 4. At last the rotation of the vehicle around its center of mass is calculated:

$$\omega_c = \frac{v_r}{l} \cdot \tan(\varphi) \tag{5}$$

Eq. 4 and Eq. 5 combined will describe the motion of the vehicle as stated in Eq. 1. (Full calculations and derivations can be found in Appendix A)

## III. Flocking algorithm

The boids algorithm described in subsection II-A has undergone some changes and additions to be applied in this research to support the motion of a vehicle as described in subsection II-B with the addition of the strict spatial constraints necessary for this research. Since the boids algorithm expects the vehicle to be able to accelerate in all directions and vehicles which move accordingly to the described kinematic model can not accelerate in all directions (accelerating in other directions than forward or backward can be done by steering, an action which makes certain ways of movement impossible) a conversion must be made from the acceleration given by the boids algorithm and the way the vehicle steers to achieve such acceleration. Furthermore rules for obstacle avoidance must be added to make it possible to let vehicles stay on a road.

Before adding things to the boids algorithm, the algorithm itself must be specified as given below. This sections shows the calculations to be done each time step (the algorithm is meant to be ran step by step) for each vehicle $i$ surrounding a vehicle (lets call this vehicle $\mathcal{V}$) where $r$ is the radius of perception, $r_{sep}$ is the smaller separation radius, $d_i$ is the distance between the vehicles $i$ and $\mathcal{V}$, $\boldsymbol{p}$ is the position of vehicle $\mathcal{V}$ itself and $\boldsymbol{q_i}$ is the position of vehicle $i$ within perception radius.

$$\boldsymbol{f_{c,i}} = \boldsymbol{q_i} - \boldsymbol{p} \tag{6}$$

This equation represents the cohesion factor and generates a vector pointing from vehicle $\mathcal{V}$ to vehicle $i$. To get the center of mass of all vehicles surrounding vehicle $\mathcal{V}$ all these vectors are averaged.

$$\boldsymbol{f_{a,i}} = \frac{\boldsymbol{v_{c,i}} + \boldsymbol{v'_{c,i}}}{2} \tag{7}$$

This equation represents the alignment factor. It takes the velocity of vehicle $i$ ($\boldsymbol{v_{c,i}}$) and is averaged with the desired velocity of vehicle $i$ ($\boldsymbol{v'_{c,i}}$). The desired velocity is the output of the algorithm of the previous time step. Taking this average will result in a more stable behaviour since it takes not only the current state of a vehicle into account but also takes in the near future of a vehicle.

$$\boldsymbol{f_{s,i}} = -e^{r_{sep}-d_i} \cdot \frac{\boldsymbol{q_i} - \boldsymbol{p}}{|\boldsymbol{q_i} - \boldsymbol{p}|} \tag{8}$$

This represents the separation factor. It generates a vector pointing from vehicle $i$ to vehicle $\mathcal{V}$ where its length is proportional to the distance between the vehicles. This relation between the length and the distance is negatively exponential and is chosen by trial and error (thus created the most stable behaviour compared to other relations as $1/d_i$ or a constant factor).

$$\boldsymbol{a_d} = \sum_{d_i \le r} \mathrm{w}_c \cdot \boldsymbol{f_{c,i}} + \sum_{d_i \le r} \mathrm{w}_a \cdot \boldsymbol{f_{a,i}} + \sum_{d_i \le r_{sep}} \mathrm{w}_s \cdot \boldsymbol{f_{s,i}} \tag{9}$$

combines the three parts of the boids algorithm which is used in further steps where $\mathrm{w}_c$, $\mathrm{w}_a$ and $\mathrm{w}_s$ are the specific weights of each part. For simplicity the mass of a vehicle is not incorporated into this equation which implies the mass is encapsulated in the weights. $\boldsymbol{a_d}$ is the desired acceleration of the vehicle. After combining all parts $\boldsymbol{a_d}$ is maximized to the maximum acceleration of the vehicle and the desired velocity can be altered as stated below. First define the clamp function, a function which clamps a value to a minimum and a maximum, as stated in Eq. 10 where $[\pm a]$ is the same as $[-a, a]$.

$$x\big|_{[a,b]} = \max(\min(x, b), a) \tag{10}$$

$$\Delta \boldsymbol{v'_c} = \boldsymbol{a_d}\big|_{[\pm a_{max}]} \tag{11}$$

The following part describes the calculations needed to obtain the variables required for the vehicle to move ($\varphi$ and $v_r$) from the output of the boids algorithm (the desired velocity $\boldsymbol{v'_c}$). The maximum steering angle $\varphi_{max}$ is converted to a maximum angle of the center-velocity ($\varphi_{c,max}$):

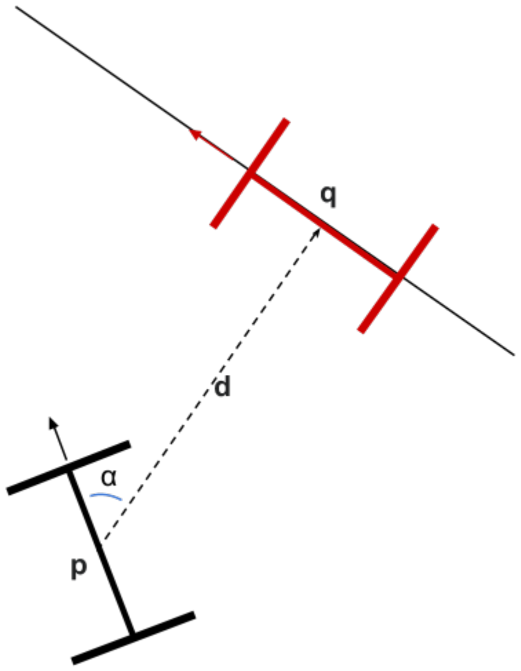


Fig. 3: Virtual vehicle on a line piece

$$\varphi_{c,max} = \arctan\left(\frac{\tan(\varphi_{max})}{2}\right) \tag{12}$$

Then the desired velocity is used to determine the new angle of the center-velocity with

$$\varphi_c = \operatorname{arctan2}(\boldsymbol{v'_{c,x}}, \boldsymbol{v'_{c,y}})\big|_{[\pm \varphi_{c,max}]} \tag{13}$$

The change of $\varphi_c$ can be clamped to emulate a maximum steering speed. From the new angle of the center-velocity the new steering angle can be calculated as stated below.

$$\varphi = \arctan(2 \cdot \tan(\varphi_c)) \tag{14}$$

The vehicle has a maximum speed ($v_r$) which is converted to a maximum magnitude of the velocity vector (Eq. 15) which is, in its turn, used to determine the new magnitude of the velocity in

$$v_{c,max} = v_{r,max} \cdot \frac{\tan(\varphi)}{2 \cdot \sin(\varphi_c)} \tag{15}$$

$$\boldsymbol{v_c} = v'_c\big|_{[\pm v_{v,max}]} \cdot \begin{pmatrix} \cos\left(\frac{1}{2}\pi - \varphi_c - \theta\right) \\ \sin\left(\frac{1}{2}\pi - \varphi_c - \theta\right) \end{pmatrix} \tag{16}$$

The new velocity vector is determined and the new speed of the vehicle is calculated as below:

$$v_r = v_c \cdot \frac{2 \cdot \sin(\varphi_c)}{\tan(\varphi)} \tag{17}$$

At this point there is enough information to use the kinematic model in combination with the boids algorithm. The next addition to the boids algorithm is the avoidance of objects. Within this research an object which must be avoided by all vehicles is constructed from line pieces. The distance between a vehicle and the line piece is calculated (calculation steps are shown in Appendix B). On the line piece a virtual vehicle is placed with the front-rear axis aligned with the line piece pointing in the same direction as the vehicle itself (see Figure 3) as is done in [3]. The same steps for the 'alignment' and 'separation' parts of the boids algorithm are applied. After this calculation $\boldsymbol{a_d}$ is clipped at the maximum velocity again and is used as output of the algorithm for the movement calculations. The equation below shows the total calculation needed per vehicle where $\boldsymbol{f'_{a,j}}$ and $\boldsymbol{f'_{s,j}}$ are the factors for the alignment and the separation calculated using the, on an obstacle placed, virtual vehicle $j$.

$$\boldsymbol{a_d} = \Bigg[ \Bigg[ \Bigg( \sum_{d_i \le r} \mathrm{w}_c \cdot \boldsymbol{f_{c,i}} + \sum_{d_i \le r} \mathrm{w}_a \cdot \boldsymbol{f_{a,i}} \\ \sum_{d_i \le r_{sep}} \mathrm{w}_s \cdot \boldsymbol{f_{s,i}} \Bigg) \Bigg|_{[\pm a_{max}]} + \sum_{d_j \le r} \mathrm{w}_c \cdot \boldsymbol{f'_{a,j}} \\ + \sum_{d_j \le r_{sep}} \mathrm{w}_a \cdot \boldsymbol{f'_{s,j}} \Bigg] \Bigg|_{[\pm a_{max}]} \tag{18}$$

## IV. Simulation implementation

To validate the algorithm explained in Section III for correctness and test it for its performance a certain setup is necessary. Since testing highly experimental algorithms on real life cars will probably not only be very expensive, it will be dangerous as well and in most countries prohibited (or extremely regulated) by law ([11]), the most logical, and probably only, choice would be testing the algorithm in a dedicated simulation environment. This part is split up in three sections: the choice of the simulation environment itself, a more in-depth look of the environment and an evaluation of performance of the simulation.

### A. Simulation environment

For this research a dedicated simulation environment is made in Python accompanied by OpenGL to evaluate the proposed algorithm. The reasons for this choice can be placed in the two separate categories expanded below:

- *Existing simulations*: Before writing an own simulation environment existing software packages which can be used to approach the problem should be considered. A traffic simulator as Veins (on top of OMNeT++ [12]) arises as possible solution, together with the simulation written for the previously done research on this topic [9]. The first one would have been a great environment for testing and evaluation if the proposed algorithm is known to work and an approach for its networking has come up. The latter is specifically written for evaluating flocking behaviour in autonomous driving traffic so it would have been a perfect solution apart from the fact that, since it does not make any use of parallel calculation, the simulation is relatively slow and running parameter sweeps will take many hours.
- *Programming environment*: Concluding from the point above an own simulation environment shall be written. The next question states 'in what?'. A big range of possibilities opens up as answer to this question such as writing it completely from the bottom up (in Java, C(++), Python, etc...) or taking an environment dedicated to mathematical operations such as MATLAB. Since the simulation written for the previous research [9] is written in Python, the thought of adding parallelism to the calculations should significantly speed up the simulation and the fact that I have more experience with Python than MATLAB the choice for Python was a clear one. With this choice arises one difficulty: multi-threading in Python is not directly supported. For support of parallelism OpenGL with its compute shaders (from version 4.3 and up) is taken with again experience in mind (and the added bonus of simulating on the GPU).

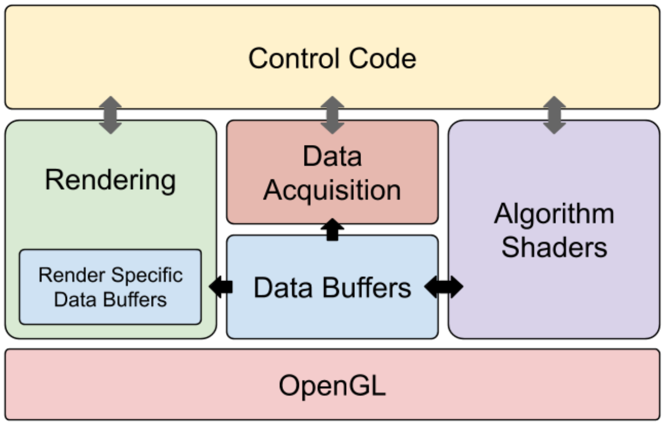


Fig. 4: Basic overview of simulation

### B. Implementation and design

Due to the choice of working with OpenGL the architecture of the simulation is in its core data-oriented. This resulted in the global architecture as seen in Figure 4. All calculations for the simulation are done in OpenGL compute shaders, programs which can run within the OpenGL context on a graphics card and are not directly linked to rendering graphics. Apart from the shaders, the data storage and the part which is responsible for rendering to the screen are connected to the OpenGL context as well. Creating said context and queuing specific functions is done by the control code in Python.

All the data for the simulation is stored on the graphics card as buffers. For each vehicle its position, rotation, velocity and its state within the simulation (collided or not and the time to enter the simulation) are stored as well as all the positions and normals of obstacles. Another big matrix holds the distance between two objects (vehicles and obstacles) as well as the

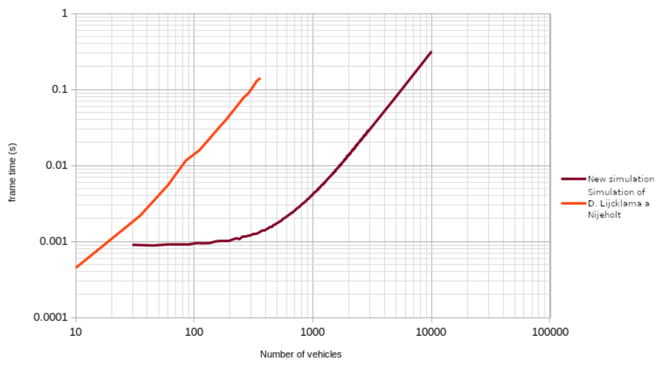


Fig. 5: Simulation time comparison

angle between them. Due to the nature of data buffers in OpenGL, the size of allocated data must be known beforehand. Randomly spawning vehicles is done by adding all vehicles to the simulation on start and setting a time at which they may appear. The shaders ignore all vehicles which have not appeared yet.

The control code will activate and run each part of the simulation in its turn. In broad sense the steps which are taken for each time step (or frame) are:

- Start the shader which calculates all distances between objects and wait for it to finish;
- Start the shader which executes the proposed algorithm and wait for it to finish;
- Start the shader which moves all vehicles one step and wait for it to finish;
- If rendering is enabled, render the next frame to the screen;
- If data acquisition is turned on, the control code retrieves data from the buffers on the graphics card

The split between the three shaders must be present and cannot be combined into one shader: The algorithm step needs the distances between all objects so these must be calculated first. The last step cannot be ran until the second step is completed since the movement of a vehicle is dependent on (practically) all other vehicles. Moving a vehicle when other vehicles are still executing the algorithm will possibly cause corruptions to appear in the used data.

### C. Evaluation of simulation

Within the process of creating the simulation environment a lot of testing and evaluation had to be done. This is done by placing a vehicle next to another vehicle or obstacle in known positions (situations in which all values stored in the data buffers are calculatable by hand) and altering the calculations until all stored values match known values. Since this had to be done at the same time as developing the algorithm and this process took a lot of iterations, all these steps are not included in this paper.

One of the main reasons for the written simulation environment was that it executed faster than the one existing

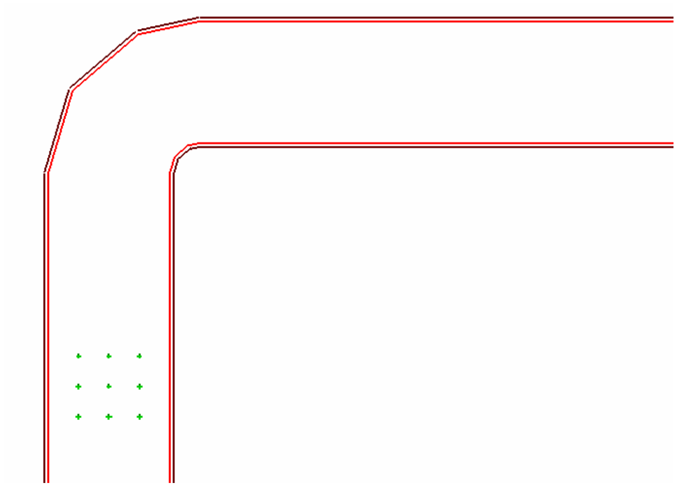


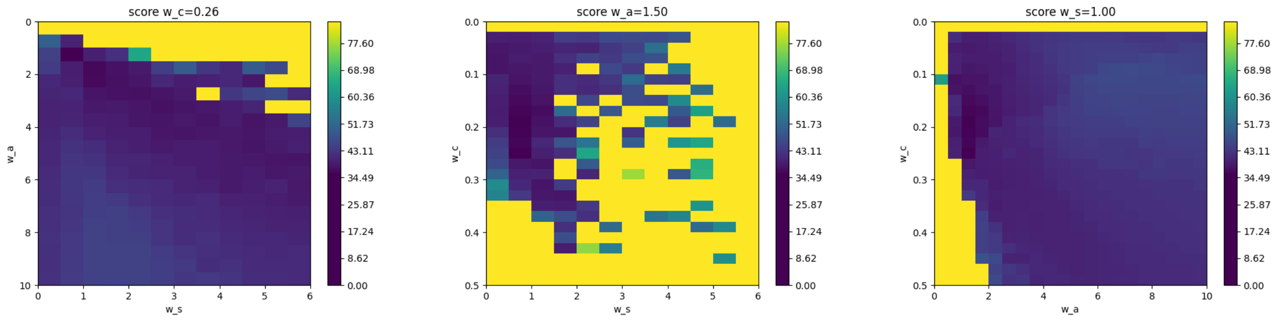
Fig. 6: Simulation setup of parameter sweep

environment. A test had to be done to verify if that goal was accomplished. Both the simulation of Lijcklama a Nijeholt and the new simulation are ran with a growing amount of vehicles just driving around. The total run time and amount of calculation steps are recorded and per amount of vehicles the time per step is averaged over four different runs. A note must be made that the existing simulation environment is not changed meaning that the implemented algorithm itself is not adapted to the here proposed algorithm. The time per step (or frame) is plotted against the amount of vehicles in the simulation, see Figure 5. As one can see the simulation environment written for this research is faster in all cases with a difference of two powers of ten with a large amount of vehicles which confirms the claim of being a more efficient simulation environment.

## V. Evaluation

This section will show the steps taken to validate and evaluate the proposed algorithm with matching results. Since testing the algorithm and simulation environment while developing one or the other is mostly done by visually assessing the renders of the simulation, jumping back and forth with a lot of trial and error, these steps are not included in this paper. Only the steps taken to evaluate the correctness and the performance are stated. This section is divided into several parts describing the process of finding the right parameter combinations and finding the capabilities of the algorithm in several situations: a road with a certain width, a 90° turn and a narrowing road.

For fair evaluation a measure for performance is needed. The measure chosen for the evaluation aimed at safety and stability and is calculated from the $\boldsymbol{f_{c,i}}$, $\boldsymbol{f_{a,i}}$ and $\boldsymbol{f_{s,i}}$ vectors and the amount of collisions. It is assumed that when a stable situation is reached, the length of these vectors is small relative to unstable situations and collisions did not happen. The length of these vectors is averaged over all vehicles present in the simulation, squared and summed over time (full calculation shown below in Eq. 19 where $N$ is the number of vehicles and $T$ is the number of time steps, the lower the score the better



(a) (b) (c)

Fig. 7: Score on all three axis on point $w_c = 0.26$, $w_a = 1.5$ and $s = 1.0$

the algorithm performs). The values are squared to count short unstable periods as well.

$$s(\mathrm{w}_c, \mathrm{w}_a, \mathrm{w}_s) = \sum_{i=1}^{T} \left[ \frac{\left( \sum_{j=1}^{N} |\boldsymbol{f_{c,j}}[i]| \right)^2}{N^2 \cdot T} + \frac{\left( \sum_{j=1}^{N} |\boldsymbol{f_{a,j}}[i]| \right)^2}{N^2 \cdot T} + \frac{\left( \sum_{j=1}^{N} |\boldsymbol{f_{s,j}}[i]| \right)^2}{N^2 \cdot T} \right] \tag{19}$$

### A. *Finding parameters*

The first step to be done is finding a valid combination of parameters which results in a stable behaviour of the vehicles. The parameters which could be altered are the weights of the *cohesion*, *alignment* and *separation* vectors $\mathrm{w}_c$, $\mathrm{w}_a$ and $\mathrm{w}_s$ combined with the radius of perception $r$ and the separation radius $r_{sep}$.

By visually assessing the behavior it is found that with a radius of perception of $15m$ and a separation radius of $10m$ a range containing valid parameter settings is at least $0 < \mathrm{w}_c < 0.5$, $0 < \mathrm{w}_a < 10$ and $0 < \mathrm{w}_s < 6$. Over this range a full parameter sweep is done in the situation depicted in Figure 6: 9 vehicles placed on a road in a grid with a bend in the road. The width of the road is $40m$, the distance between each vehicle is $10m$ and the inner radius of the turn is $10m$. The distance between the start of the turn and the first line of vehicles is $60m$, the starting velocity of the vehicles is $1ms^{-1}$, $v_{max} = 2.0ms^{-1}$ and $a_{max} = 2ms^{-2}$. At last $\varphi_{max}$ is set to be 37°. The simulation is ran for 900 steps with a time step of $16.6ms$ which results in a simulated time of $15s$. With each simulation step the lengths of $\boldsymbol{f_{c,i}}$, $\boldsymbol{f_{a,i}}$ and $\boldsymbol{f_{s,i}}$ are broken out of the simulation and are averaged over all 9 vehicles. For each combination of parameters these averaged lengths of the cohesion, alignment and separation vectors are used to calculate the performance score as shown above. The lower the score the better the combination performs. Combinations with collisions should be assigned the maximum score.

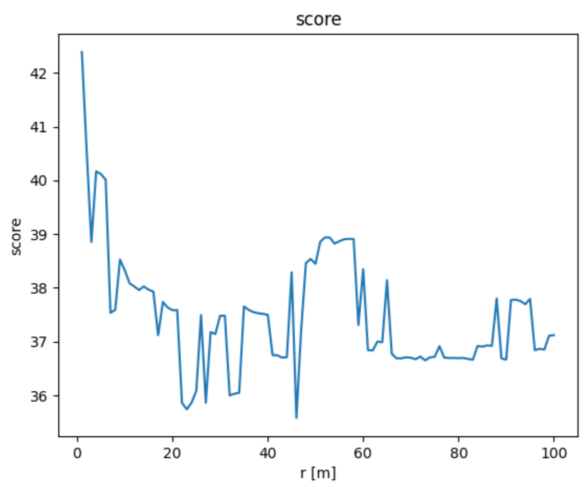


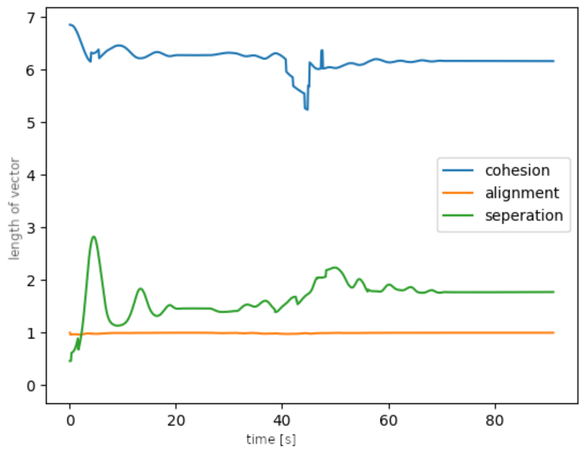
Fig. 8: Score of algorithm

Taking the complete data set generated by the parameter sweep it is found that the lowest score is with $\mathrm{w}_c = 0.26$, $\mathrm{w}_a = 1.5$ and $\mathrm{w}_s = 1$ as depicted in Figure 7a. These values will be used for the rest of the simulations. Figure 7 shows the performance scores of the parameter combinations surrounding the combination resulting in the lowest score. As one can see the range of parameter combinations which will result in stable behaviour is quite broad.

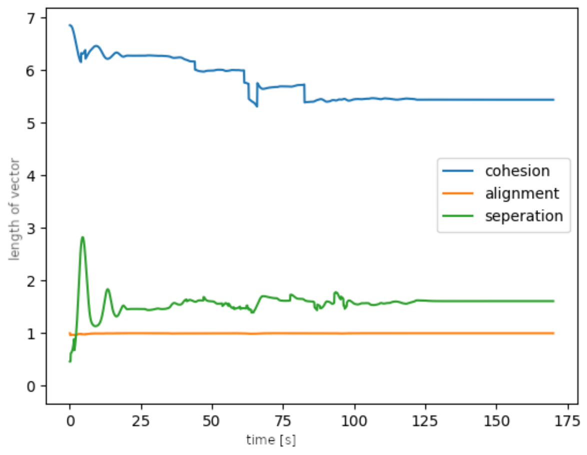
### B. *Radius of turn*

The next part of evaluating the algorithm is finding the minimum turn radius of a bend in the road with the parameters previously found. This is done in the same situation as previously used where the inner radius of the bend is changed between $0m$ and $100m$ where a radius of $0m$ indicates a sharp turn with right angles. The previously used performance score is applied to this simulation as well and provides view on how well the algorithm performs in the situation.

Figure 8 shows the performance score plotted against the radius of the turn. While no collisions are encountered at all,



(a) $r = 1m$



(b) $r = 80m$

Fig. 9: development of the cohesion, alignment and separation vectors over time in a turn

smaller radii clearly result in a less stable operation. Figure 9 shows that with a small turn radius the overall length of the cohesion and separation vectors are greater than with a larger turn radius. The relatively larger separation vectors suggest the vehicles are closer together after the turn and are left in a less ideal situation when the radius is small in comparison with a larger radius. As can be seen in Figure 9 the length of the alignment vectors are (relatively) constant. This is due to the fact that the alignment factor is independent of the distance between the vehicles and only depends on the velocities of other vehicles. Since all vehicles drive at roughly the same speed, the magnitude of the alignment vectors do not chagne significantly.

## C. *Width of road*

Testing the capabilities of the algorithm for different road widths is done as follow: on a straight road with width $w$ are

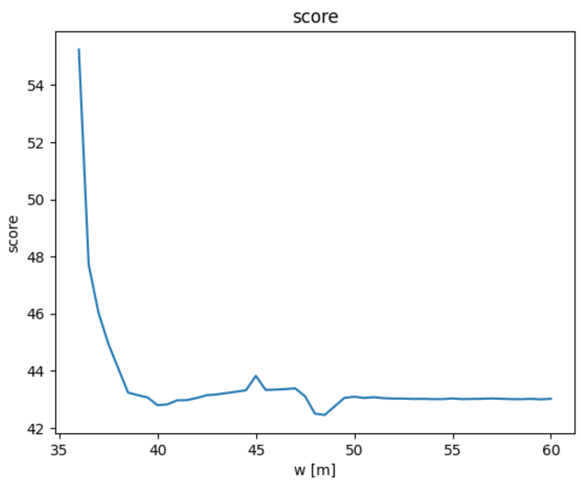


Fig. 10: Score of algorithm

placed nine vehicles in a grid formation (as is done in the previous evaluations). The distance between the vehicles is $d_s$ for situations where $w \geq 4d_s$ and $w/4$ for the other situations (to fit the three rows of vehicles on the road). For all widths between $20m$ and $60m$ the performance score is calculated and is shown in Figure 10.

As can be seen, for all widths larger than $4d_s$ the algorithm performs relatively well and for widths smaller than a few meters less than $4d_s$ collisions are encountered. Visually assessing those last situations it becomes clear that the collisions are caused by the initial positions of the vehicles: the vehicles start horizontally too close to each other and causes the separation part to take immediate effect, resulting in over steering and a collision is inevitable.

## D. *Narrowing road*

At last the effect of a road narrowing on the performance of the algorithm is tested. A road with a width of $40m$ is used which narrows down to $30m$ over a length of $l$. Three rows of three vehicles are placed on the road in the same manner as in previous evaluations. Since the three rows of vehicles may merge, the effect of 'phase difference' between the two outer lanes and the middle lane is measured as well. The middle row is moved forward from its original position to a full distance between the cars (and thus creating a $360°$ phase difference). Again, the performance score is calculated. Figure 11 shows the performance score graphed against the length of the road narrowing and the phase difference between the middle lane and the outer lanes (where a $ph = 0$ means all three lanes are exactly aligned and $ph = 1$ means a $360°$ phase difference). All the yellow squares depict a situation where collisions have occurred.

Visually assessing a few $l$ and $ph$ combinations will show that the three lanes are 'squished' together without merging into fewer lanes which implies vehicles do not merge with the current algorithm. The region containing low scores around a $180°$ phase shift could be explained with the fact that if

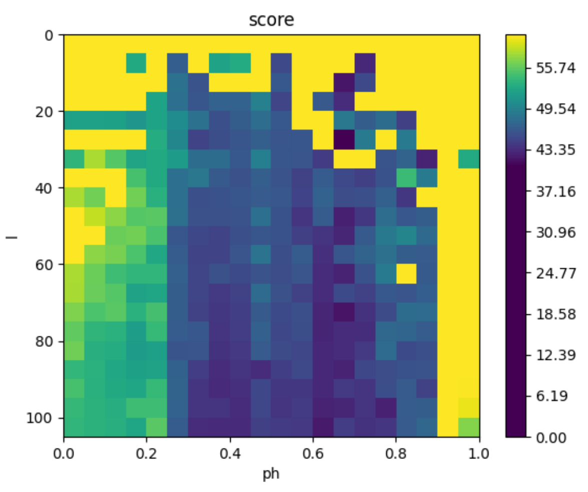


Fig. 11: Score of algorithm

there is roughly a 180° phase difference between the lanes the vehicles could move closer to each other thus creating a more narrow formation without being so close to each other that the separation factor of the algorithm is significantly large. Furthermore it is clear that longer road narrowings result in a relatively more stable behaviour.

A peculiar thing to mention is the fact that the region with the lowest scores is between a phase difference of 200° and 300° instead of an expected 180°. A possible cause could be the lack of vertical symmetry: when moving the middle lane forward there is an empty spot in the back which is not filled by another vehicle. Future research should be done to confirm this.

## VI. Conclusion and Future Work

The boids algorithm presented by Reynolds with the small alterations and additions proposed in this research provide a control strategy for Ackermann steered vehicles in relatively simple situations with strict spatial constraints. It is found that a wide range of parameter combinations result in relatively stable behaviour where a distinct best combination is found for the situation in which the algorithm is tested. Next to the proposed algorithm, this research provides the calculations needed to bridge the algorithm itself to the vehicle. At last an efficient simulation environment using parallelism on the graphical processor is written and it is shown to be faster than a previously used dedicated simulation environment.

The performance score used in the evaluation is found to be a reasonable measure for the stability of the algorithm hence an acceptable way to measure performance within this research. When comparing the algorithm to other existing algorithms other ways of measuring performance will be needed. By letting a vehicle project itself as a virtual vehicle on an obstacle and performing the alignment and separation steps to it, obstacles can be avoided and vehicles will stay on a road. Overall the usage of a flocking behaviour inspired control algorithm for autonomous driving vehicles looks to be promising but a lot more research has to be done to reach a safe control strategy which can be used in real live cars.

When doing this research a major difficulty did arise: The choice for OpenGL made the debugging of the simulation environment quite cumbersome: Simply attaching a debugger or printing out values between calculations is not possible and should be done in a complete different way (using stand-alone graphics debuggers which capture single frames). This made finding errors time consuming and difficult while constantly searching for the source of the problem: the simulation environment or the algorithm.

This research provides a way to implement flocking behaviour in autonomous driving vehicles but this is mere a basis: a lot of future research has to be done to reach a feasible and safe control strategy. A first addition on top of the current algorithm could be a way to let vehicles decelerate when a collision is inevitable. This will possibly open a way for different 'lanes' to merge without collisions. Apart from decelerating and accelerating the vehicles, the velocity at which the vehicles drive should be increased to a more realistic speed to examine if the provided algorithm still performs the way it does as in this research. To reach a usable control strategy destination driven behaviour should be added so that vehicles can unmerge after merging and diverge in the direction of its destination. The most simple way this could be achieved is a constantly present 'force' directed to its destination (as is done in [9]). A more sophisticated way, one which allows for a route not directly pointed towards the destination, could be a virtual vehicle which is placed on a pre-defined place on a road which follows a certain route and can be followed by the vehicle itself. At last the performance should be tested against real life situations and be compared to other control strategies for autonomous driving vehicles: narrower roads, highway intersections and large road crossings.

## References

[1] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 25–34, 8 1987. [Online]. Available: https://doi.org/10.1145/37402.37406

[2] R. O. Saber and R. M. Murray, "Flocking with Obstacle Avoidance: Cooperation with Limited Communication in Mobile Networks," Tech. Rep.

[3] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 3 2006.

[4] Khac Duc Do, "Flocking for Multiple Elliptical Agents With Limited Communication Ranges," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 931–942, 2011.

[5] W. J. Crowther, "Rule-based guidance for flight vehicle flocking," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 218, no. 2, pp. 111–124, 2004.

[6] "Boids On Wheels," Tech. Rep., 2016.

[7] Y. Hayashi and T. Namerikawa, "Flocking algorithm for multiple nonholonomic cars," in *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2016*. Institute of Electrical and Electronics Engineers Inc., 3 2016, pp. 1660–1665.

[8] Hu Cao, Jie Chen, Yutian Mao, Hao Fang, and Huagang Liu, "Formation control based on flocking algorithm in multi-agent system," in *2010 8th World Congress on Intelligent Control and Automation*, 2010, pp. 2289–2294.

[9] D. Lijcklama à Nijeholt, “Control for Cooperative Autonomous Driving Inspired by Bird Flocking Behavior,” *B.Sc Thesis, University of Twente*, 2020.

[10] W. C. Mitchell, A. Staniforth, and I. Scott, “Analysis of Ackermann Steering Geometry,” in *Motorsports Engineering Conference & Exposition*. SAE International, 12 2006. [Online]. Available: https://doi.org/10.4271/2006-01-3638

[11] Dutch Government, “Art 149 WVW, Stb. 2018, 347,” 2018.

[12] C. Sommer, R. German, and F. Dressler, “Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis,” *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, 1 2011.

## APPENDICES

### A. Kinematics

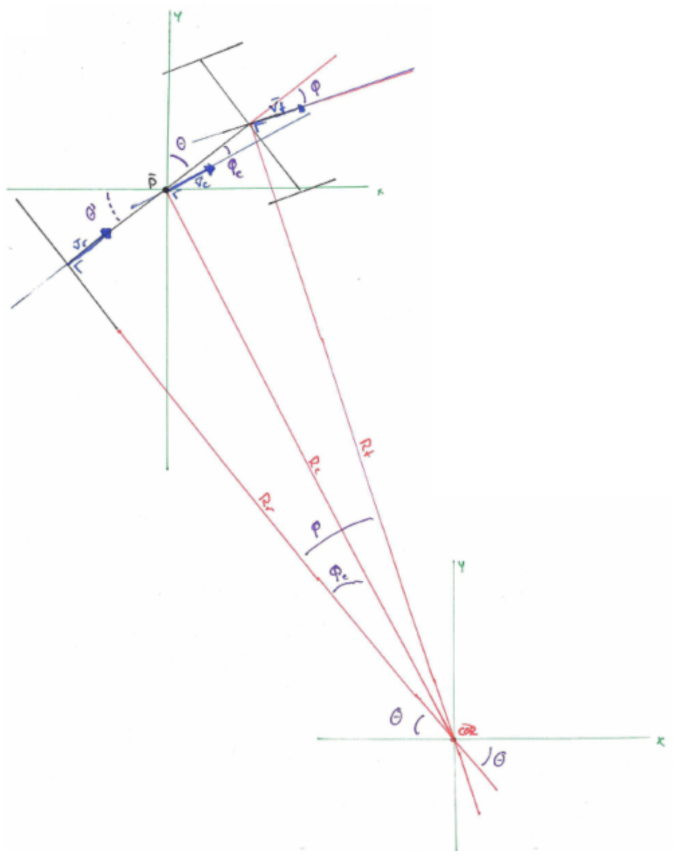


Fig. 12: Vehicle kinematics

Given that the angle between the axis of the vehicle and the front velocity $\boldsymbol{v_f}$ is $\varphi$ one can say that the angle between radii $R_r$ and $R_f$ is $\varphi$ as well. In the same way the angle between radii $R_r$ and $R_c$ is $\varphi_c$. Knowing this the following can be said:

$$\tan(\varphi) = \frac{l}{R_r} \tag{20}$$

$$\tan(\varphi_c) = \frac{\frac{1}{2}l}{R_r} \tag{21}$$

$$\sin(\varphi) = \frac{l}{R_f} \tag{22}$$

$$\tan(\varphi_c) = \frac{\frac{1}{2}l}{R_c} \tag{23}$$

Rewriting Eq. 20 and filling in into Eq. 21:

$$\varphi_c = \arctan\left(\frac{\tan(\varphi)}{2}\right) \tag{24}$$

$\boldsymbol{v_c}$ can be constructed from $\varphi_c$ and its own magnitude $v_c$ (note that $\theta$ is chosen to be measured from the y-axis):

$$\boldsymbol{v_c} = v_c \cdot \begin{pmatrix} \cos\left(\frac{1}{2}\pi - \theta - \varphi_c\right) \\ \sin\left(\frac{1}{2}\pi - \theta - \varphi_c\right) \end{pmatrix} \tag{25}$$

Driving in a circle means rotating around the center of rotation (COR) with a radius of $R_r$ from the rear axis. Due to symmetry one can say that the rotation around the COR is equal to the rotation around the center of the vehicle. This results in the rotational velocity around the COR is equal to the rotational velocity around the center of the vehicle and one can conclude:

$$\omega_c = \frac{v_c}{R_c} = \omega_r = \frac{v_r}{R_r} \tag{26}$$

This means that

$$v_c = \frac{v_r \cdot R_c}{R_r} \tag{27}$$

Filling in Eq. 20 and Eq. 23

$$v_c = v_r \cdot \frac{\tan(\varphi)}{2\sin(\varphi_c)} \tag{28}$$

Filling that in into Eq. 26

$$\omega_c = \frac{2 \cdot v_c}{l}\sin(\varphi_c) = \frac{v_r}{l}\tan(\varphi) \tag{29}$$

### B. Distance between point and line piece

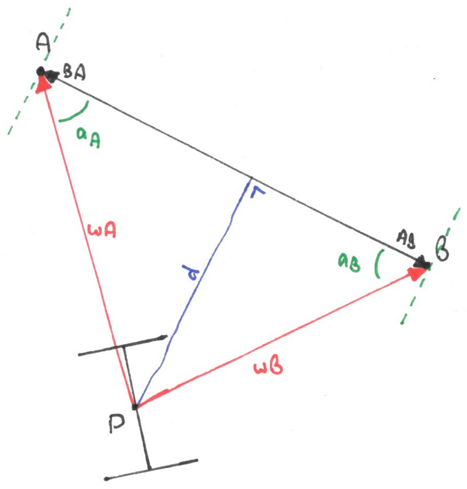
The distance between a point and a line piece is calculated as follow. One defines the vectors $\boldsymbol{AB}$ and $\boldsymbol{BA}$ as the vectors pointing from $A$ to $B$ and the other way around and the vectors $\boldsymbol{wA}$ and $\boldsymbol{wB}$ as the vectors pointing from the point $P$ to $A$ and $B$ respectively (see Figure 13a). The angles $\alpha_A$ and $\alpha_B$ are calculated as stated below:

$$\alpha_A = \arccos\left(\frac{\boldsymbol{wA} \cdot \boldsymbol{BA}}{|\boldsymbol{wA}| \cdot |\boldsymbol{BA}|}\right) \tag{30}$$

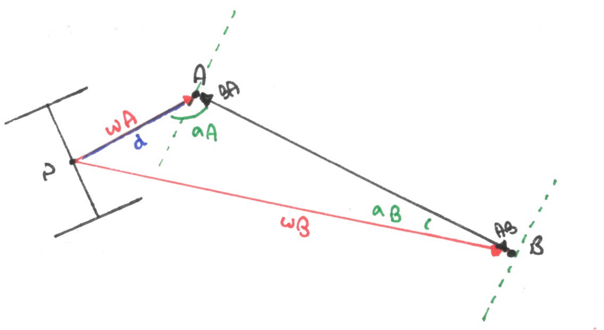
$$\alpha_B = \arccos\left(\frac{\boldsymbol{wB} \cdot \boldsymbol{AB}}{|\boldsymbol{wB}| \cdot |\boldsymbol{AB}|}\right) \tag{31}$$

When one of $\alpha_A$ or $\alpha_B$ is bigger then $\frac{\pi}{2}$ the distance to the line piece is equal to the vector pointing to the closest point ($\boldsymbol{w_A}$ or $\boldsymbol{w_B}$) as shown in Figure 13b. If both angles are smaller then $\frac{\pi}{2}$ as shown in Figure 13a the distance to the line piece is equal to the length of a line crossing the line piece at a right angle coming from the point.

$$d = \begin{cases} |\boldsymbol{wA}| & \alpha_A \geq \frac{\pi}{2}, \alpha_B < \frac{\pi}{2} \\ |\boldsymbol{wB}| & \alpha_A < \frac{\pi}{2}, \alpha_B \geq \frac{\pi}{2} \\ |\boldsymbol{wA}| \cdot \sin(\alpha_A) & \alpha_A < \frac{\pi}{2}, \alpha_B < \frac{\pi}{2} \end{cases} \tag{32}$$



(a)



(b)

Fig. 13: Situations for calculating distance between point and line piece