Navigation Function based Control of Dynamically Constrained Agents

Stockholm, Sweden
June – August 2012

Hanna Ohlsson hanna843@gmail.com
Mikaela Lokatt mlokatt@kth.se

Supervisors: Meng Guo
Dimos Dimarogonas

Abstract

Navigation functions can be used for navigation of agents in known as well as unknown environments. However, they are based on many assumptions which make it difficult to use the theory in a real application; as when navigating a car or a truck. This report gives ideas to some control algorithms that are based on navigation function theory but also vehicle dynamics, making them applicable in real scenarios. Most algorithms consider a point agent, while one algorithm takes into account the volume of the agent. Platooning scenarios are also studied where followers are supposed to follow a leader keeping a constant distance. Some controllers are proposed and simulation results when testing them in Matlab [8] are provided.

Acknowledgements

We would like to thank our supervisors Dr. Dimos V. Dimarogonas and PhD Candidate Meng Guo for giving us the opportunity to work with this project during the summer. It has been interesting, fun and we have learned a lot! We would also like to thank the Automatic Control Department for welcoming us and helping out in different ways. Last, but not least, we would like to thank our families and friends for their never ending support in life.

Again, thank you all!

Contents

1. Introduction

This report presents the ideas developed during nine weeks work at the Automatic Control Laboratory at the Royal Institute of Technology in Stockholm, Sweden. The aim was to investigate Navigation Function based control of single and multi-agent systems, where platooning was to be considered for the multi-agent scenario. The main idea was to control the motion of an agent to make it reach a goal position, without colliding with any obstacles on the way. The Navigation Function used was presented by D. E. Koditschek and E. Rimon in 1990 [1] and is built on a sphere world with point agents.

Two scenarios for the environment have been studied. In the first case an assumption was made that the environment was known from the beginning whereas in the other case the assumption was that the agent had information about the destination point and was able to detect obstacles within a certain sensing radius, but had no information about the rest of the world.

Both single integrator and unicycle dynamics have been studied, as well as the dynamics of a rear driven truck. Controllers based on the different dynamical models are proposed and simulation results when testing them in Matlab [8] are provided. Platooning of vehicles has also been studied and some controllers designed to make the following vehicles keep a certain distance to the vehicle in front are suggested. In addition, two controllers which are based on the navigation function but also influenced by other sources are given. The first one enables navigation of a non-point agent in a sphere world and the second smoothens out harsh turns which might appear when following the negated gradient of the navigation function.

Navigation Function

Navigation functions, NF, are a family of potential functions that can easily be transformed into control algorithms. They were presented for navigation of a point agent in a known sphere world by D. E. Koditscheck and E. Rimon in 1990 [1] and, since then, the theory has been extended to include both multiple- and non-point agents in known- as well as unknown environments, see for example [2],[3]. For a single point agent in a known environment, [1] defines the world β_0 as a sphere centered at the origin with radius ρ_0 and obstacles β_j as, non-intersecting, spheres with radii ρ_j and centers $\boldsymbol{q}_j$. The destination point for the agent is denoted $\boldsymbol{q}_d$ and the current position of the agent is denoted $\boldsymbol{q}$. A navigation function is defined as

$$\varphi = \frac{\gamma_d}{(\gamma_d^k + \beta)^{\frac{1}{k}}} \tag{1.1}$$

where

$$\gamma_d = \|\boldsymbol{q} - \boldsymbol{q}_d\|^2 \tag{1.2}$$

$$\beta \triangleq \prod_{j=0}^{M} \beta_j \tag{1.3}$$

$$\beta_0 = \rho_0^2 - \|\boldsymbol{q}\|^2; \qquad \beta_j = \left\|\boldsymbol{q} - \boldsymbol{q}_j\right\|^2 - \rho_j^2,\ j = 1 \dots M \tag{1.4}$$

and k is a positive constant. The function will take a global minimum value equal to zero at the destination point and global maximum values equal to one at the boundaries of the obstacles. If k is given a large enough value, no local minima will arise. This means that if letting an agent follow the

negated gradient, it will end up at the destination point from almost all initial positions, without colliding with any obstacles.

Figure 1 shows a surface plot of a NF in a two dimensional world with two obstacles and Figure 2 shows the contour lines of the same NF.

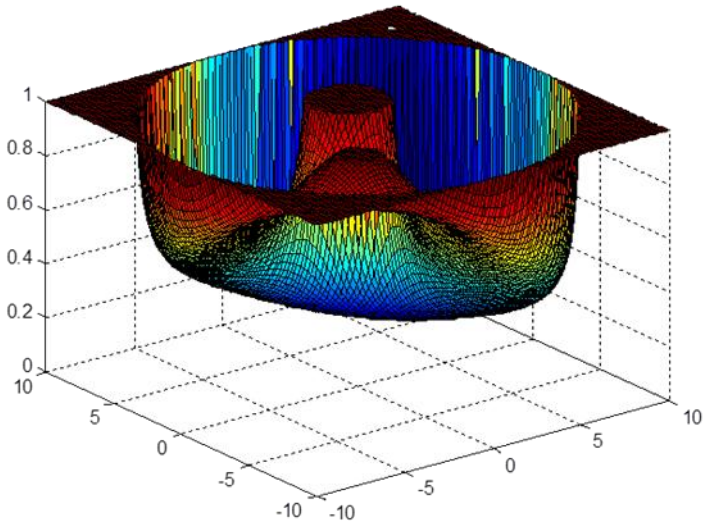

Figure 1. Surface plot of a NF in two dimensional world with two obstacles.

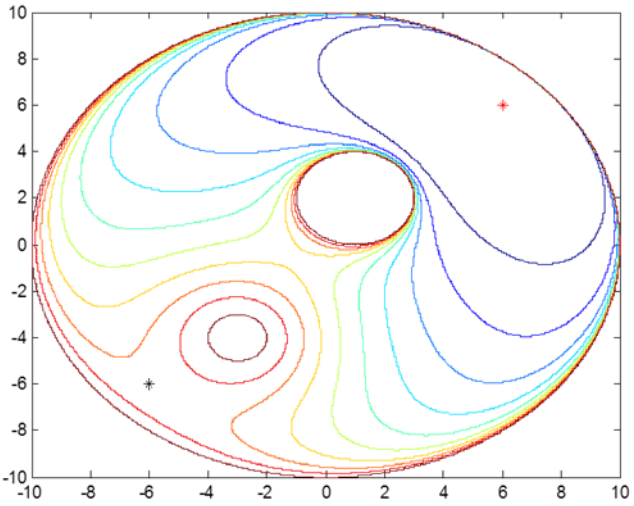

Figure 2. Contour lines of a NF in a two dimensional world with two obstacles. The red star marks the destination point and the black star marks an arbitrary starting position.

2. Single integrator Dynamics

The change in position of an agent following single integrator dynamics is described by the velocities in the different directions

$$\dot{\boldsymbol{q}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{2.1}$$

2.1. NF-based Controller

The agent was supposed to move in the direction of the negated gradient of the NF, keeping an arbitrary speed. The gradient was therefore calculated and the negated gradient was found by changing the sign of the expression. Multiplying the negated gradient with an arbitrary constant made it possible to control the magnitude of the speed. The gradient of the NF was found to be

$$\boldsymbol{\nabla}\varphi = \frac{2\beta\gamma_d}{\left(\gamma^k+\beta\right)^{\left(\frac{1}{k}+1\right)}} \cdot \left(\frac{1}{\gamma_d}(\boldsymbol{q}-\boldsymbol{q}_d) - \frac{1}{k}\sum_{j=0}^{M} \frac{\boldsymbol{q}-\boldsymbol{q}_j}{\beta_j} \right) \tag{2.2}$$

giving the controller

$$\dot{\boldsymbol{q}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = -B \cdot \left(\frac{1}{\gamma_d}(\boldsymbol{q}-\boldsymbol{q}_d) - \frac{1}{k}\sum_{j=0}^{M} \frac{\boldsymbol{q}-\boldsymbol{q}_j}{\beta_j} \right) \tag{2.3}$$

where B is an arbitrary positive constant.

2.2. Known Environment

The world was considered to be two dimensional and spherical and defined as in [1]. The agent had information about the destination point as well as the positions of all of the obstacles. The information was put into equation (2.3) giving a controller for a known environment.

The result when simulating the trajectory of a point agent controlled by (2.3) in Matlab [8] is shown in Figure 3.

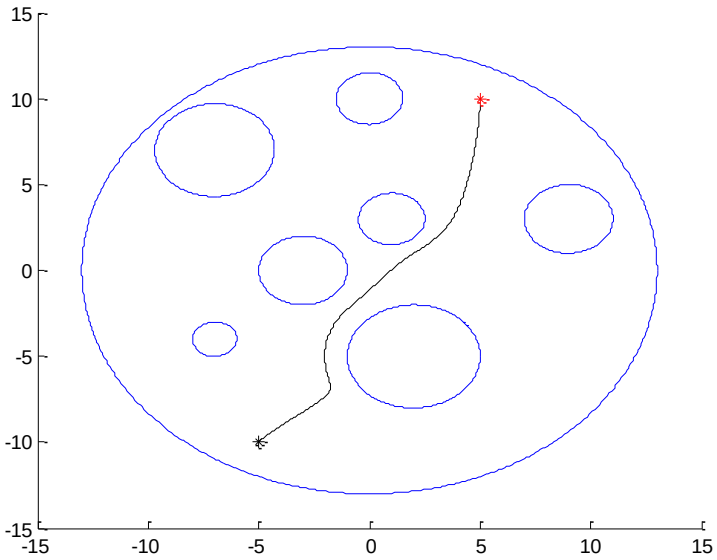

Figure 3. The path of an agent with single integrator dynamics moving from the black star to the red star, following the negated gradient of the NF. The tuning parameter of the NF was set to $k = 20$.

2.3. Unknown Environment

For the navigation in an unknown environment two different approaches based on the NF (1.1) and ideas from [3], [4] were used. The world was considered to be two dimensional and spherical and the agent was assumed to have a circular sensing zone within which it could detect obstacles. In the first approach the NF was built on information about the destination point and the obstacles currently within the sensing zone, i.e. the obstacles currently within the sensing zone were included in the β-part of the NF (1.3). The NF was updated every time an obstacle entered or left the zone and the desired path was found from the negated gradient (2.1). The second approach was similar, the only difference that the NF was based on information about obstacles which had been detected earlier as well as information about the obstacles currently within the sensing zone.

Figure 4 shows the path of a point agent only using the information about the obstacles currently within the sensing zone. When instead using the information about all the detected obstacles the trajectory in Figure 5 was obtained.

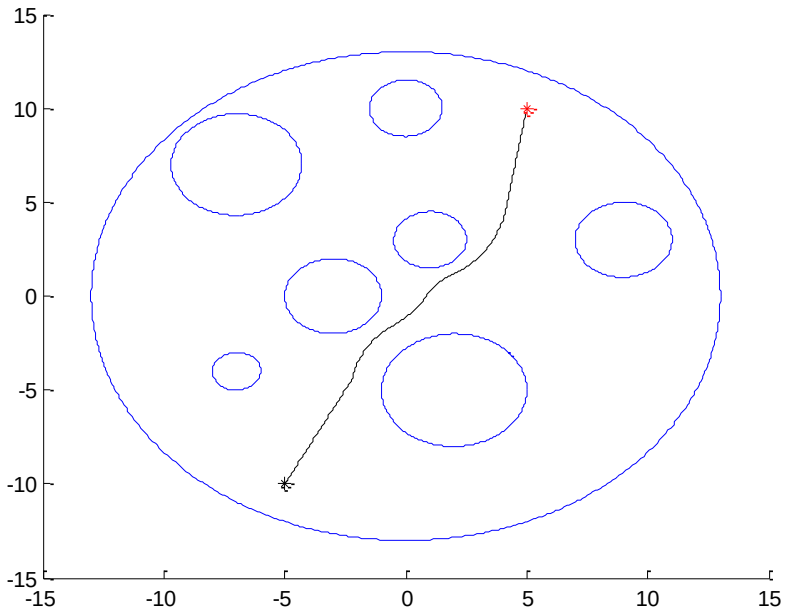

Figure 4. The path of an agent with single integrator dynamics following the negated gradient of the NF in an unknown world. All the obstacles are shown in the plot, but only the information about the obstacles currently within the agent's sensing zone is used in the NF. The black star marks the starting position for the agent and the red star marks the destination point. The tuning parameter for the NF was set to $k = 25$ and the sensing radius to $r_{sense} = 3$.

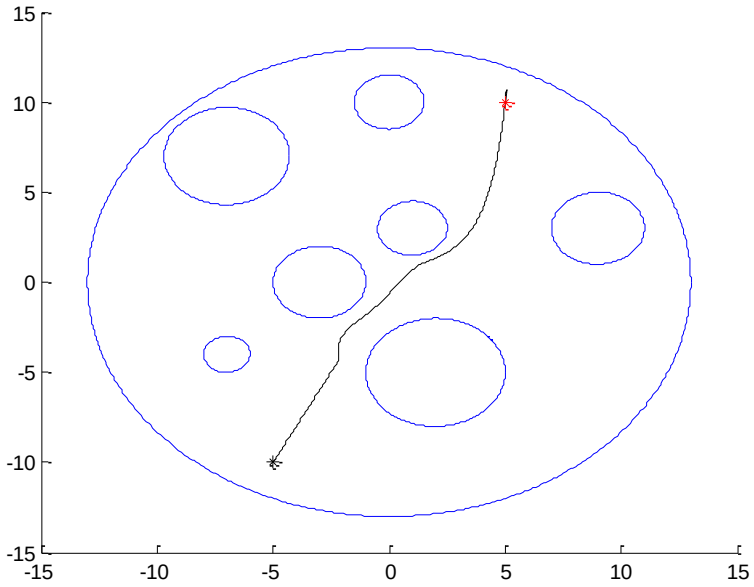

Figure 5. The path of an agent with single integrator dynamics following the negated gradient of the NF in an unknown world. All the obstacles are shown in the plot, but only the information about the detected obstacles is used in the NF. The black star marks the starting position for the agent and the red star marks the destination point. The tuning parameter for the NF was set to $k = 25$ and the sensing radius to $r_{sense} = 3$.

3. Unicycle Dynamics

The dynamics of unicycle type agents is usually described by the following model

$$\dot{\boldsymbol{q}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{pmatrix} \tag{3.1}$$

where $\boldsymbol{q} = (x, y, \theta)^T$ denotes the position and orientation of the agent. If assigning constraints to the angular velocity ω and speed v the motion of a unicycle type agent becomes more restricted than that of a single integrator type agent. This means that if using the NF for the navigation of a unicycle type agent, the constraints might prevent the agent from following the negated gradient.

3.1 NF-based Controller

In the unicycle case the negated gradient of the NF is not used as controller input, but for directional guidance. The inputs are based on the orientation of the negated gradient and the controller design ensures that the outputs are bounded $|\omega| \leq \omega_{max}$ and $0 < v \leq v_{max}$. The idea for the proposed controller is that ω and v should be adapted to the curvature of the path. If the change in direction is large, i.e. a harsh turn, ω should be big and v should be small and vice-versa.

The following controller is proposed for a unicycle-type agent

$$\omega = \begin{cases} \frac{\omega_{max}}{\pi} \cdot \Delta\theta, & |\Delta\theta| < \pi \\ -sign(\Delta\theta) \cdot \frac{\omega_{max}}{\pi} \cdot (2\pi - |\Delta\theta|), & |\Delta\theta| \geq \pi \end{cases} \tag{3.2}$$

where $\Delta\theta = \theta_d - \theta_c$ is the difference between the desired direction θ_d and the current direction θ_c. θ_d is found from

$$\theta_d = \begin{cases} atan2\left(-\frac{\partial\varphi}{\partial y}, -\frac{\partial\varphi}{\partial x}\right) & , y \geq 0 \\ atan2\left(-\frac{\partial\varphi}{\partial y}, -\frac{\partial\varphi}{\partial x}\right) + 2\pi & , y < 0 \end{cases} \tag{3.3}$$

$$v = \begin{cases} v_{max} \cdot e^{\left(\frac{-5|\Delta\theta|}{\pi}\right)}, & |\Delta\theta| < \pi \\ v_{max} \cdot e^{\left(\frac{-5(2\pi - |\Delta\theta|)}{\pi}\right)}, & |\Delta\theta| \geq \pi \end{cases} \tag{3.4}$$

3.2 Known Environment

The same method as for the single integrator model was used, see page (6). Figure 6 shows the trajectory of an agent using the controller proposed for a unicycle type agent (3.2) – (3.4).

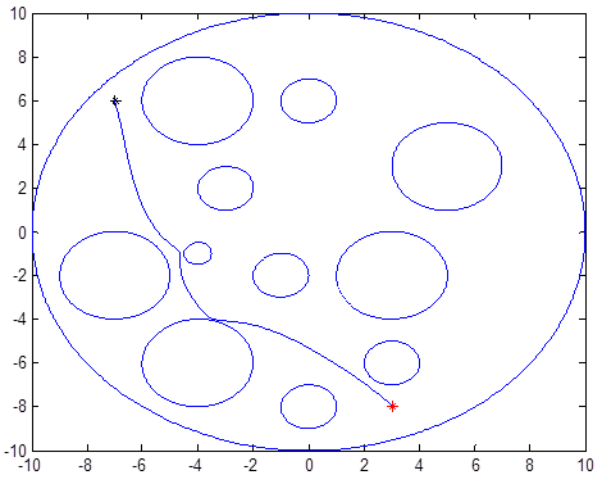

Figure 6. The path of an agent with unicycle dynamics controlled by a unicycle controller.

3.3 Unknown Environment

The method was built on a NF using information about all obstacles which had been detected by an agent, as described in the second approach for navigation of a single integrator type agent in an unknown environment on page (7). Figure 7 shows the trajectory of an agent controlled by the proposed unicycle type controller (3.2) – (3.4).

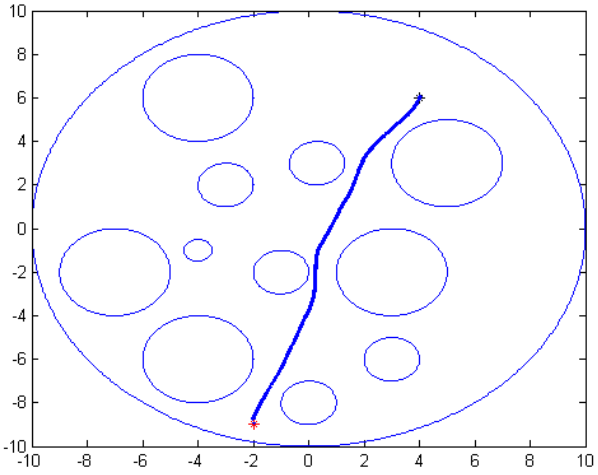

Figure 7. Trajectory of unicycle type agent in an unknown environment. All of the obstacles are shown in the plot, but only the information about the detected ones is used in the NF. The tuning parameter for the NF was set to $k = 30$ and the sensing radius to $r_{sense} = 1$.

4. Application

The maneuverability of real vehicles is restricted due to dynamical constraints. This has to be taken into account when applying the NF theory to different types of vehicles. This section presents the modeling of a truck and related controller designs. Autonomic control can also be used for platooning of vehicles and in the end of this section some platooning algorithms, based on the truck dynamics, are presented.

4.1. Truck Modeling

A dynamical model describes the behavior of a system and is thus necessary in order to simulate its behavior. In this section, two different dynamical models are proposed; one of a rear driven truck predicting the movement of the center point of the rear axis and one of a rear driven truck predicting the movement of the center of gravity. The second model can also be adapted to predict the movement of the center of gravity of a front driven truck.

4.1.1 Dynamic Model 1

In Dynamical model 1 it is assumed that the motion of a rear driven truck can be described by the motion of the center point of the rear axis. The model is based on [6, pages 722-726] and Figure 8 shows the variables used to describe the truck dynamics.

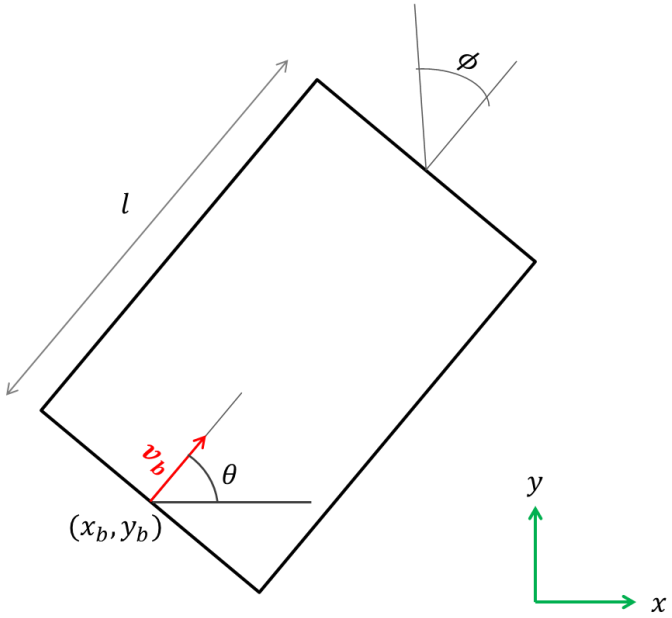

Figure 8. The notations used in Dynamical model 1.

The following dynamical model is proposed

$$\frac{\partial}{\partial t}\begin{pmatrix} x_b \\ y_b \\ \dot{x}_b \\ \dot{y}_b \\ \theta \\ \phi \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}\dot{\phi} + \begin{pmatrix} 0 \\ 0 \\ \cos(\theta) \\ \sin(\theta) \\ 0 \\ 0 \end{pmatrix} a_b + \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \\ 0 \\ \frac{\tan(\phi)}{l} \\ 0 \end{pmatrix} v_b \qquad (4.1)$$

where x_b and y_b denotes the spatial coordinates for the center point of the rear axis, $v_b = \sqrt{\dot{x}_b^2 + \dot{y}_b^2}$ and $a_b = \sqrt{\ddot{x}_b^2 + \ddot{y}_b^2}$. l is the distance between the rear and the front axis, θ is the orientation of the truck and ϕ is the steering angle.

4.1.2 Dynamical Models 2 and 3

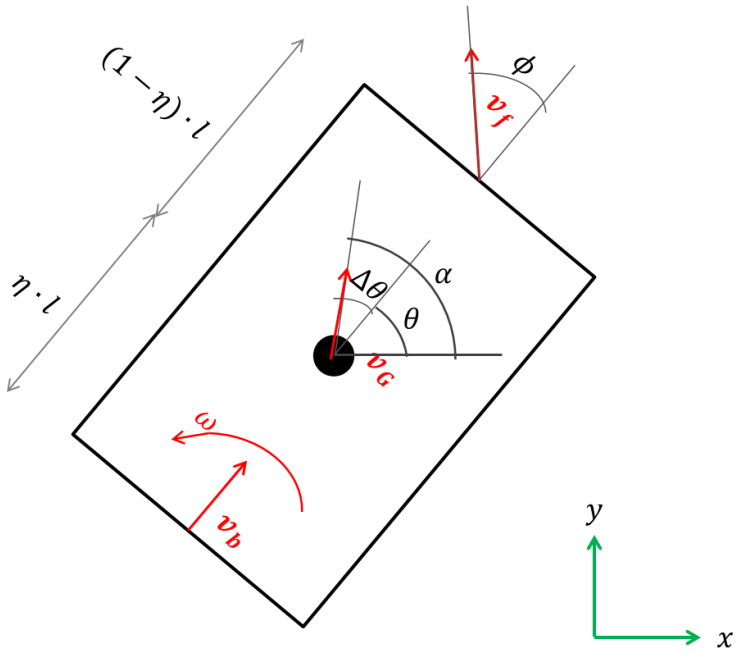

Figure 9. The notations used in Dynamical model 2 and Dynamical model 3.

Figure 9 describes a truck modeled as a rigid body. $\boldsymbol{v}_f, \boldsymbol{v}_b$ are the velocities of the center of the front and of the back of the truck respectively and $\boldsymbol{\omega}$ is the rotational velocity of the body. $\boldsymbol{v}_G$ is the velocity of the center of gravity and θ represents the orientation of the truck. η is a constant that can take values between $0 \leq \eta \leq 1$ depending on the position of the center of gravity. A rear driven truck is steered by controlling the orientation of the front wheels and the speed of the back wheels and the inputs to the dynamical system are thus ϕ, v_b

$$\frac{\partial}{\partial t}\begin{pmatrix} x_G \\ y_G \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{1+\eta^2\tan^2(\phi)} \cdot \cos(\theta + \tan^{-1}(\eta \cdot \tan(\phi))) \\ \sqrt{1+\eta^2\tan^2(\phi)} \cdot \sin(\theta + \tan^{-1}(\eta \cdot \tan(\phi))) \\ \frac{1}{l}\tan(\phi) \end{pmatrix} v_b \qquad (4.2)$$

The observable outputs are $x_G, y_G, \dot{x}_G, \dot{y}_G, \theta$

If a truck is front driven the speed of the front wheels is controlled instead of the speed of the back wheels. v_b in the model above is then replaced by v_f giving

$$\frac{\partial}{\partial t}\begin{pmatrix} x_G \\ y_G \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{1-(1-\eta^2)\sin^2(\phi)} \cdot \cos(\theta + \tan^{-1}(\eta \cdot \tan(\phi))) \\ \sqrt{1-(1-\eta^2)\sin^2(\phi)} \cdot \sin(\theta + \tan^{-1}(\eta \cdot \tan(\phi))) \\ \frac{1}{l}\sin(\phi) \end{pmatrix} v_f \quad (4.3)$$

Since the dynamical models are based on a velocity and a steering angle, some problems might arise if using them in a real application. This is because the assumption that the speed and the steering angle can be controlled directly might demand very high values of the acceleration and the angular velocity. If the system is fast responding it will not be a problem and the relatively simple model (4.2) can then be used. Otherwise the acceleration and angular velocity has to be taken into account and from (4.2) another model can be derived which considers the said properties taking $\dot{\phi}$, a_b as inputs

$$\frac{\partial}{\partial t}\begin{pmatrix} x_G \\ y_G \\ \dot{x_G} \\ \dot{y_G} \\ \alpha \\ \phi \end{pmatrix} = \begin{pmatrix} v_b\sqrt{1+\eta^2\tan^2(\phi)}\cos(\alpha) \\ v_b\sqrt{1+\eta^2\tan^2(\phi)}\sin(\alpha) \\ \cos(\alpha)\left(a_b\sqrt{1+\eta^2\tan^2(\phi)} + v_b\frac{\eta^2\tan^2(\phi)}{\sqrt{1+\eta^2\tan^2(\phi)}}\frac{\dot{\phi}}{\cos^2(\phi)}\right) + v_b\sin(\alpha)\sqrt{1+\eta^2\tan^2(\phi)}\left(\frac{\eta\dot{\phi}}{\cos^2(\phi)+\eta^2\sin^2(\phi)} + \frac{v_b}{l}\tan(\phi)\right) \\ \sin(\alpha)\left(a_b\sqrt{1+\eta^2\tan^2(\phi)} + v_b\frac{\eta^2\tan^2(\phi)}{\sqrt{1+\eta^2\tan^2(\phi)}}\frac{\dot{\phi}}{\cos^2(\phi)}\right) + v_b\cos(\alpha)\sqrt{1+\eta^2\tan^2(\phi)}\left(\frac{\eta\dot{\phi}}{\cos^2(\phi)+\eta^2\sin^2(\phi)} + \frac{v_b}{l}\tan(\phi)\right) \\ \frac{\eta\dot{\phi}}{\cos^2(\phi)+\eta^2\sin^2(\phi)} + \frac{v_b}{l}\tan(\phi) \\ \dot{\phi} \end{pmatrix} \quad (4.4)$$

$x_G, y_G, \dot{x}_G, \dot{y}_G, \theta, \phi$ are observable outputs and α can be estimated from $\alpha = \theta + \tan^{-1}(\eta \cdot \tan(\phi))$.

4.2. Controller Design

Some controller designs based on the dynamical models of a truck are presented.

4.2.1. Controller Design for Dynamic Model 1

For Dynamical model 1 the following controller is proposed

$$\dot{\phi} = sign(\Delta\phi)\sqrt{\dot{\phi}_{max}{}^{2} \cdot \frac{|\Delta\phi|}{\phi_{max}}} \tag{4.5}$$

where $\dot{\phi}$ is the angular velocity, ϕ_{max} is the maximal steering angle and

$$\Delta\phi = \phi_d - \phi_c, \qquad |\Delta\phi| \leq 2 \cdot \phi_{max} \tag{4.6}$$

where ϕ_d is the desired steering angle and ϕ_c is the current steering angle. The acceleration a_b is controlled by a PID controller based on the speed taking

$$v_b = v_{b,max} \cdot e^{\left(\frac{-5|\Delta\phi|}{\phi_{max}}\right)} \tag{4.7}$$

as reference value, where $v_{b,max} \geq 0$ is an upper bound for the speed.

Figure 10 shows a block diagram describing the controlled system and the controllers.

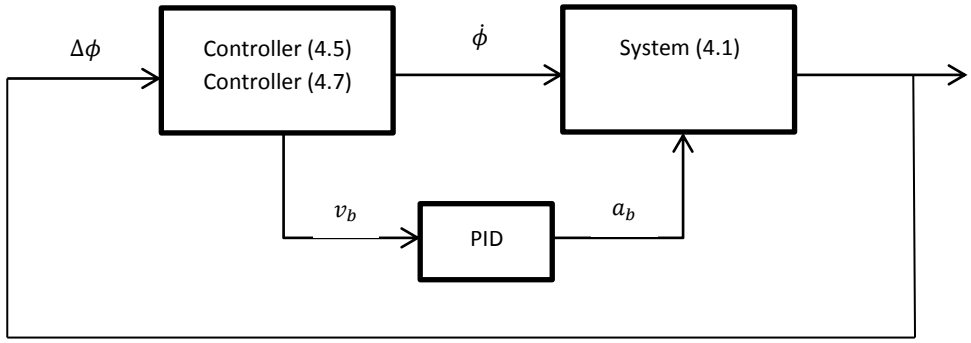

Figure 10. A block diagram of Dynamical model 1 and the corresponding controllers.

4.2.2. Controller Design for Dynamic Model 2

The velocity of the back wheels and the orientation of the front wheels is controlled using information about the wanted motion of the center of gravity. The wanted velocity v_{want} and the wanted direction of movement θ_{want} are given as controller inputs together with information about the current orientation of the truck's body θ

$$\phi = \tan^{-1}\left(\frac{1}{\eta} \cdot \tan(\Delta\theta)\right) \tag{4.8}$$

$$v_b = \frac{v_{want}}{\sqrt{1+\tan^2(\Delta\theta)}} \tag{4.9}$$

$$v_f = v_{want} \cdot \frac{\sqrt{1+\frac{1}{\eta^2}\tan^2(\Delta\theta)}}{\sqrt{1+\tan^2(\Delta\theta)}} \tag{4.10}$$

where $\Delta\theta = \theta_{want} - \theta$. The steering angle ϕ has a maximum value $|\phi_{max}| \leq \frac{\pi}{2}$ which constrains the maximum value of $|\Delta\theta|$ that can be achieved

$$|\Delta\theta|_{max} = \tan^{-1}\left(\frac{1}{2}\tan(|\phi_{max}|)\right) \tag{4.11}$$

For the system to be able to follow the control input it is thus necessary that

$$\Delta\theta \leq |\Delta\theta|_{max}$$

4.2.2. Controller Design for Dynamic Model 3

Based on Dynamical model 3 (4.4) a controller can be found that provides acceleration for the back wheels and angular velocity for the front wheels from information about the wanted motion of the center of gravity. The controller also uses information about the current velocity of the back wheels and the current steering angle.

$$\dot{\phi} = \begin{cases} \left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\frac{(\cos^2(\phi)+\eta^2\sin^2(\phi))}{\eta}, & \left|\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\frac{(\cos^2(\phi)+\eta^2\sin^2(\phi))}{\eta}\right| < \dot{\phi}_{max} \\ sign\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\cdot\dot{\phi}_{max}, & \left|\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\frac{(\cos^2(\phi)+\eta^2\sin^2(\phi))}{\eta}\right| \geq \dot{\phi}_{max} \end{cases} \tag{4.12}$$

$$a_b = \begin{cases} \frac{a_{g,want}}{\sqrt{1+\eta^2\tan^2(\phi)}} - v_b\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\eta\tan(\phi), & \left|\frac{a_{g,want}}{\sqrt{1+\eta^2\tan^2(\phi)}} - v_b\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\eta\tan(\phi)\right| < a_{b,max} \\ sign\left(\frac{a_{g,want}}{\sqrt{1+\eta^2\tan^2(\phi)}} - v_b\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\eta\tan(\phi)\right)\cdot a_{b,max}, & \left|\frac{a_{g,want}}{\sqrt{1+\eta^2\tan^2(\phi)}} - v_b\left(\dot{\alpha}_{want} - \frac{v_b}{l}\tan(\phi)\right)\eta\tan(\phi)\right| \geq a_{b,max} \end{cases} \tag{4.13}$$

where $a_i = \dot{v}_i$ and the subscript $want$ denotes the desired value for the center of gravity.

4.3. Platooning

Two or more vehicles traveling in a line after each other is called platooning. If the distance between the vehicles is relatively short the drag is reduced for the vehicles, leading to lower fuel consumption [7]. Two different platooning approaches are proposed where the vehicles follow the truck dynamics described in section (4.1).

4.3.1. Platooning Approach 1

The platooning approach is built on Dynamical model 1 (4.1) and each vehicle is treated as a separate system.

Leader

The leader is navigated using the controller proposed for dynamical model 1 (4.5) – (4.7).

Follower

For the followers, the distance to the vehicle in front d_{bf} and the angle between the follower's orientation and the desired orientation σ, see Figure 11, are controlled by PID controllers. The controller for d_{bf} gives v_b as output and the controller for σ gives $\dot{\phi}$ as output. Two different scenarios are tested. In the first scenario only the leader knows about the positions of the obstacles whereas in the other scenario also the followers have information about the obstacles. In the first scenario the desired orientation is always sett towards the back of the vehicle in front. For scenario number two another approach is used; when a follower is within a certain distance from an obstacle the desired direction is no longer set towards the back of the leader, instead the desired direction is found by an approach described later in the section about avoiding harsh turns, see page (27). As soon as the follower has passed the obstacle the desired direction is once again set to the back of the vehicle in front. Figure 12-13 shows a simulation of scenario one with three vehicles, i.e. when only the leader has informatio about the obstacles, and Figure 14 shows the distance between the vehicles. Figure 15 shows the trajectories for the platooning vehicles in scenario two, i.e. when also the followers have information about the obstacles.

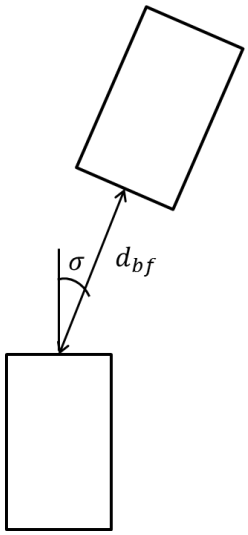

Figure 11. Notations used in Platooning approach 1.

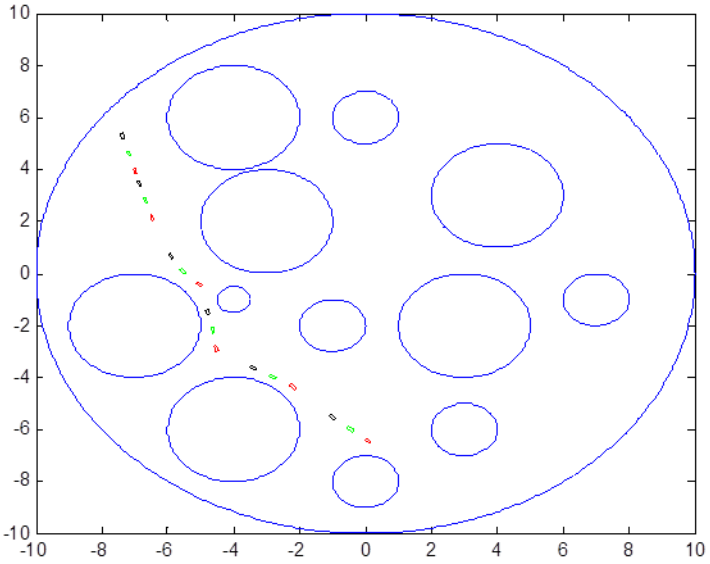

Figure 12. Simulation of platooning with three vehicles when only the leader has information about the obstacles. The red truck is the leader and the green and black ones are the followers.

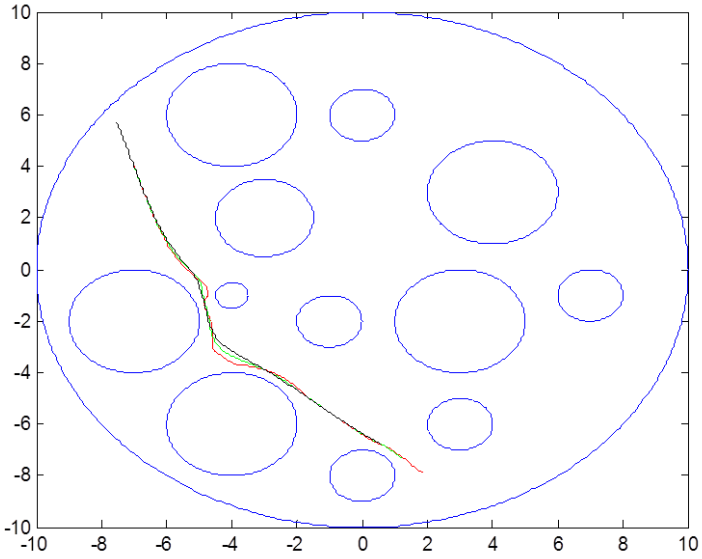

Figure 13. Trajectories for platooning with three vehicles when only the leader has information about the obstacles. Red trajectory represents the leader, green the first follower and black the second follower.

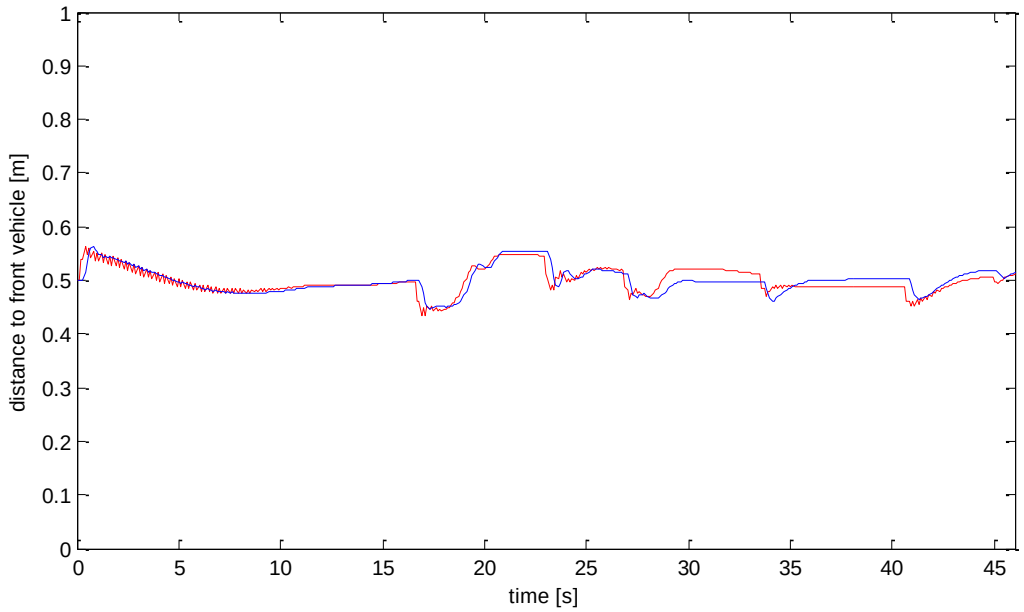

Figure 14. Distance between vehicles when only the leader has information about the obstacles. The red line represents the distance between the leader and the first follower. The blue line represents the distance between the first follower and the second follower. The reference distance is set to 0.5 m.

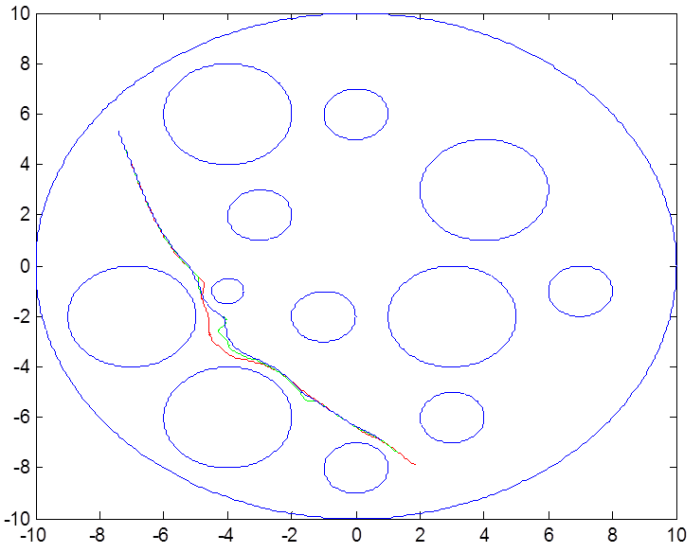

Figure 15. Trajectories for platooning with three vehicles when all vehicles have information about the obstacles. Red trajectory represents the leader, green the first follower and black the second follower.

4.3.2. Platooning Approach 2

The leader follows an arbitrary trajectory, in this case based on the NF, and is controlled by the controller proposed for Dynamical truck model 2 (4.8) – (4.9). Each follower keeps track of the vehicle in front of it, i.e. either the leader or one of the other followers, and remembers the positions of that vehicle. A controller based on the truck model and some linear algebra is then used to make the follower follow the same trajectory as closely as possible while keeping a certain distance to the vehicle in front.

Figure 16 describes a platooning scenario for a leader L and a follower F.

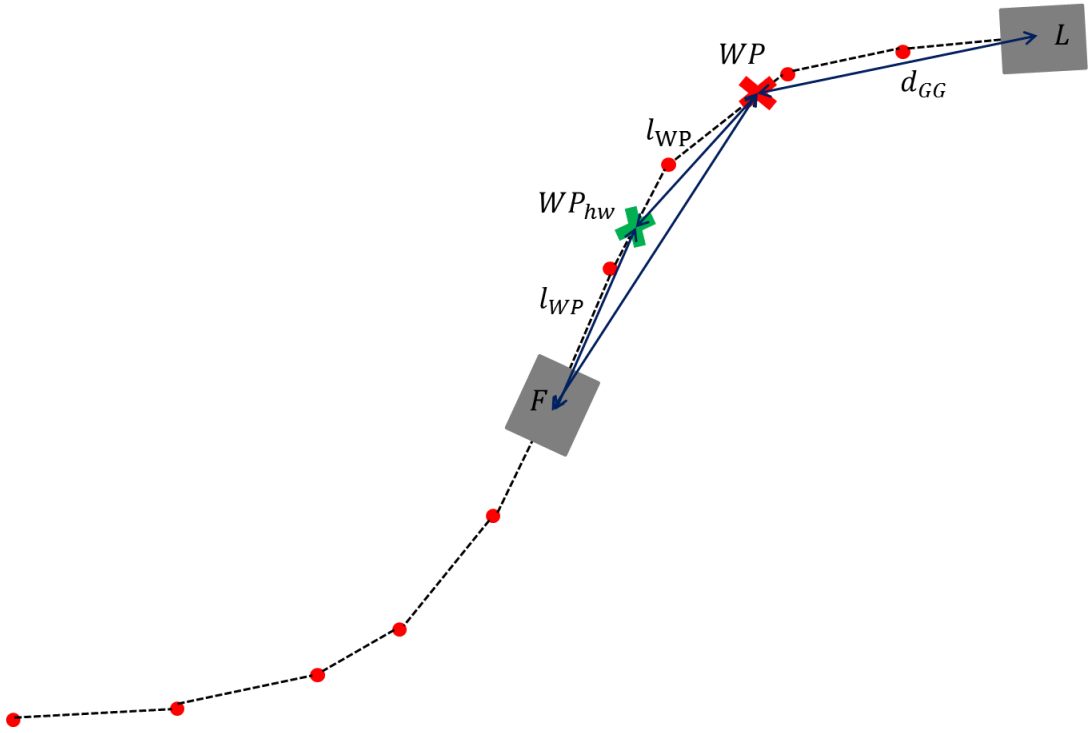

Figure 16. Notations used for Platooning approach 2. The red dots show the prior positions of the leader.

Leader

The leader is controlled by the controller proposed for Dynamical model 2 (4.8) – (4.9) and follows a trajectory given by the negated gradient of the NF (2.1). In the beginning of each iteration the leader calculates the negated gradient and decides the values of $\Delta\theta_{leader}$ and $v_{want,leader}$ that are to be passed on to the controller. To make sure that the controller inputs are within certain limits the following relations are used to decide $\Delta\theta_{leader}$ and $v_{want,leader}$

$$\Delta\theta_{leader} = \begin{cases} \arg(\nabla\varphi) - \theta_{leader}, & |\arg(\nabla\varphi) - \theta_{leader}| < \Delta\theta_{max,leader} \\ \Delta\theta_{max,leader}, & |\arg(\nabla\varphi) - \theta_{leader}| \geq \Delta\theta_{max,leader} \end{cases} \tag{4.14}$$

$$v_{want,leader} = v_{max,leader} \cdot \frac{1}{c_1(\Delta\theta_{leader})^2+1} \tag{4.15}$$

where c_1 is a positive constant.The leader has information about its own dynamics (4.2) and is thus able to calculate the assumed velocity of the center of gravity. Based on this velocity it can estimate its position $\boldsymbol{q}_{leader,next}$ in Δt seconds and pass on the information to the follower.

Follower

The follower measures the current position of the leader and also remembers the leader's prior positions. It assumes that the leader has moved in straight lines between the positions and uses this information to find the point on the leader's path that is placed a distance d_{GG} from the assumed next position. This is where the follower should be in Δt seconds and it is therefore used as a waypoint WP (momentary destination point). To find the WP the follower first decides between which two previous positions of the leader $\boldsymbol{q}_{leader,1}$, $\boldsymbol{q}_{leader,2}$ the WP should be. It then finds the

point on the line connecting these points that is placed a distance d_{GG} from the assumed next position of the leader by solving

$$\left\|\boldsymbol{q}_{leader,1}\cdot\xi+\boldsymbol{q}_{leader,2}\cdot(1-\xi)-\boldsymbol{q}_{leader,next}\right\|=d_{GG}^2,\qquad 0\leq\xi\leq 1 \tag{4.16}$$

and then set

$$WP=\boldsymbol{q}_{leader,1}\cdot\xi+\boldsymbol{q}_{leader,2}\cdot(1-\xi) \tag{4.17}$$

But if the follower goes straight towards the WP its trajectory might deviate quite a lot from the leader's if the leader is following a curved path. To make the follower follow the leader's path more closely it is assumed that the follower can change orientation twice as often as the leader. This enables the follower to use another point on the leader's path as a halfway waypoint WP_{hw} and move straight towards this one for $\frac{\Delta t}{2}$ seconds and thereafter change orientation and move towards the WP. WP_{hw} is supposed to be the point on the leaders path with equal distance to the WP and to the current position of the follower and is found using some linear algebra. The middle point on the line between the WP and the current position of the follower is identified and the equation of a line that passes through that point and is perpendicular to said line is found to be

$$\boldsymbol{p}=\begin{pmatrix}x_p\\y_p\end{pmatrix}+K\begin{pmatrix}y_{follower}-y_{WP}\\x_{WP}-x_{follower}\end{pmatrix} \tag{4.18}$$

where $\begin{pmatrix}x_p\\y_p\end{pmatrix}=0.5\begin{pmatrix}x_{follower}+x_{WP}\\y_{follower}+y_{WP}\end{pmatrix}$.

The equations of the lines connecting the points on the leaders track can be expressed as

$$\boldsymbol{r}_i=\begin{pmatrix}x_i\\y_i\end{pmatrix}\xi+\begin{pmatrix}x_{i+1}\\y_{i+1}\end{pmatrix}(1-\xi) \tag{4.19}$$

where the indices $i,\ i+1$ denotes the points placed between the current position of the follower and the WP. WP_{hw} is supposed to be the point where $\boldsymbol{p}$ crosses the leader's path and is found by solving

$$\begin{pmatrix}y_{follower}-y_{WP} & x_{i+1}-x_i\\x_{WP}-x_{follower} & y_{i+1}-y_i\end{pmatrix}\begin{pmatrix}K\\\xi\end{pmatrix}=\begin{pmatrix}x_{i+1}-x_p\\y_{i+1}-y_p\end{pmatrix} \tag{4.20}$$

for K,ξ using the different x_i,y_i and x_{i+1},y_{i+1}. The solution where $0\leq\xi\leq 1$ gives WP_{hw} when putting the value of ξ in the equation for the corresponding $\boldsymbol{r}_i$.

To decide the magnitude of the speed the follower calculates the distance from its current position to WP_{hw} and from WP_{hw} to WP, both l_{WP}. Based on that information it gives a velocity input to the controller proposed for Dynamical model 2 (4.8) – (4.9) so that it will travel approximately that distance in Δt seconds. The follower also finds out the wanted orientation of the center of gravity and decides a suitable input to the controller.

$$v_{want,follower}=\begin{cases}\frac{2l_{WP}}{\Delta t}, & \frac{2l_{WP}}{\Delta t}<v_{max,follower}\\ v_{max,follower}, & \frac{2l}{\Delta t}\geq v_{max,follower}\end{cases} \tag{4.21}$$

$$\Delta\theta_{follower} = \begin{cases} \arg(WP_{hw} - \boldsymbol{q}_{follower}) - \theta_{follower}, & |\arg(WP_{hw} - \boldsymbol{q}_{follower}) - \theta_{follower}| < \Delta\theta_{max,follower} \\ \Delta\theta_{max,follower}, & |\arg(WP_{hw} - \boldsymbol{q}_{follower}) - \theta_{follower}| \geq \Delta\theta_{max,follower} \end{cases} \quad (4.22)$$

The paths of a leader and a follower when simulating Platooning approach 2 are shown in Figure 17 and the distance between the vehicles is shown in Figure 18

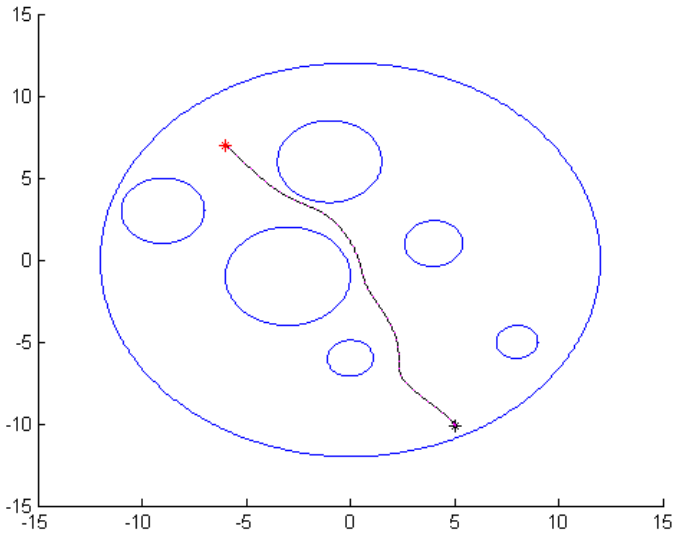

Figure 17. Path of a leader and a follower using Platooning approach 2 with $k = 15, c_1 = 5$. The black line shows the path of the leader and the pink line (on top of the black line) shown the path of the follower. The black star shows the starting position and the red star shows the destination point.

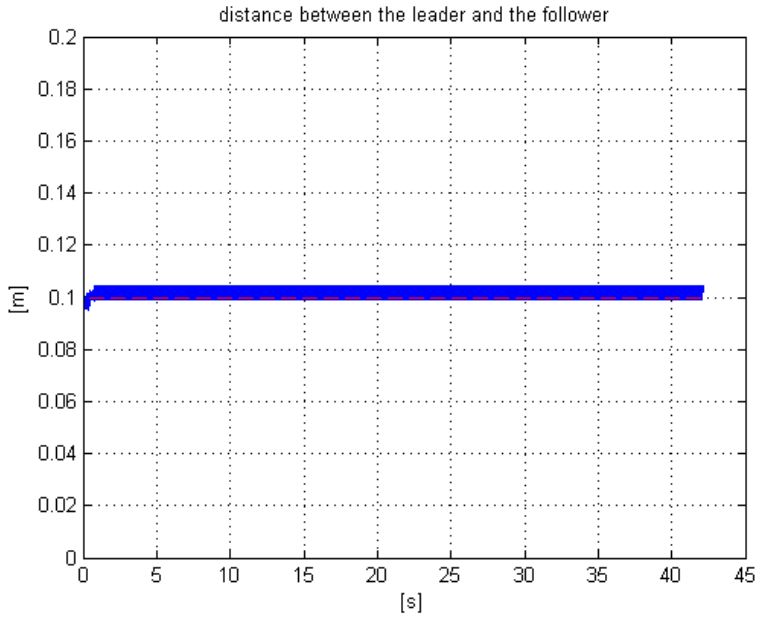

Figure 18. The distance between the leader and the follower using platooning approach 2. The red dotted line shows the reference distance and the blue line, oscillating around the red line, shows the distance as a function of the time.

4.3.2. Platooning Approach 3

Platooning approach 3 is based on Dynamical model 3 (4.4). The leader follows an arbitrary track, in this case based on the NF, and the follower is supposed to follow the same track keeping a constant distance to the leader. Since the vehicles are steered by controlling the acceleration and angular velocity they will not take a new speed and steering angle instantaneously, but there will be a time delay. This means that the vehicles have to plan their path some time in advance.

Leader

The leader uses the negated gradient of the NF to find a path that leads to the destination point. Based on its current position and velocity it can approximate its position in Δt seconds

$$\boldsymbol{q}_{leader,est}(t+\Delta t) = \boldsymbol{q}_{leader}(t) + \boldsymbol{v}_{leader}(t)\cdot\Delta t \tag{4.23}$$

and calculate the negated gradient of the NF (2.1) in that point. The negated gradient gives the wanted orientation in Δt seconds and that information together with a wanted velocity in Δt seconds enables the leader to give suitable inputs to the controller proposed for Dynamical model 3 (4.4)

$$\dot{\alpha}_{want.leader} = \frac{\arg(\nabla\varphi)-\alpha_{leader}}{\Delta t} \tag{4.24}$$

$$a_{g,want.leader} = \frac{v_{g,want}-v_{leader}}{\Delta t} \tag{4.25}$$

where

$$v_{g,want} = \frac{b_1}{b_1+\phi^2}\cdot\begin{cases} 1, & |f(\dot{\alpha})| \le \dot{\phi}_{max} \\ \frac{1}{1+b_2(|f(\dot{\alpha})|-\dot{\phi}_{max})}, & |f(\dot{\alpha})| > \dot{\phi}_{max} \end{cases} \tag{4.26}$$

b_1, b_2 are positive constants and $f(\dot{\alpha}) = \left(\dot{\alpha} - \frac{v_b}{l}\tan(\phi)\right)\frac{\cos^2(\phi)+\eta^2\sin^2(\phi)}{\eta}$.

The first part of $v_{g,want}$ is supposed to give the leader a lower velocity where the path is curved, and the second part is to decrease the velocity if the constrained steering prevents the leader from following the wanted direction.

The leader has information about its dynamical model (4.4) and is thus able to predict its velocity in Δt seconds and there through its position in $2\cdot\Delta t$ seconds, based on the controller inputs. This information is then passed on to the follower to help its path planning.

Follower

The follower gets information from the leader about its estimated position in $2\cdot\Delta t$ seconds and uses the method described in (4.16) – (4.17) to find the point on the leader's track that is placed a distance d_{GG} from the expected position of the leader. This point is to be used as a WP. The follower estimates its own position in Δt seconds according to

$$\boldsymbol{q}_{follower}(t+\Delta t) = \boldsymbol{q}_{follower}(t) + \boldsymbol{v}_{follower}(t)\cdot\Delta t \tag{4.27}$$

and then decides the controller inputs to (4.12), (4.13) from

$$\dot{\alpha}_{want,follower} = \frac{\arg\left(WP-\boldsymbol{q}_{follower}(t+\Delta t)\right)-\alpha_{follower}}{\Delta t} \tag{4.28}$$

$$a_{g,want,follower} = \frac{v_{g,want,follower} - v_{follower}}{\Delta t} \tag{4.29}$$

where

$$v_{g,want,follower} = \left(\frac{|WP - \boldsymbol{q}_{follower}(t+\Delta t)|}{\Delta t}\right) \cdot \begin{cases} 1, & |f(\dot{\alpha})| \leq \dot{\phi}_{max} \\ \frac{1}{1+b_3(|f(\dot{\alpha})| - \dot{\beta}_{max})}, & |f(\dot{\alpha})| > \phi_{max} \end{cases} \tag{4.30}$$

$b_3$is a positive constant and $f(\dot{\alpha}) = \left(\dot{\alpha} - \frac{v_b}{l}\tan(\phi)\right)\frac{\cos^2(\phi)+\eta^2\sin^2(\phi)}{\eta}$.

The first term in $v_{g,want,follower}$ is supposed to give the follower a suitable velocity to reach the WP and the second term is supposed to decrease the velocity if the constrained steering prevents the follower from following the wanted direction.

The path when simulating Platooning approach 3 is shown in Figure 19 and the distance between the vehicles is shown in Figure 20.

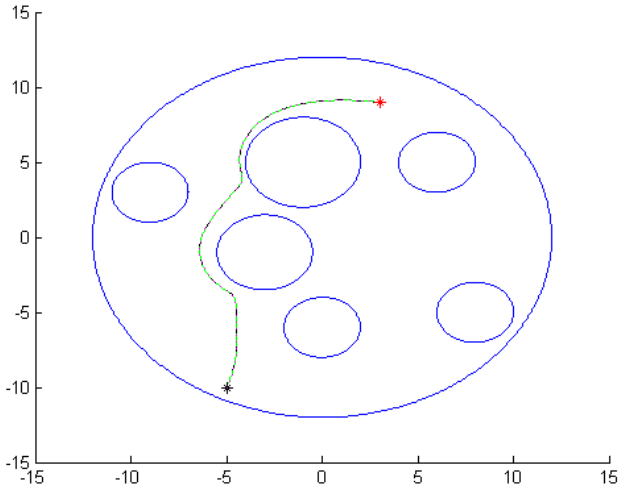

Figure 19. Path of a leader and a two followers using Platooning approach 3 with $k = 15, b_1 = 1, b_2 = 0.01, b_3 = 0.0025$. The black line shows the path of the leader, the green line (on top of the black line) shows the path of the first follower and the pink line (on top of the green line) shows the path of the second follower. The black star shows the starting position and the red star shows the destination point.

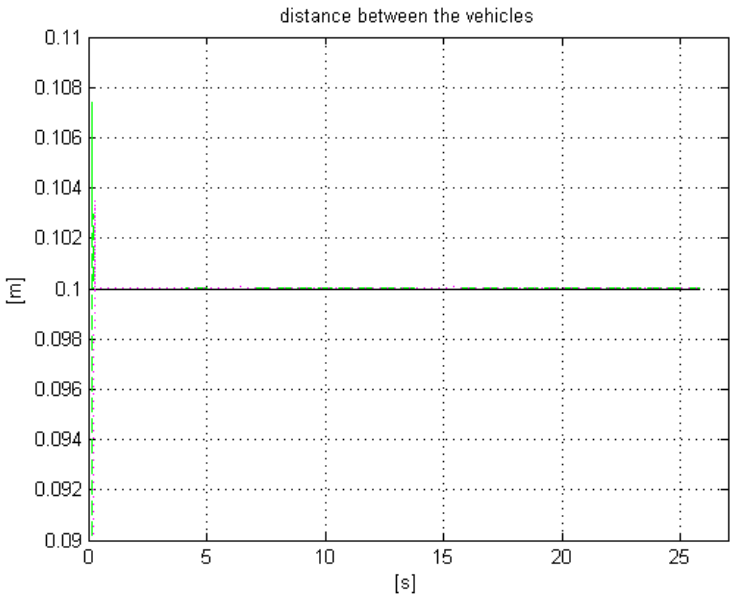

Figure 20. The distances between a leader and two followers using platooning approach 3. The black line shows the reference distance, the green line shows the distance between the leader and the first follower and the pink line shows the distance between the first follower and the second follower.

5. Other Ideas

During the work a couple of ideas came up which are based on the NF but also include inspiration from other sources. The first one enables navigation of a non-point agent in a sphere world and the second one smoothens out harsh turns on paths given by the negated gradient of the NF.

5.1 Follow Boundary

The NF defined in [1] is built on the assumption that the agent is a point agent. This might cause some troubles when using the NF for the navigation of real agents, which cannot be considered to be point agents since they take up a certain volume. The problem appears if the NF proposes a trajectory very close to an obstacle boundary or in between two obstacles with just a narrow space in between. If the agent is a point agent it will be able follow the trajectory, but if it has a certain volume it will probably hit the obstacle boundary or get stuck between the obstacles.

Apparently another method has to be used for the navigation of a non-zero sized agent. [5] gave the idea of letting the agent move straight towards the destination point as long as there were no obstacles blocking the way and, if the agent got close to an obstacle, let it follow the boundary of it until it could once again go towards the destination point. This idea led to an algorithm were the agent uses the negated gradient of the $\gamma_d$part of the NF (1.2) to go straight towards the destination point and vectors tangential to the obstacle boundaries when following them around to avoid a collision. By adjusting the value of the parameter telling how close to an obstacle boundary the agent is allowed to go before it has to follow the boundary around, it is possible to make the algorithm work for different sizes of the agent.

Controller design

The agent is assumed to be a sphere with radius r_{agent} and the position of the agent is represented by its center point $\boldsymbol{q}$. A distance r_{avoid} is set to be the closest to an obstacle boundary that the center point is allowed to go and if $r_{avoid} > r_{agent}$ it means that no point of the agent will intersect with an obstacle. The obstacles are also supposed to be circular, as in [1], and can thus be represented by a center point $\boldsymbol{q}_j$ and a radius ρ_j. In the beginning of each iteration the agent calculates the distance to each of the M center points of the obstacles $\|\boldsymbol{q} - \boldsymbol{q}_j\|$ and the distance to the boundary of the world $\rho_0 - \|\boldsymbol{q}\|$. If

$$\|\boldsymbol{q} - \boldsymbol{q}_j\| > \rho_j + r_{avoid},\ j = 1, \dots, M \qquad (5.1\text{ a})$$

$$\rho_0 - \|\boldsymbol{q}\| > r_{avoid} \qquad (5.1\text{ b})$$

is fulfilled the agent is sufficiently far away from the obstacle boundaries and can go towards the destination point. The controller used is then

$$\dot{\boldsymbol{q}} = e_1 \cdot (\boldsymbol{q}_d - \boldsymbol{q}) \qquad (5.2)$$

where e_1 is an arbitrary positive constant deciding the magnitude of the speed.

If the inequality (5.1) is not fulfilled the agent is close to an obstacle boundary and should thus follow the boundary to avoid a collision. The number of the obstacle whose boundary the agent is closest to is denoted by n, and the following controller is used

$$\dot{\boldsymbol{q}} = e_2 \begin{pmatrix} y_n - y \\ x - x_n \end{pmatrix} + \frac{(r_{avoid} - (|\boldsymbol{q} - \boldsymbol{q}_n| - \rho_n))}{|\boldsymbol{q} - \boldsymbol{q}_n|} \begin{pmatrix} x - x_n \\ y - y_n \end{pmatrix} \qquad (5.3)$$

where e_2 is a constant. The sign of e_2 decides whether the agent should turn left or right and the second term is to make sure that the agent keeps the distance r_{avoid} to the obstacle boundary and neither gets too close nor too far away from it.

Figure 21 shows the track of a single integrator type agent controlled by (5.2) – (5.3)

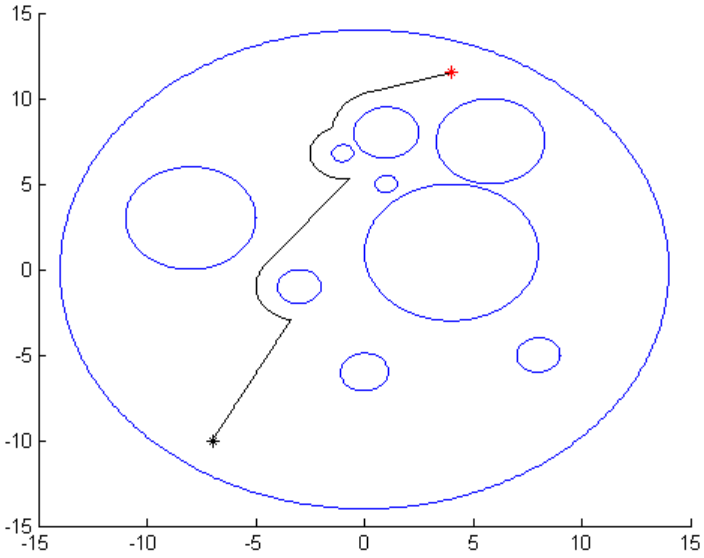

Figure 21. The track of an agent with single integrator dynamics following the boundaries of the obstacles. The black star marks the starting position and the red star marks the destination point.

5.2 Avoiding Harsh Turns

When considering single-integrator dynamics it is always possible to follow the direction of the negated gradient of the NF. When considering for example unicycle dynamics however, constraints on the angular velocity might prevent the agent from changing direction fast enough to be able to follow the path given by the negated gradient. If the path takes a sudden direction change near an obstacle, the agent's limited pivot ability might make it collide with the obstacle, unless it is able to attain a very low speed instantly.

It is therefore desirable to smoothen out the path of the negated gradient near obstacles. A possible way is to add a vector tangent to an imaginary circle, with center point at the center of the closest obstacle and radius equal to the distance between said point and the position of the agent, whenever the agent is within a certain distance from an obstacle. The new desired orientation of the agent will thus be the orientation of the vector sum of the negated gradient and the tangent vector. This means that the agent will start deviating from obstacles before it is too close and hence avoid sharp turns generated by the negated gradient close to obstacle boundaries.

Controller design

The tangent vector should be added whenever

$$\|\boldsymbol{q}_i - \boldsymbol{q}\| - \rho_i \leq r_{add} \tag{5.4}$$

for any obstacle i. The vector $\boldsymbol{t}$ is found from

$$\boldsymbol{t} = k_1 \cdot R \cdot \frac{(\boldsymbol{q}-\boldsymbol{q}_j)}{\|\boldsymbol{q}-\boldsymbol{q}_j\|} \tag{5.5}$$

where $\boldsymbol{q}_j$ denotes the center of the obstacle whose boundary the agent is closest to at the moment and $\boldsymbol{q}$ is the position of the agent. R denotes a rotation matrix rotating $\pm\frac{\pi}{2}$ radians depending on the orientation of the agent and k_1 is a constant that can be tuned to adjust the norm of $\boldsymbol{t}$. Adding $\boldsymbol{t}$ to the negated gradient gives the vector

$$\boldsymbol{s}_{new} = \boldsymbol{\nabla}\varphi + \boldsymbol{t} \tag{5.6}$$

pointing in the new desired direction, see Figure 22. For an agent with single integrator dynamics the following controller can then be used

$$\dot{\boldsymbol{q}} = \begin{cases} \boldsymbol{\nabla}\varphi, & \|\boldsymbol{q}_i - \boldsymbol{q}\| - \rho_i > r_{add} \\ \boldsymbol{s}_{new}, & \|\boldsymbol{q}_i - \boldsymbol{q}\| - \rho_i \leq r_{add} \end{cases} \tag{5.7}$$

The path of an agent with single integrator dynamics that is always following the negated gradient is shown in Figure 23 and the path of the same agent when applying the path smoothing algorithm is shown in Figure 24.

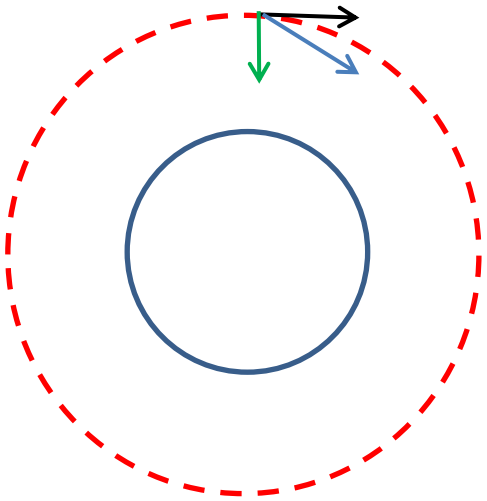

Figure 22. The green arrow shows the negated gradient and the black arrow shows the vector tangent to the red circle. The blue arrow shows the resulting vector, pointing in the new desired direction.

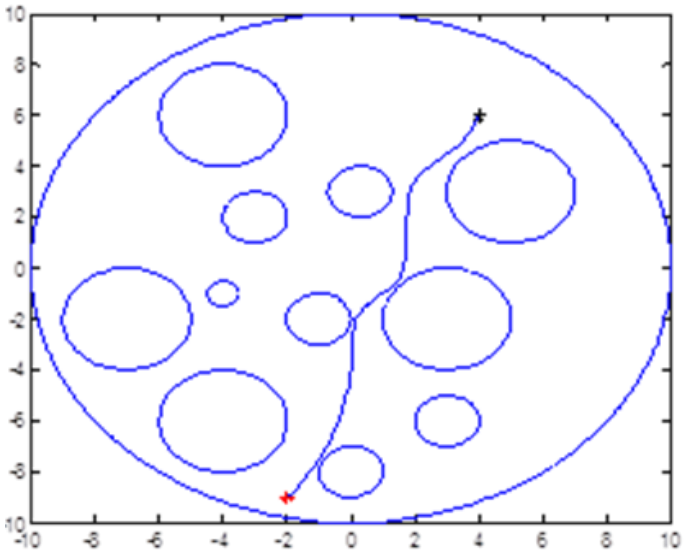

Figure 23. The path of an agent with single integrator dynamics following the negated gradient.

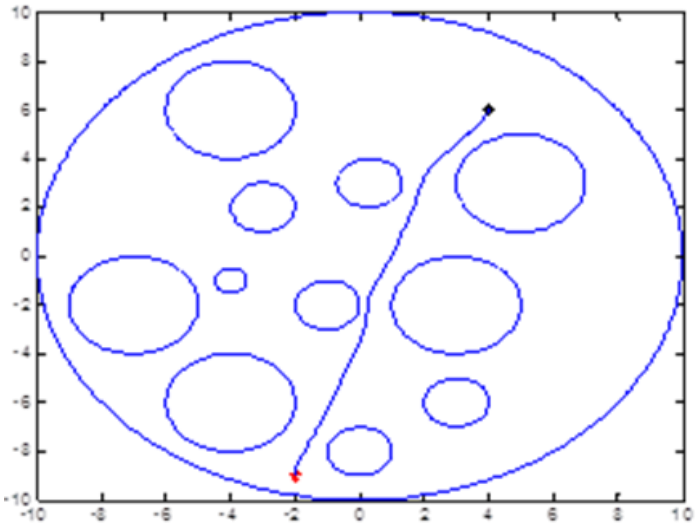

Figure 24. The path of an agent with single integrator dynamics following the path smoothing algorithm.

6. Software package

6.1. Available scripts

Throughout the work several Matlab scripts were written to simulate the behavior of agents controlled by the controllers proposed in the report. Some of them are easy to handle and are therefore included in a software package called “Mixed Navigation” to be used by anyone who wishes to make their own simulations. The available scripts are the ones used for;

- Single integrator type agent in a known- as well as an unknown environment, controller (2.3).
- Platooning approach 2 & 3 with one or two followers, controllers (4.14) – (4.15) and (4.24) – (4.25).
- Single integrator type agent following the boundaries of obstacles, controllers (5.2) – (5.3).

6.2. Structure

The structure of the different scripts is described below. More details can be found in the scripts’ commentary lines.

6.2.1. Single integrator type agent in known – as well as unknown environment

In a main script called *main_integrator.m* it is possible to specify the environment, i.e. position and size of obstacles, and decide if it should be considered to be known or unknown. *main_integrator.m* calls *integrator_known_unknown.m* that in turn calls *obs_plot.m* and also performs the simulations out from the specification in the main script, see Figure 25.

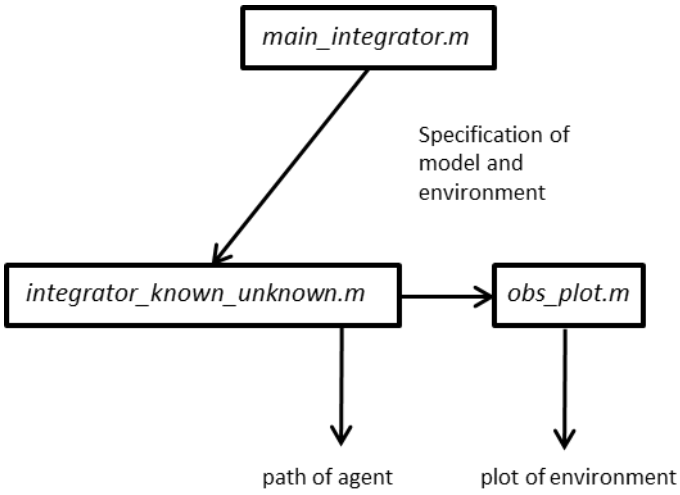

Figure 25. The structure of the single integrator scripts.

6.2.2. Platooning approach 2

Information about the world in which the vehicles are to navigate, i.e. center points and radii of the obstacles, is given to the main script *platoon_line_several.m*. The script is used to plot the environment and to call the function files *leader_line.m, follower_line.m* and *follower2_line.m* in which the navigation of the vehicles is performed, see Figure 26.

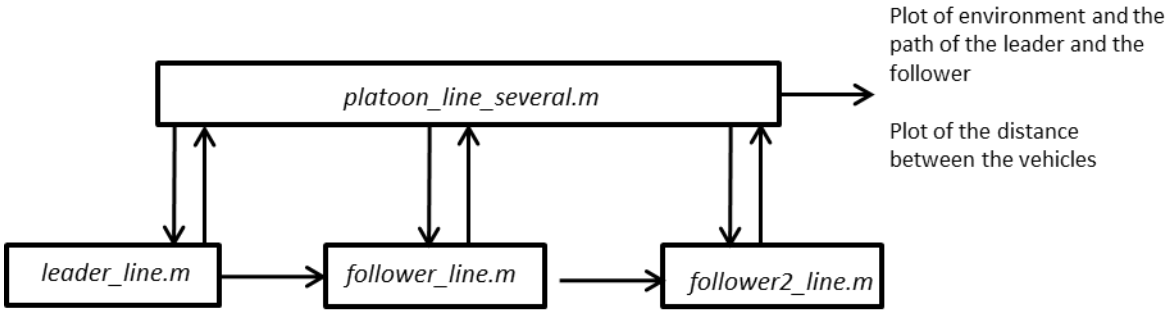

Figure 26. The structure of the scripts used for Platooning approach 2.

6.2.3. Platooning approach 3

As in the scripts used for Platooning approach 2, information about the world in which the vehicles are to navigate is given to the main script *platoon_acc_2.m*. The script is used to plot the environment and to call the function files *leader_acc.m, follower_acc.m, follower2_acc.m* in which the navigation of the vehicles is performed, see Figure 27.

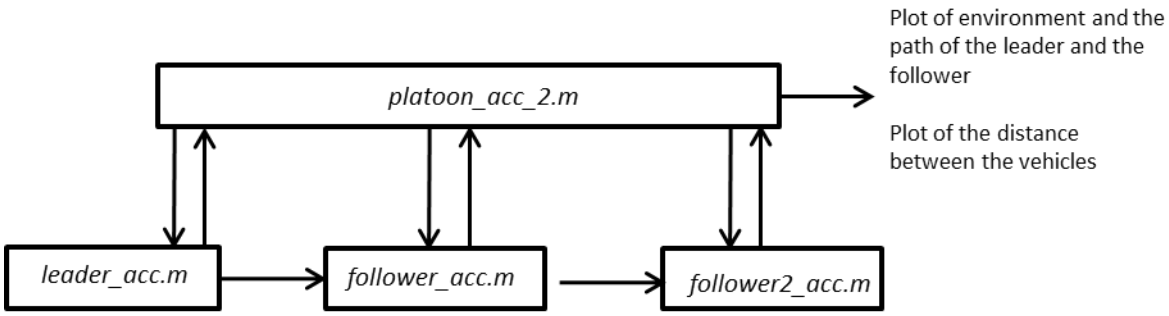

Figure 27. The structure of the scripts used for Platooning approach 3.

6.2.4. Single integrator type agent following the boundaries of obstacles

For the single integrator type agent following the boundaries of obstacles the world is specified in the main script *integrator_unknown_env.m*. It calls *obs_plot.m* in which the environment is plotted and then performs the navigation of the agent, see Figure 28.

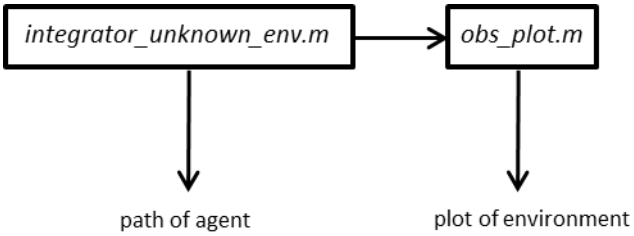

Figure 28. The structure of the scripts used for the integrator type agent following the boundaries of obstacles.

6.3. Useful tips

If making changes to the scripts there are some things to keep in mind

- Be careful if changing the values of the tuning parameters since a poorly chosen value might make the agent lose its track.

- It is possible to include more followers in the platooning algorithms. Simply copy the script for the follower and change the name of the input and output variables by increasing their number, i.e. v_f becomes v_f2, v_f2 becomes v_f3 and so on.
- There is a variable specifying the maximum number of iterations. This variable makes sure that the iterations stops if something goes wrong and can be given any positive value.
- Before making any changes to the scripts, it is a good idea to save a copy of the original version.
- If something goes wrong and the iterations do not stop; *ctrl+c* can be typed in the command window and will stop the iterations immediately.

7. Discussion

7.1. Sphere worlds

Most of the presented motion planning strategies are based on assumptions of a sphere world and a point agent. This is a natural consequence of using the NF theory, since it is built on the same assumptions, but nevertheless it complicates the use of them in a real world. One reason is that most obstacles in a real world are not spherical but have an arbitrary shape. A solution to that problem could be to define even the arbitrarily shaped obstacles as spherical ones by placing center points inside the obstacles and then define all points within certain radii from the center points as parts of the obstacles. This would solve the problem with an arbitrary shape, but as a consequence points which were originally part of the free space would now be considered to be obstacles and the agent would thus be prevented from reaching them.

If the world is unknown another problem might arise. The agent will then have a spherical sensing zone and will thus be able to detect the boundaries of the obstacles. However, most of the motion planning strategies are based on information about the center points of the obstacles and the agent must then be able to find the center points based on information about the boundaries. If the obstacles are spherical and the sensors give accurate information about the distance to some points on the boundaries, the center points can probably be found based on the curvature of the boundaries. However, if the information from the sensors is not accurate enough or if the obstacles are not spherical but arbitrarily shaped, it will be difficult to find the curvatures and consequently also the center points.

7.2. Point agents and non-point agents

Most of the motion planning has been based on the assumption of a point agent, mainly because the NF used was based on the same assumption. Since most of the controllers were designed to respect dynamical constraints of real agents, it would have been preferable to plan paths that made sure that all parts of a non-zero sized agent stayed away from obstacles, not just one point of it. However, no NF was found that enabled such path planning in an environment with static obstacles. Based on [2] an attempt was made to simply increase the radii of the obstacles with the radius of a spherical agent, but the boundaries of obstacles placed close to each other then came to intersect and so the condition of non-intersecting obstacles from [1] was violated causing the agent to sometimes get stuck in front of the obstacles.

7.3. Dynamical models and platooning

The platooning algorithms were based on dynamical models of trucks and since the models were simplifications of real systems it is difficult to tell how well the platooning algorithms would work if applying them to real trucks. The accuracy of the simulation results is hence dependent on the exactness of the models used. Another aspect that has to be considered in a real life application is what information the agents actually can obtain and how long the computations will take. The platooning algorithms for the followers suggested in this report is mostly based on information about the position of the vehicle in front, which should be possible to obtain. However, the time needed to get the information is unknown and it is thus possible that there could be a problem with time delays when applying the algorithms in real time.

As for the path planning, the main concern for the platooning was to make the leader and the followers follow the same track, irrespective of what it looked like. Therefore it did not really matter

that the NF provided a path meant for a point agent, since the only difference between following that path or any other path would be the inputs to the controllers.

7.4. Further work

For further work it would be interesting to see if the NF can be adapted to work for non-zero sized agents in a world of static obstacles. [2] suggests an algorithm for collision avoidance for non-zero sized agents but it only concerns multiple agents in a sphere world, not static obstacles. It is possible that an algorithm for static obstacles already exists, but it has not been found during the research for this project. It would also be interesting to see if the NF could be adapted to work in a non-sphere world.

As for the motion planning of vehicles, the possibility to design a NF with a negated gradient smooth enough to respect the dynamical constraints of vehicles could be investigated. It would also be interesting to see if any of the controllers suggested in this report can be proved to guarantee convergence to a destination point, when using inputs based on the direction of the negated gradient.

References

Papers

[1] D. E. Koditschek, E. Rimon. "Robot Navigation Functions on Manifolds with Boundary". *Center for Systems Science, Department of Electrical Engineering*, Yale University, 1990.

[2] D.V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos, M. M. Zavlanos." A Feedback Stabilization and Collisions Avoidance Scheme for Multiple Independent Non-point Agents*"*. *Control Systems Laboratory, National Technical University of Athens*, 2005.

[3] I. Filippidis. "Navigation Functions for Unknown Sphere Worlds, General Geometries, their Inverse Problem and Combination with Formal Methods*"*. *Control Systems Laboratory, Mechanical Engineering Department, National Technical University of Athens*, 2011.

[4] G. Lionis, X. Papageorgiou, K. Kyriakopoulos et al. "Locally Computable Navigation Functions for Sphere Worlds*"*. *2007 IEEE International Conference on Robotics and Automation*, April 2007.

[5] A. Kokosy, F-O Defaux, W. Perruquetti et al. "Autonomous navigation of a nonholonomic mobile robot in a complex environment*"*. *Proceeding of the 2008 IEEE International Workshop on Safety, Security and Safety Robotics*, October 2008.

Litterature

[6] S. M. LaValle. "Planning Algorithms", *Cambridge University Press*, 2006.

Websites

[7] Scania AB. *"*http://newsroom.scania.com/en-group/2012/04/04/scania-lines-up-for-platooning-trials/*"*. Södertälje, Sweden, August 2012.

Computer software

[8] The MathWorks, Inc*. Matlab R2011a*, 2011