Time-varying model predictive control for formations of nonholonomic multi-agent systems

Schreurs, E.

Award date:
2012

Time-varying Model Predictive Control for Formations of Nonholonomic Multi-agent Systems

E. Schreurs

DC 2011.062

Master's thesis

Committee: prof. dr. H. Nijmeijer
dr. ir. D. Kostić
dr. M. Lazar
dr. ir. A. Alvarez Aguirre

Eindhoven University of Technology
Department of Mechanical Engineering
Dynamics & Control

Eindhoven, December, 2011

Contents

Chapter 1

Introduction

This chapter starts by giving an introduction to the field of multi-agent systems. It also explains the motivation behind the project that is described in this thesis and gives an overview of the components that are combined in this thesis. The chapter ends with a problem description, and an outline of the thesis.

1.1 Multi-agent systems

Systems that consist of multiple agents have been receiving increasing attention over the last decades. This is due to their wide range of applications, and the fact that multiple agents can cooperate to achieve more complex tasks [6]. Some examples of these tasks are mapping and exploration of unknown terrain and cooperative transportation of objects. Overviews of the fields in which such research is conducted, along with numerous references are given in [3, 46]. A few examples of uses of multi-agent systems are shown in Figure 1.1.

(a) Multiple quadrotors grasping an object [41].

(b) Highway driving with communication [25].

(c) Two rovers carrying a load cooperatively [52].

(d) A team of RoboCup soccer robots [57].

Figure 1.1: Examples of different applications of multi-agent systems.

In Figure 1.1(a), a situation is shown where multiple quadrotor robots are used to cooperatively transport a large object. Figure 1.1(b) shows a number of cars that are driving on a highway and are commu-

nicating with each other to ensure smooth traffic flow. Another example of cooperative transportation is shown in Figure 1.1(c), where two rovers are used to carry a large beam. They can also be used as all terrain explorers. Finally, Figure 1.1(d) shows a part of a team of the soccer robots at the Eindhoven University of Technology. These robots participate in the RoboCup competition, which strives towards the objective of making robots that are able to beat a professional human team in soccer around 2050.

1.2 Scope of the project

A different area where a multi-agent system of mobile robots can be used, is to transport goods in warehouses. There are research projects, such as the Falcon project [15], that focus on the development of a new generation of warehouses. The goal of the Falcon project is to design a system with a maximum degree of automation. In this project, particular attention is payed to

- New warehouse architectures that are suited for the automated handling of goods,
- Optimization of the set of parallel processes that determine the flow of goods,
- The design of automated solutions for handling and transportation of goods.

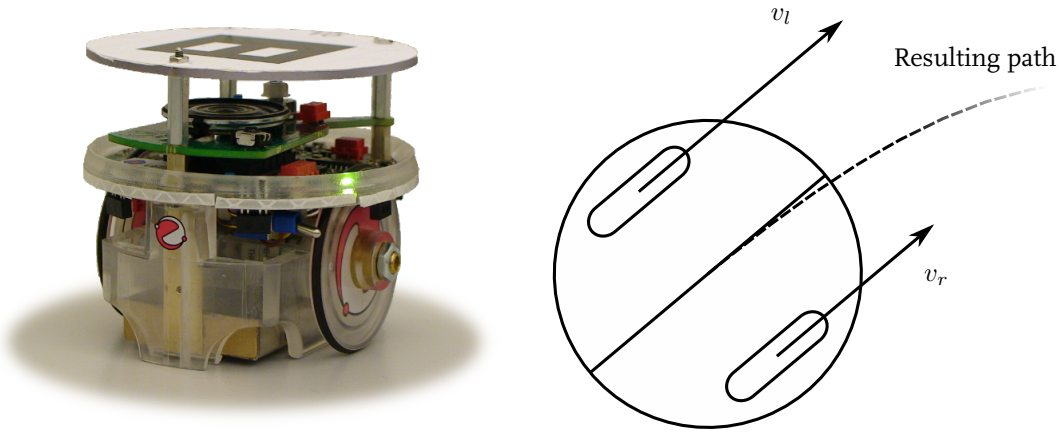
This thesis focusses on a part of this last research topic, namely the automated handling of goods. In a traditional warehouse, conveyor belts are used for the transportation of goods, but in a new generation distribution center, autonomous vehicles are used.

The main advantages of using autonomous vehicles instead of conveyer belts are increased scalability of the system capacity, and increased system reliability [58]. Conveyor belts have a maximum capacity, which can not easily be increased. If an increase of capacity is required, the layout of the whole transport system has to be altered. The capacity of a transport system that uses autonomous vehicles can be increased by simply adding new vehicles to the system.

The use of autonomous vehicles can also increase reliability of the system. When a conveyor belt breaks down, it causes a block which can influence the whole transportation system. The breakdown of an autonomous vehicle will only lead to a small decrease in capacity without disturbing the rest of the system.

1.3 Unicycle robots

The autonomous vehicles that can be used for transportation of goods within a new generation warehouse are nonholonomic mobile unicycle robots. From here on, the term unicycle will be used to describe these systems. An example of a unicycle is shown in Figure 1.2(a). This is the e-puck mobile robot, which is used in this thesis to perform experiments to validate simulation results. A schematic representation of the e-puck, or any other unicycle, is shown in Figure 1.2(b).



(a) The e-puck mobile robot.

(b) Schematic representation of the e-puck.

Figure 1.2: The e-puck unicycle and a schematic representation of a unicycle.

This robot has two motors that drive the two wheels. By controlling the left and right motors, and therefore controlling velocities v_l [m/s] and v_r [m/s], the robot can drive forward and backward when the wheel speeds are equal. When the wheel speeds are not equal, the unicycle will turn.

This system falls into a specific class of so-called nonholonomic robots, because of the nonholonomic constraints that are imposed on its the motion. These nonholonomic constraints establish a relationship between the system coordinates and their time derivatives, which is not integrable [32]. Because of these constraints, the unicycle can not move sideways without slipping. This means that the unicycle can not follow every desired path. However, if enough space if available, the unicycle will be able to reach every desired position. The unicycle controller should stabilize and control the unicycle, under the influence of the nonholonomic constraints.

1.4 Model predictive control

A control method that is able to deal with nonholonomic constraints is model predictive control, from here on referred to as MPC. This control method determines the control inputs of the system to be controlled by means of optimization. The controller uses a model of the system to predict what the effect of a future input sequence will be. The effect of this future input sequence is then compared to the control objectives in a so-called cost function. This results in a cost function value that indicates the performance of the system. An optimization algorithm is then used to find the most optimal future input sequence. The first part of this future input sequence is implemented on the system, and the optimization process is repeated at the next instant.

1.5 Problem description

In this thesis, the goal is to design a model predictive controller that is able to control multiple unicycles that operate in the same workspace. The unicycles need to be able to follow reference trajectories or drive to a desired location. While doing so, they need to avoid collisions with other unicycles, as they operate in a shared workspace. It is desired that collisions with obstacles that are present in the environment are also avoided. The unicycles need to be able to drive in a formation with multiple unicycles as well. The controller needs to fulfill these control objectives while dealing with saturation constraints on control signals. It is also desired that the controller can cope with time-varying control objectives, to end up with a flexible controller.

1.6 Outline of this thesis

This thesis is constructed in the following manner. Chapter 2 gives an overview of control methods that have already been designed to control one or multiple unicycles. Chapter 3 gives a description of traditional MPC, where the time it takes to compute the control inputs is much shorter than the sampling time. It then discusses what happens when MPC is used in a system where these time scales are closer to each other. After this, Chapter 4 introduces the unicycle model that is used in this thesis. It also shows how the continuous-time model can exactly be discretized and how this model can be used to predict the future system behavior. In Chapter 5, the cost function is discussed. This determines a scalar performance value of a given future input sequence. To obtain the optimal future input sequence, the cost function is used in an optimization problem, which is discussed in Chapter 6. Then, in Chapter 7 simulation results of the controller are shown. To validate these results, experiments are performed. The experimental setup, experimental results, and a comparison with simulation results are presented in Chapter 8. Finally, conclusions and recommendations about the results of this thesis are given in Chapter 9.

Chapter 2

Literature Review

Numerous different control methods have already been proposed to control one or multiple unicycles. This chapter discusses some of these control methods. First, control methods, other than MPC, of one and multiple unicycles are discussed. After that, MPC for one and multiple unicycles is discussed. Finally, some conclusions are drawn based on the results of the literature review.

2.1 Control of a single unicycle

A Lyapunov function is proposed in [29], from which a stabilizing control law is derived. This controller has adjustable gains that can be used to adjust convergence. Simulations are performed with and without limited control signals. Backstepping is used in [19], to obtain a stabilizing controller that is also suitable for tracking a reference signal. After the design, Lyapunov theory is used to show asymptotic stability. A similar approach is used in [34], but in this work, obstacle avoidance is included as well. This is done by using a so-called deformable virtual zone, which is a safe area surrounding the robot. When an object intrudes this zone, the robot tries to maintain the original shape of the zone, implicitly repelling obstacles.

It is also possible to use optimal control methods to control a unicycle [31]. This is discussed in Chapter 3 of [47], where formation control and dynamic obstacle avoidance is implemented, using optimal control. Another work that discusses optimal control of a unicycle is [49]. Here, optimal control is used to make a unicycle follow a shadower. The goal of the unicycle is to make it appear to the shadower as if it is not moving, only approaching slowly. Optimal control is combined with differential geometry to control a unicycle [56]. A large number of possible trajectory pieces are available, and optimal control is used to combine trajectory pieces to obtain a nonholonomic path. This work is a continuation of [50]. In Chapter 3 of [35], optimal control and other methods are discussed to determine an optimal path for a nonholonomic system. This path is generated without the presence of obstacles.

In [17], a stabilizing controller is derived. This controller consists of two parts: an aiming algorithm and a final stabilizer. The aiming algorithm drives the unicycles close to a desired location, after which the final stabilizer is switched on. This controller is tested in a simulation as well as in an experiment. In [44], a sliding-mode controller is designed for the path-tracking problem. It is assumed that there is a delay due to communication over a network. This delay is taken into account when deriving an exact discrete-time model. The controller is obtained from this model and tested in simulation. Another way of controlling a nonholonomic system is by using differential flatness [20]. Here, several case studies are presented that show the potential of this control method. In [43], a unicycle is converted to a class of systems that can be controlled by using sinusoids. The controller is able to connect two arbitrary orientations of a nonholonomic system and connect them via sinusoidal paths.

These are just a few control methods that can be used for the control of a unicycle. In Chapter 3 of [47], the control of nonholonomic systems is treated. Here, additional references to nonholonomic control methods are given. A comprehensive work that discusses nonholonomic systems is [4]. In this book, the basics of nonholonomic systems is discussed, along with methods to control them.

2.2 Control of multiple unicycles

To control more than one mobile robot, a range of control methods can be used. A virtual structure controller for a formation of unicycles is presented in [5]. In this controller, unicycles are mutually coupled, so that the formation is more robust against disturbances. It is shown that this controller is locally exponentially stable. To control the formation of a team of unicycles, virtual structures and artificial potential fields are used in [36]. Here, each unicycle needs to track its own virtual leader. Collisions with other unicycles are avoided by restricting each unicycle to operating in its own safety sector.

It is also possible to use synchronization to control a formation of robots that are modeled as point masses [55]. In this work, the robots can track their own references, but to keep a formation, they are linked via synchronization. Another work that uses synchronization is shown in [24]. A combination of the virtual structure and the path following method is used to determine the shape of the formation. The derivative of the unicycle's path is left free to allow robots to synchronize. A virtual structure controller for unicycle formations that also uses the same synchronization principle is given in [13].

In [11], a leader-follower formation strategy is used to control a group of unicycles. The followers are asymptotically stabilized using feedback linearization. In [40], a controller for multiple unicycles is derived using Lyapunov theory which guarantees collision avoidance and tracking of a reference trajectory. Leader-follower formations can also be used with this controller.

A control framework to control large groups of mobile robots is discussed in [61]. Here, decentralized control and consensus protocols are used to guide a large number of holonomic agents into a desired formation. In [12], a decentralized controller that utilizes navigation functions is derived. This controller is able to drive non-point agents to a desired location, while avoiding collisions that might occur. Optimal reciprocal collision avoidance is discussed in [1]. This work focusses on the collision avoidance when multiple unicycles are involved. Here, unicycles can even avoid collisions with each other in very crowded situations. The same collision avoidance strategy is used in [54]. It is extended so that obstacles, represented here as a defective unicycle, can be avoided as well. A controller that uses theory found in fluid mechanics, the so-called panel method, is utilized in conjunction with harmonic potential functions is presented in [18]. This controller can drive unicycles to a goal, while avoiding collisions with other unicycles and static obstacles.

Overviews of multi-robot controllers and their issues and strategies that are used nowadays are given in [3, 9, 46].

2.3 Control of a single unicycle using MPC

While it is possible to control a unicycle with the control methods mentioned in Section 2.1, there are certain advantages for using MPC. The aforementioned controllers can stabilize the unicycle and make it follow the desired reference trajectory. However, a disadvantage of these control methods is that most of them do not take the constraints into account that are present in the system. MPC is one of the few methods that is able to take these constraints into account. An example where MPC is used to make a unicycle track a given reference trajectory is given in [27]. In this work, and most others that are discussed in this section, the inputs of the unicycle are constrained. Another example of a nonlinear MPC path tracking controller is given in [60], where a car-like nonholonomic rear wheel drive car model tracks a reference signal, while also avoiding obstacles. The controller has a number of terms in its cost function to prevent input saturation, increase tracking performance, avoid obstacles, and reduce final error. In this case, trajectory generation is done beforehand, and MPC is used to track this reference signal.

It is also possible to use MPC to generate the path that the unicycle should follow. The advantage of doing this is that the generated path is feasible for the unicycle. Examples of controllers that both plan a path and drive a unicycle to a desired location are given in [26, 62, 64]. The so-called back-into-garage problem is solved in [39] for a nonholonomic car-like vehicle. MPC is used there, which replaces the need for a switching controller, as MPC automatically switches direction. This is made possible by well chosen weights in the cost function.

Another advantage of MPC is the simplicity of the formulation of the control problem. This is because all control objectives, such as reaching a desired state, obstacle avoidance, formation control, and other objectives are all grouped in a single cost function. For instance, in [38], collision avoidance is ensured by adding penalties on the state of the unicycle to the cost function.

The use of MPC can have some drawbacks however. The most important one becomes clear in [16], where nonlinear MPC is applied to a unicycle. The goal of this controller is to stabilize a unicycle at a desired location. The problem with this controller is that simulations take approximately 500 times longer than real-time on a Pentium III processor with 500 MHz, which was launched in 1999. This is due to the complexity of the optimization problem. The same problem is also encountered in [33], where both nonlinear and linear MPC is used. The nonlinear controller also causes a large computational load. Therefore, a linear controller is proposed which uses successive linearization, resulting in a much lower computational load. These last two works indicate that MPC can be a computationally intensive control method.

Another problem with MPC is that it is in most cases hard to prove stability of the control scheme. In [28], a receding horizon controller is used to park a unicycle at a desired location. This is done using a terminal state penalty to guarantee stability and increase convergence speed. Most works discussed here use a terminal state penalty in the cost function [26, 27, 63, 64]. There are some other methods that can be used, such as exponentially increasing the weight of the states [16], or adding a constraint that forces a norm on the first predicted state to decrease [62].

2.4 Control of multiple unicycles using MPC

With MPC, it is possible to combine the control objectives of a single unicycle in a single cost function. By then minimizing this cost function, an optimal or suboptimal future input vector is obtained. It is of course also possible to combine the control objectives of multiple mobile robots into one or more cost functions and to apply the same steps as in the case of a single unicycle.

An example of the use of MPC to control a formation of unicycles is given in [51]. Here, MPC is used to plan a trajectory that a leader should follow, and also use it to stabilize the followers. For the formation control, a leader-follower approach is used. First, a trajectory for the leader is generated. This avoids obstacles and plans the shortest route possible given a parameter that determines the precision of the planning optimization. The trajectories of the followers are generated from a delayed and shifted version of the leaders trajectory. When the formation is moving, every robot stays in formation and is also able to avoid static and dynamic obstacles that are detected on the way. The formation is able to move to a desired position in the optimal way, and it is also able to turn around in a confined area by switching between virtual leaders. At the end of the work, an experiment is discussed where multiple unicycles drive in formation to remove snow from a runway. A similar approach is taken in [63], where a dual-mode receding horizon controller is designed for the leader-follower formation control of multiple unicycles. This dual mode controller uses receding horizon control with a terminal state constraint to get the unicycles within a target area. After that, a different controller is switched on to drive the vehicle to the desired target. Control of a leader-follower formation is treated in [8], where the leader is driving along the desired trajectory. MPC is then used to stabilize the followers at their desired positions.

It is also possible to use a different approach for the path planning. In [23], distributed MPC for leader-follower formation control of unicycles is presented. Also collision avoidance is implemented. This is done by generating the trajectory of one unicycle at one sampling instant. The next sampling instant is used to create a trajectory for a different unicycle. Another example of such an approach is presented in [7], where MPC is used for collision free path planning for air traffic. To that end, a complex aircraft model is replaced with a model of a unicycle. A scenario where four planes would cross each other, is solved by selecting a plane and planning a trajectory. After that, another plane is selected and the process is repeated. This means that aircraft that can plan first have more freedom in their trajectories than the ones that can plan later. Another MPC controller is designed in [59] that also plans the paths of multiple unicycles sequentially. The trajectories that are generated are free of collisions. The robots with the highest priorities again have more freedom when choosing their paths.

There are also works that simultaneously plan paths of vehicles, such as shown in [37]. Here, a leader-follower approach is used to control a formation of unicycles using MPC. The leader trajectory is generated with the controller. The followers try to stay at a certain distance under a certain angle behind the leader. Each unicycle has a separate cost function, which is optimized iteratively, while keeping the trajectories of all agents in mind. In [48], low level navigation functions are used to control a group of unicycles. This low-level controller is combined with a high level controller. MPC is used to simultaneously generate a collision free flight path with constrained velocities and turning radii.

A comparison between different control methods is given in [53]. Here, a leader-follower approach is used to control a group of nonholonomic unmanned aerial vehicles. A cost function is introduced as usual, which is to be minimized. Then three different control methods are discussed, namely centralized, sequentially decentralized and fully decentralized. These methods are compared in a simulation with some obstacles involved. Another comparison where centralized, sequentially decentralized and fully decentralized MPC is compared, is given in [10]. The decentralized method uses presumed trajectories of neighboring robots when planning their own trajectory. If a collision with this presumed trajectory would occur, information is exchanged and a collision is avoided. This results in less communication and lowers the computational burden.

A different example of MPC formation control, where systems other than unicycles are used, is given in [21]. A receding horizon controller is used here to control a formation of hovercraft vehicles. This is done with a single cost function, containing the dynamics and control objectives of all the vehicles. After that, a stability analysis is performed using input to state stability. In this work, it is assumed that there is a communication delay between the vehicles. In [30], receding horizon control is used to control a formation of flying vehicles. The low-level control of the model is done via model inversion. The high level controller consists of a single cost function which is optimized to generate optimal trajectories. There is also a different control system active that prevents collisions between vehicles. In [14], multiple vehicles, which can have different dynamics, are controlled by grouping them in one cost function. This function is optimized to control all vehicles and to avoid collisions. In a simulation that uses simple dynamics, this works well if updates are fast enough and vehicles do not deviate far from their paths.

2.5 Conclusions

The current state of the art is that it is possible to use a wide range of controllers to control one or multiple unicycles. It is possible to use controllers that allow unicycles to follow reference signals, avoid both static and dynamic obstacles, drive in a formation with multiple unicycles, and avoid collisions with other unicycles. While it is possible to use other control methods to accomplish these control objectives, in this thesis MPC is used.

The advantages of using MPC with respect to other control methods is that it can take constraints on inputs, states, and outputs into account when calculating control inputs. It also naturally handles multivariable systems. Another advantage is that the system behavior is taken into account, which can lead to better decisions made by the controller. A price has to be paid for this however, as computation times with MPC are generally much longer than with other controllers.

Chapter 3

Model Predictive Control

This chapter starts by giving a brief historic introduction of MPC. After that, the general principle of MPC for systems with long time scale behavior is discussed. Then, a comparison is made with MPC for systems with much shorter time scale behavior. The chapter ends with a description of different control strategies that can be chosen when MPC is used to control multiple systems simultaneously.

3.1 Brief history of MPC

It has been more than 25 years since MPC first appeared in the industry. Its first applications were mainly focussed on multivariable constrained processes in chemical industries [22]. The use of MPC was limited to these very slow processes, because MPC is a computationally intensive control method, and the computers at that time were not fast enough yet. Since then, the theory has been developed further and computing power has increased. As a result, it has now become possible to use MPC in much faster processes, such as electromechanical systems.

3.2 General principle of MPC

As mentioned before, MPC is traditionally used for systems with long time scale behavior. The general working principle of this version of MPC is depicted in Figure 3.1.

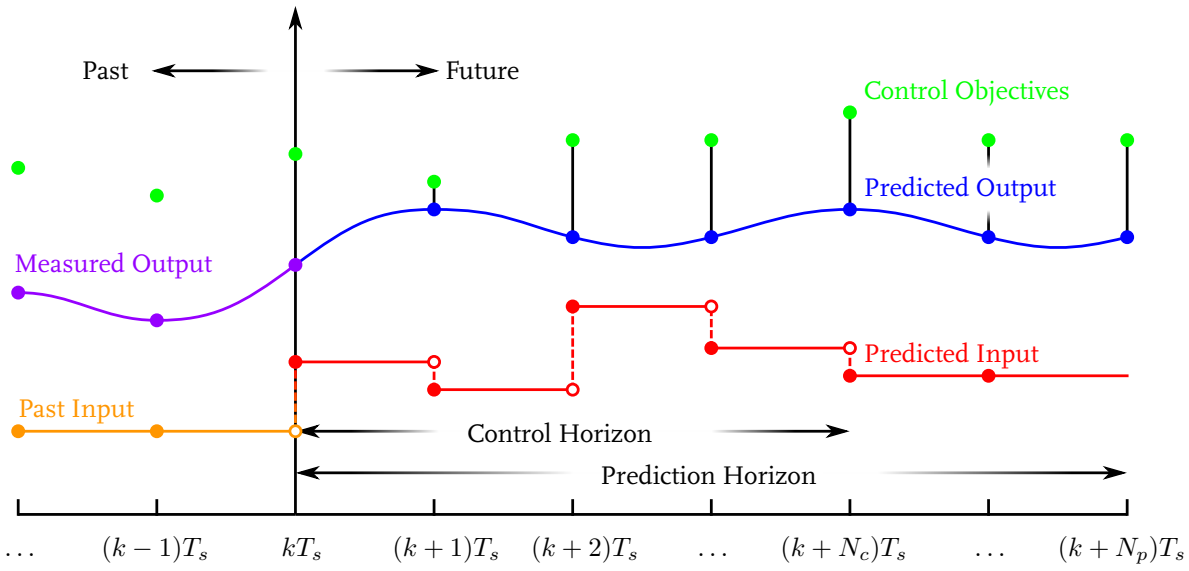


Figure 3.1: General principle of MPC of systems with a long time scale.

In this figure, the past, present, and future of a simulation or an experiment are shown. The present time is indicated by $t = kT_s$, where t [s] indicates the time instant, $k \in \mathbb{N}$, and T_s is the sampling time.

The past shows what has happened up to $t = kT_s$. Only two past instants are shown in this figure. The future is shown up to $t = (k + N_p)T_s$, where N_p is the prediction horizon, which indicates up to how many samples in the future behavior of the system needs to be taken into account.

The output of the system up to and including $t = kT_s$ is indicated by the measured output, shown in purple. The output is generally a continuous-time signal, but it is only measured at discrete sampling instants. Results of the measurements of the output are indicated with the purple markers.

When the current output of the system is known, and a model of the system is available, it might be possible to calculate what the effect of the predicted input will be on the predicted output. In this thesis, it is assumed that this is possible. The predicted input is depicted in red. It remains constant between two sampling instants as indicated by the open and full red markers. After the control horizon of N_c samples, where $1 \leq N_c < N_p$, has been reached, the predicted input remains constant. The effect of the predicted input is the predicted output, which is indicated by the blue markers.

It is desired that the system output corresponds to the control objectives, which are indicated by the green markers. How far the systems predicted output deviates from the control objectives, is indicated by the cost function. This function compares the predicted output with the control objectives and returns the performance as a scalar value, which is used as a performance measure. This is indicated with the black lines that connect the predicted output to the control objectives. A lower cost function value usually indicates that performance is better.

To obtain the predicted output that follows the control objectives as close as possible, it is possible to make use of optimization. The goal of the optimization problem is to minimize the cost function value by adjusting the predicted input. If there are constraints present in the system, such as on the inputs, states, or outputs, it is possible to include those in the optimization problem. When an optimization process is completed, and a predicted input is found that minimizes the cost function value, the first part of the predicted input is implemented. At the next time instant, the optimization process is repeated.

3.3 MPC of electromechanical systems

The description above is valid for systems with long time scale behavior. In that case, the time it takes to perform the optimization is negligible compared to the sampling time. In case of electromechanical systems, time scales are much shorter and the time it takes to solve the optimization problem has to be taken into account. This leads to the situation that is depicted in Figure 3.2.

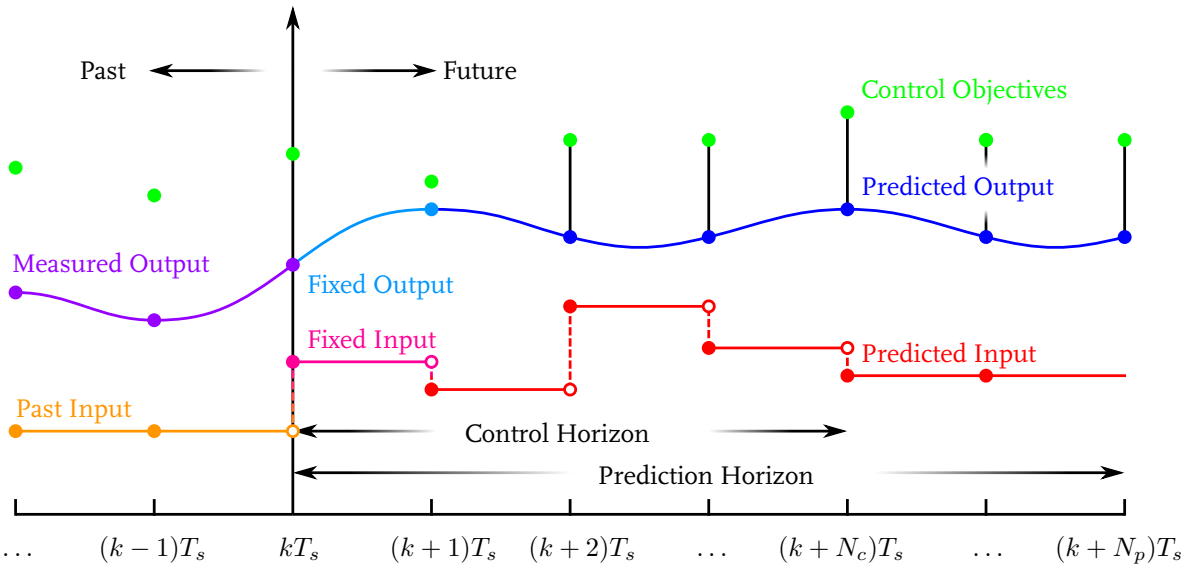


Figure 3.2: General principle of MPC of systems with a short time scale.

Because the time it takes to solve the optimization problem is much closer to the sampling time than before, the optimization process is given one sampling time to come up with a predicted control input that returns the lowest possible value of the cost function. This is done to prevent time-varying delays from occurring in the system and to keep the system behavior deterministic. However, a delay of one sampling time is introduced. This means that, in the situation that is depicted in Figure 3.2, the input that is applied to the system from $t \in [kT_s, (k+1)T_s)$ is the result of the optimization process that took place when $t \in [(k-1)T_s, kT_s)$. Because of the delay, the inputs become fixed for one sampling time, while new inputs are calculated. The fixed inputs result in fixed outputs for $t \in [kT_s, (k+1)T_s]$. This is indicated with the light blue line and marker. While the input and output are fixed, the control input is calculated that will be applied at the next time instant.

Since prediction is used, the introduction of this delay is not a problem. By using the inputs that are fixed to calculate the output that is fixed, it is still possible to predict the future system behavior. Because of the delay of one sample time, the first future instant $t = (k+1)T_s$ is now uncontrollable. Another effect of the delay is that the control horizon is now bounded by $1 < N_c < N_p$.

3.4 Controlling multiple systems in the same environment

In this thesis, a single model predictive controller is used to control multiple systems. These systems operate in a common workspace. Because of this, there is a possibility for inter-system collisions to occur. The controller has to avoid these collisions, while trying to fulfill other control objectives. To avoid collisions, the controller has to plan the paths of the systems collision free.

To plan collision free paths, the controller needs to have certain information on the systems available. In this thesis, it is assumed that the controller has the following knowledge of every system at $t = kT_s$:

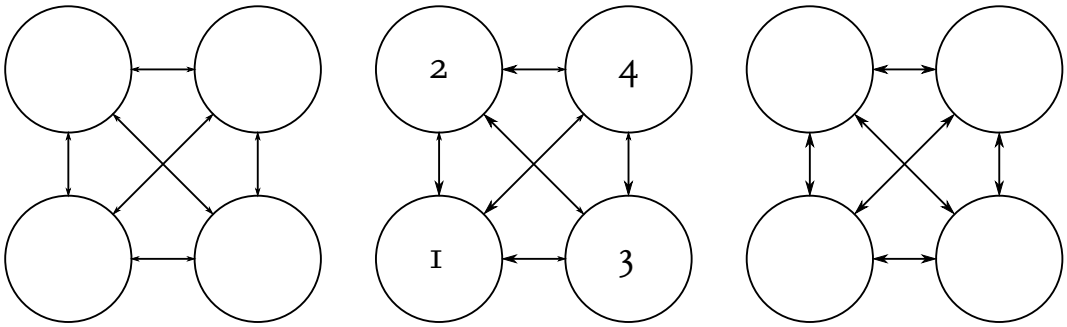
- Control objectives (e.g. tracking a reference trajectory),
- Current outputs, which are the result of measurements,
- Predicted inputs from the previous optimization from $t \in [kT_s, (k+1)T_s)$,
- Predicted outputs from the previous optimization from $t \in [(k+2)T_s, (k+N_p-1)T_s]$,

The first part of the predicted input from the previous optimization is sent to the systems as soon as $t = kT_s$. This information is used in combination with the current output of every system to determine where the systems will end up at $t = (k+1)T_s$. Because of the fixed inputs and outputs, is not possible to directly avoid a collision at $t = (k+1)T_s$. It is only possible to indirectly avoid collisions, by avoiding them at $t = (k+2)T_s$ and later time instants. However, avoiding collisions at $t = (k+2)T_s$ is the most important, as this is the last time any changes can be made to the future inputs and outputs. If the optimization results in a collision at $t = (k+2)T_s$, it will occur two sampling instants later. The predicted output from the previous optimization can be used in the planning of the paths of the systems. The path planning can be accomplished by using different control strategies, some of which are shown in Figure 3.3.

3.4.1 Fully decentralized MPC

In Figure 3.3(a), a schematic representation of fully decentralized MPC is shown. In the figure, systems are represented by circles. In this case, all systems plan their paths independent from one another. The only thing they all have to consider is where the other systems will be at $t = (k+2)T_s$, to avoid future collisions from happening. This is the result of the previous optimization, and it is indicated in the figure with the small arrows that connect the systems.

An advantage of using fully decentralized MPC is that the optimization problem is simple compared to the one of the centralized control strategy as each optimization problem is solved for only a single system. Another advantage is that the optimization problem of the systems can be solved in parallel. A disadvantage is that when systems operate in a crowded environment, it becomes hard to ensure that no inter-system collisions occur.



(a) Fully decentralized. (b) Sequentially decentralized. (c) Centralized.

Figure 3.3: Different control strategies to obtain predicted inputs.

3.4.2 Sequentially decentralized MPC

With sequentially decentralized MPC, predicted inputs are determined for one system at a time. The order in which the inputs are calculated, is determined by the priority level of the systems, indicated by the numbers in Figure 3.3(b). A system can determine its path while only avoiding lower priority systems at $t = (k + 2)T_s$. This is again indicated by the small arrows. Higher priority systems should be avoided at all controllable future instants, which is indicated with the large arrows.

The advantage of the control strategy is that inter-system collisions can now be avoided because the paths of the systems are planned sequentially. Also, the simplicity of the optimization problem is maintained. A disadvantage is that it is no longer possible to parallelize to optimization process as the path of one system depends on earlier computed paths. It is also necessary to determine the priorities that determine the order in which the paths of the systems are planned.

3.4.3 Centralized MPC

The last control strategy that is discussed here, is centralized MPC, as shown in Figure 3.3(c). With this control strategy, all systems are are treated as a single entity and predicted inputs are calculated by solving a large optimization problem. In this problem, all control objectives of all systems are combined into one cost function. This means that it is possible that fulfilling control objectives of one system can influence other systems.

Centralized MPC can lead to better performance than the other two methods as the behavior of every system is simultaneously taken into account. Also, determining the order of the systems is not necessary anymore. However, priorities can still be used by weighing control objectives of certain systems heavier than of others. A disadvantage of using this method is that the optimization problem is more complex than the previously discussed methods.

3.5 Conclusions

The controller that is discussed in this thesis is implemented in MATLAB, which is able to make use of multiple processors using the Parallel Computing Toolbox. The problem is that the use of this toolbox is only advantageous if a calculation takes roughly two seconds or longer. As the systems that are considered in this thesis operate on a shorter time scale, parallelization can not be used. Therefore, fully decentralized MPC is not discussed further. The sequentially decentralized method is discussed in this thesis. A way in which the priority order can be determined is further discussed in Section 5.7. The centralized control strategy is also discussed. Finally, a comparison between these two control methods is presented in Chapter 7.

Chapter 4

Unicycle Model

In this chapter, a kinematic model of a unicycle is given. First, the continuous-time kinematic model of the unicycle is discussed. After this, the model is exactly discretized so that it can be used by the discrete-time model predictive controller. Finally, it is discussed how to the predicted input can be used to obtain the predicted output for use in the cost function.

4.1 Kinematic model of a unicycle

A unicycle drives around using a differential drive system, where a difference in velocity between two wheels determines the velocity and angular velocity of the unicycle. A schematic model of a unicycle can be seen in Figure 4.1.

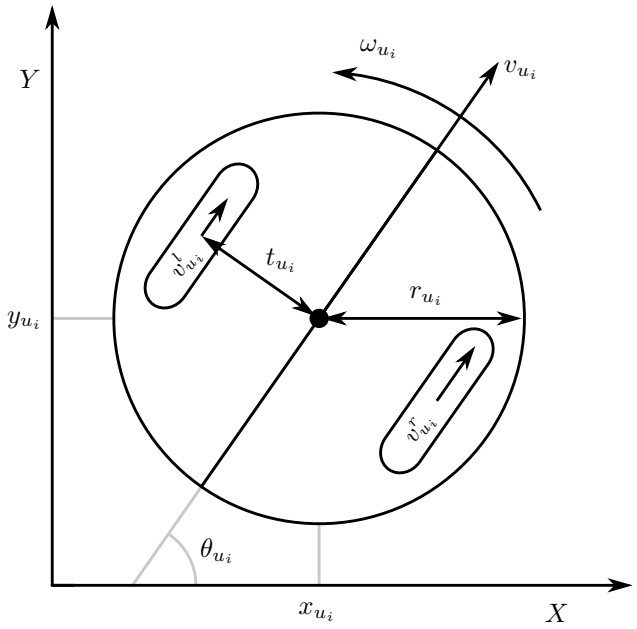


Figure 4.1: Schematic representation of a unicycle mobile robot.

In this figure, the location of the center of unicycle i is indicated with x_{u_i} [m] and y_{u_i} [m] along the X- and Y-axes respectively. The angle θ_{u_i} [rad] indicates rotation of the unicycle with respect to the X-axis, which also indicates in which direction the unicycle is driving with velocity v_{u_i} [m/s], where $v_{u_i} \in \left[v_{u_i}^{\min}, v_{u_i}^{\max}\right]$. The angle θ_{u_i} can be changed by adjusting the angular velocity ω_{u_i} [rad/s], where $\omega_{u_i} \in \left[\omega_{u_i}^{\min}, \omega_{u_i}^{\max}\right]$. There are no specific requirements on these input bounds. For example, it is possible to have a unicycle that can only drive backwards and turn left. If collision avoidance has to be ensured, v_{u_i} should be able to become zero.

The unicycle has a radius r_{u_i} [m] and a track width t_{u_i} [m], which indicates the distance from the wheel to the center of the unicycle. The velocities of the two wheels, $v^l_{u_i}$ [m/s] and $v^r_{u_i}$ [m/s] are related to v_{u_i} and ω_{u_i} as

$$\begin{aligned} v^l_{u_i} &= v_{u_i} - t_{u_i}\omega_{u_i}, \\ v^r_{u_i} &= v_{u_i} + t_{u_i}\omega_{u_i}. \end{aligned} \tag{4.1}$$

It is assumed that the velocities $v^l_{u_i}$ and $v^r_{u_i}$ that are sent to the unicycle, are tracked be a controller that is present in the unicycle itself. The controller which is used for that, is not investigated in this thesis. The continuous-time kinematics of unicycle i are given by

$$\begin{bmatrix} \dot{x}_{u_i}(t) \\ \dot{y}_{u_i}(t) \\ \dot{\theta}_{u_i}(t) \end{bmatrix} = \begin{bmatrix} \cos(\theta_{u_i}(t)) & 0 \\ \sin(\theta_{u_i}(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{u_i}(t) \\ \omega_{u_i}(t) \end{bmatrix}, \tag{4.2}$$

where the state vector consists of $x_{u_i}(t)$, $y_{u_i}(t)$, and $\theta_{u_i}(t)$. The input vector consists of $v_{u_i}(t)$ and $\omega_{u_i}(t)$. These kinematics of the unicycle restrict sideways movement, which is indicated by

$$\dot{x}_{u_i}(t)\sin(\theta_{u_i}(t)) - \dot{y}_{u_i}(t)\cos(\theta_{u_i}(t)) = 0.$$

To determine whether this constraint is holonomic or nonholonomic, it needs to be checked if the constraint can be expressed in the form of $f(x, y, \theta, t) = 0$. To check this, the constraint needs to be integrated. In this case, this is not possible because both $\dot{x}_{u_i}(t)$ and $\dot{y}_{u_i}(t)$ are in the constraint. This means that the constraint is nonholonomic, which usually makes it more difficult to design a controller for a unicycle. With MPC, no extra effort needs to be put in to the controller to overcome this constraint. In fact, if a designer would have no knowledge of this constraint, it would not create a problem, as MPC takes the kinematics of the system into account.

4.2 Exact discretization of unicycle model

A discrete-time model predictive controller requires a discrete-time kinematic description of the system. An exact discrete-time model of this continuous-time model can be obtained in the following manner [2]. It is assumed that inputs v_{u_i} and ω_{u_i} remain constant on the interval between two sampling instants t_k, which is defined as

$$t_k = t \in [kT_s, (k+1)T_s).$$

The exact discrete-time model is given by

$$\begin{bmatrix} x_{u_i}((k+1)T_s) \\ y_{u_i}((k+1)T_s) \\ \theta_{u_i}((k+1)T_s) \end{bmatrix} = \begin{bmatrix} x_{u_i}(kT_s) \\ y_{u_i}(kT_s) \\ \theta_{u_i}(kT_s) \end{bmatrix} + \int_{kT_s}^{(k+1)T_s} \begin{bmatrix} \cos(\theta_{u_i}(\lambda)) & 0 \\ \sin(\theta_{u_i}(\lambda)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{u_i}(kT_s) \\ \omega_{u_i}(kT_s) \end{bmatrix} d\lambda, \tag{4.3}$$

where the integrals still need to be solved. To solve the integrals, it is necessary to know the instantaneous value of the angle $\theta(t)$. This instantaneous value can be obtained by taking the bottom part of (4.3) and modifying the integration bounds, leading to

$$\begin{aligned} \theta_{u_i}(t) &= \theta_{u_i}(kT_s) + \int_{kT_s}^{t} \omega_{u_i}(kT_s)d\lambda \\ &= \theta_{u_i}(kT_s) + [t - kT_s]\omega_{u_i}(kT_s). \end{aligned} \tag{4.4}$$

To solve the integrals in (4.3), the instantaneous angle (4.4) is used. This means that the integrals are solved using $\theta_{u_i}(\lambda) = \theta_{u_i}(t)$. The first integral is solved as

$$\begin{aligned} &\int_{kT_s}^{(k+1)T_s} v_{u_i}(kT_s)\cos(\theta_{u_i}(\lambda))d\lambda \\ =&\int_{kT_s}^{(k+1)T_s} v_{u_i}(kT_s)\cos(\theta_{u_i}(kT_s) + [\lambda - kT_s]\omega_{u_i}(kT_s))d\lambda \\ =&\frac{v_{u_i}(kT_s)}{\omega_{u_i}(kT_s)}(\sin(\theta_{u_i}(kT_s) + T_s\omega_{u_i}(kT_s)) - \sin(\theta_{u_i}(kT_s))) \\ =&2\frac{v_{u_i}(kT_s)}{\omega_{u_i}(kT_s)}\sin\left(\frac{T_s\omega_{u_i}(kT_s)}{2}\right)\cos\left(\theta_{u_i}(kT_s) + \frac{T_s\omega_{u_i}(kT_s)}{2}\right). \end{aligned} \tag{4.5}$$

In the last step of this derivation, the sum-to-product identity is used. This identity states that

$$\sin(x) - \sin(y) = 2\cos\left(\frac{x+y}{2}\right)\sin\left(\frac{x-y}{2}\right).$$

The second integral becomes

$$\begin{aligned}
&\int_{kT_s}^{(k+1)T_s} v_{u_i}(kT_s)\sin(\theta_{u_i}(\lambda))d\lambda \\
=&\int_{kT_s}^{(k+1)T_s} v_{u_i}(kT_s)\sin(\theta_{u_i}(kT_s) + [\lambda - kT_s]\omega_{u_i}(kT_s))d\lambda \\
=&-\frac{v_{u_i}(kT_s)}{\omega_{u_i}(kT_s)}(\cos(\theta_{u_i}(kT_s) + T_s\omega_{u_i}(kT_s)) - \cos(\theta_{u_i}(kT_s))) \\
=&2\frac{v_{u_i}(kT_s)}{\omega_{u_i}(kT_s)}\sin\left(\frac{T_s\omega_{u_i}(kT_s)}{2}\right)\sin\left(\theta_{u_i}(kT_s) + \frac{T_s\omega_{u_i}(kT_s)}{2}\right).
\end{aligned} \tag{4.6}$$

A similar sum-to-product identity is used in the last step, namely

$$\cos(x) - \cos(y) = -2\sin\left(\frac{x+y}{2}\right)\sin\left(\frac{x-y}{2}\right).$$

Finally, the third integral is

$$\begin{aligned}
\int_{kT_s}^{(k+1)T_s} \omega_{u_i}(kT_s)d\lambda &= \omega_{u_i}(kT_s)\lambda\big|_{kT_s}^{(k+1)T_s} \\
&= T_s\omega_{u_i}(kT_s).
\end{aligned} \tag{4.7}$$

Now, the exact discrete-time model is derived. However, it should be noted that (4.5) and (4.6) are not defined when $\omega_{u_i}(kT_s) = 0$. A solution to this problem is to use a Taylor series approximation to determine what would be the case when $\omega_{u_i}(kT_s) = 0$. This can also be done by using l'Hôpital's rule, but in both cases, the result is

$$\lim_{\omega_{u_i}(kT_s)\to 0}\left(\frac{\sin\left(\frac{T_s}{2}\omega_{u_i}(kT_s)\right)}{\omega_{u_i}(kT_s)}\right) = \frac{T_s}{2}. \tag{4.8}$$

Now, all parts of the exact discretization are available. The exact discrete-time model is now given by combining (4.5) to (4.7) as

$$\begin{bmatrix} x_{u_i}((k+1)T_s) \\ y_{u_i}((k+1)T_s) \\ \theta_{u_i}((k+1)T_s) \end{bmatrix} = \begin{bmatrix} x_{u_i}(kT_s) + 2v_{u_i}(kT_s)\gamma_{u_i}(\omega_{u_i}(kT_s))\cos\left(\theta_{u_i}(kT_s) + \frac{T_s}{2}\omega_{u_i}(kT_s)\right) \\ y_{u_i}(kT_s) + 2v_{u_i}(kT_s)\gamma_{u_i}(\omega_{u_i}(kT_s))\sin\left(\theta_{u_i}(kT_s) + \frac{T_s}{2}\omega_{u_i}(kT_s)\right) \\ \theta_{u_i}(kT_s) + T_s\omega_{u_i}(kT_s) \end{bmatrix}, \tag{4.9}$$

where $\gamma(\omega_{u_i}(kT_s))$ is introduced, using (4.8), as

$$\gamma_{u_i}(\omega_{u_i}(kT_s)) = \begin{cases} \dfrac{\sin\left(\frac{T_s}{2}\omega_{u_i}(kT_s)\right)}{\omega_{u_i}(kT_s)}, & \text{if} \quad \omega_{u_i}(kT_s) \neq 0, \\ \dfrac{T_s}{2}, & \text{if} \quad \omega_{u_i}(kT_s) = 0. \end{cases} \tag{4.10}$$

4.3 Implementation in MATLAB

When the exact discrete-time model is implemented in MATLAB, some numerical problems arise, which are caused by (4.10). These problems occur when ω approaches the machine accuracy of MATLAB. When this happens, $\gamma_{u_i}(\omega_{u_i}(kT_s))$ takes on noticeably incorrect values, causing errors in the future prediction of outputs.

To resolve this issue, values of ω that approach the machine accuracy are increased in size slightly,

so that no noticeable numerical errors occur. This is also done when ω is zero, meaning that (4.10) can be changed to

$$\gamma_{u_i}(\omega_{u_i}(kT_s)) = \frac{\sin\left(\frac{T_s}{2}\omega_{u_i}(kT_s)\right)}{\omega_{u_i}(kT_s)}. \tag{4.11}$$

The prediction of future states that is implemented in MATLAB uses this modification of ω in combination with (4.9) and (4.11) to calculate future outputs. An advantage of using (4.11) instead of (4.10) is that there now is only one way to calculate a future state, instead of two. This simplifies the construction of the cost function (Chapter 5) and the differentiation of the cost function (Chapter 6).

4.4 Obtaining predicted outputs

The cost function uses the predicted outputs of the unicycles, as it needs to compare them to the control objectives, as discussed in Chapter 3. The comparison of the future outputs with the control objectives is discussed in the next chapter. In this work, the predicted outputs are equal to the predicted states of the unicycles. These future states can be calculated using (4.9) and (4.11). A predicted input vector $I_i(kT_s)$ is therefore defined as

$$I_i(kT_s) = \begin{bmatrix} \begin{bmatrix} \boldsymbol{v_{u_i}(kT_s)} \\ v_{u_i}((k+1)T_s) \\ \vdots \\ v_{u_i}((k+N_p-1)T_s) \end{bmatrix}^T, \begin{bmatrix} \boldsymbol{\omega_{u_i}(kT_s)} \\ \omega_{u_i}((k+1)T_s) \\ \vdots \\ \omega_{u_i}((k+N_p-1)T_s) \end{bmatrix}^T \end{bmatrix}^T. \tag{4.12}$$

Here, the bold inputs are the first inputs that are calculated while $t \in [(k-1)T_s, kT_s)$. As soon as $t = kT_s$, these inputs are sent to the unicycles and they become fixed. As discussed in the previous chapter, the control horizon N_c determines up to which point the values of v_{u_i} and ω_{u_i} can change. To reduce the complexity of the optimization problem, which is further discussed in Chapter 6, control instants N_c^i are introduced. Control instants determine at which future sampling instants the predicted input can change. At 1, a control instant is always present, so $N_c^i = [1 \ldots]$. Other instants can be added as well, as long as $1 \leq N_c^i < N_p$. For example, when $N_p = 5$, and control instants are chosen as $N_c^i = \begin{bmatrix} 1 & 3 \end{bmatrix}$, the input vector becomes

$$I_i(kT_s) = \begin{bmatrix} \begin{bmatrix} \boldsymbol{v_{u_i}(kT_s)} \\ v_{u_i}((k+1)T_s) \\ v_{u_i}((k+2)T_s) \\ v_{u_i}((k+3)T_s) \\ v_{u_i}((k+4)T_s) \end{bmatrix}^T, \begin{bmatrix} \boldsymbol{\omega_{u_i}(kT_s)} \\ \omega_{u_i}((k+1)T_s) \\ \omega_{u_i}((k+2)T_s) \\ \omega_{u_i}((k+3)T_s) \\ \omega_{u_i}((k+4)T_s) \end{bmatrix}^T \end{bmatrix}^T = \begin{bmatrix} \begin{bmatrix} \boldsymbol{v_{u_i}(kT_s)} \\ v_{u_i}((k+1)T_s) \\ v_{u_i}((k+1)T_s) \\ v_{u_i}((k+3)T_s) \\ v_{u_i}((k+3)T_s) \end{bmatrix}^T, \begin{bmatrix} \boldsymbol{\omega_{u_i}(kT_s)} \\ \omega_{u_i}((k+1)T_s) \\ \omega_{u_i}((k+1)T_s) \\ \omega_{u_i}((k+3)T_s) \\ \omega_{u_i}((k+3)T_s) \end{bmatrix}^T \end{bmatrix}^T.$$

Because of the control instants, the number of decision variables in the optimization is reduced, in this example from 8 to 4. By using the predicted input vector in (4.9) and (4.11), the predicted output matrix, $X_i(kT_s)$ can be calculated, resulting in

$$X_i(I_i(kT_s)) = \begin{bmatrix} x_{u_i}((k+2)T_s) & y_{u_i}((k+2)T_s) & \theta_{u_i}((k+2)T_s) \\ x_{u_i}((k+3)T_s) & y_{u_i}((k+3)T_s) & \theta_{u_i}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ x_{u_i}((k+N_p)T_s) & y_{u_i}((k+N_p)T_s) & \theta_{u_i}((k+N_p)T_s) \end{bmatrix}. \tag{4.13}$$

Here, the state at $t = (k+1)T_s$ is not included, as it is fixed due to the delay of one sample caused by the controller. $X_i(I_i(kT_s))$ can now be used to determine a cost function value $J_{u_i}(X_i(I_i(kT_s)))$, which is discussed in the following chapter. From here on, these notations are changed to $X_i(kT_s)$ and $J_{u_i}(kT_s)$ for brevity. The sampling time T_s and the prediction horizon N_p determine the physical prediction length, i.e. the length of the predicted path of the unicycle. A longer sampling time and a longer prediction horizon can increase this length. The physical prediction length can influence the behavior of the controller. For example, when the unicycle is driving, and encounters an obstacle that it needs to avoid, it can be beneficial to have a larger physical prediction length.

4.5 Conclusions

The continuous-time unicycle model that is described in this chapter, is exactly discretized under the assumption that the inputs remain constant over one sampling period. This results in a discrete-time model that is as close as possible to the continuous-time model. Due to numerical errors that occur with small values of ω, a modification has to be made to the exact discrete-time model. Because of this modification, it can be that the results of this model are not exact. However, errors that can occur with this modification are much smaller than errors that can occur without this modification. Finally, the modified discrete-time model is used to determine the predicted outputs that are the result of the predicted inputs. These outputs are used to determine a cost function value, which is further discussed in Chapter 5.

Chapter 5

Cost Function

This chapter discusses different elements that can make up the cost function. First, the use of the cost function is discussed. After that, terms are introduced that allow unicycles to track reference trajectories, avoid obstacles, avoid each other, and drive in a formation. The chapter ends with assembly of the total cost function, which depends on the control method that is used.

5.1 Use of the cost function

MPC uses the cost function to determine how well the system's predicted output, which is the result of a predicted input, corresponds to the specified control objectives. The cost function returns a scalar value that determines the performance. A lower cost function value indicates that the system output corresponds better to the control objectives.

The goal of MPC is to find the predicted input that results in the lowest cost function value. To find that input, an optimization algorithm is used, which is discussed in more detail in Chapter 6. From a given initial predicted input, this optimization algorithm will iteratively search for a lower value of the cost function. In this project, the steepest descent method and Newton's method are used for this. These methods use the gradient and the Hessian of the cost function with respect to the inputs that can be altered. All terms in the cost function are formulated with this in mind, meaning that the cost function can be differentiated to obtain the gradient. The gradient can then be differentiated to obtain the Hessian.

The optimization algorithm will try to find a predicted output, in this case a trajectory, that corresponds the best with the control objectives. These objectives are specified from $t = (k+2)T_s$ to $t = (k+N_p)T_s$. The control objectives are not specified at $(k+1)T_s$ because this future time instant is not controllable. This is because for $t \in [kT_s, (k+1)T_s)$, inputs and outputs of the system are fixed. The relative importance of the control objectives are determined by penalty terms that penalized deviating from the given control objectives. It is possible to use different penalty values at different future instants. By doing this, different convergence behavior to the control objectives can be obtained.

5.2 Tracking a reference trajectory

If it is desired that unicycle i follows a reference trajectory, reference tracking terms should be included in the cost function. The reference trajectory has to be specified as

$$R_i(kT_s) = \begin{bmatrix} x_{r_i}((k+2)T_s) & y_{r_i}((k+2)T_s) & \theta_{r_i}((k+2)T_s) \\ x_{r_i}((k+3)T_s) & y_{r_i}((k+3)T_s) & \theta_{r_i}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ x_{r_i}((k+N_p)T_s) & y_{r_i}((k+N_p)T_s) & \theta_{r_i}((k+N_p)T_s) \end{bmatrix}.$$

Here, $x_{r_i}(m)$, $y_{r_i}(m)$, and $\theta_{r_i}(m)$ indicate the locations and rotation of reference trajectory at a future instant m respectively. There are no requirements on the reference trajectory, so it is possible to specify a trajectory that a unicycle can not follow, such as specifying a position that a unicycle should drive to,

which can not be reached in one sample. This means that it is not necessary to specify a path to this position, as this is handled by the controller. Figure 5.1 shows a situation where a unicycle should track a reference trajectory. Note that the same color scheme is used as in Figure 3.2.

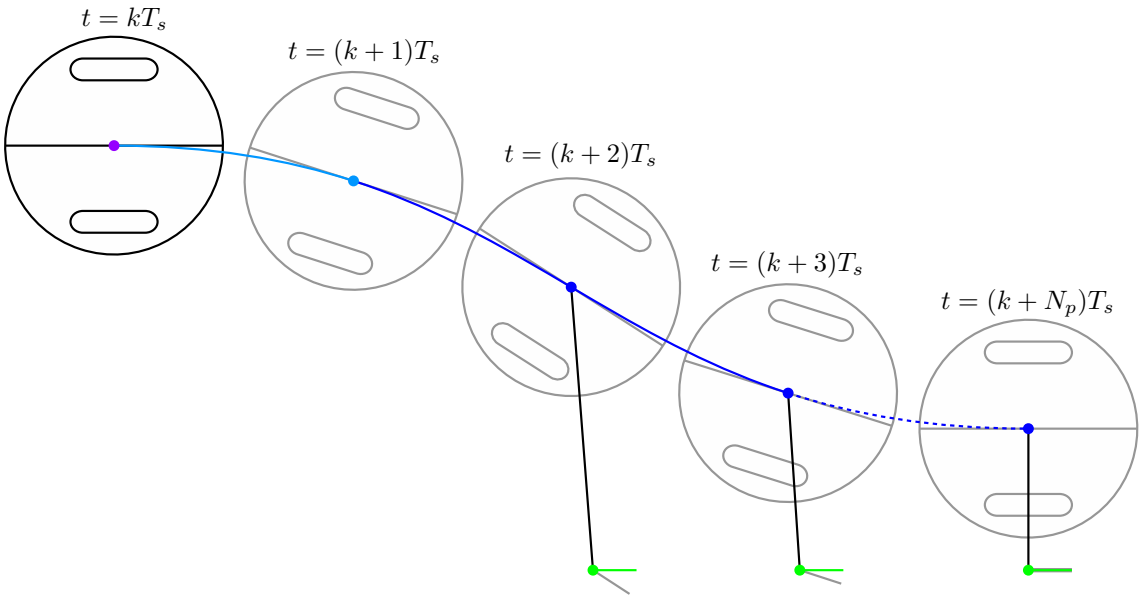


Figure 5.1: Unicycle tracking a reference signal.

At $t = kT_s$, the location of the unicycle is known from measurement, as indicated with the purple marker. The output is fixed for $t \in [kT_s, (k+1)T_s]$, which is indicated with the light blue line and marker. After that, control is regained over future states and the reference trajectory can be tracked. To do this, the distance to the reference trajectory and the absolute angular difference are defined. These will be used to penalize deviating from the reference trajectory, by increasing the cost function value. The distance and absolute angular difference at a future instant m are given by

$$D_{u_i}^{r_i}(m) = \sqrt{(x_{r_i}(m) - x_{u_i}(m))^2 + (y_{r_i}(m) - y_{u_i}(m))^2},$$
$$A_{u_i}^{r_i}(m) = (\theta_{r_i}(m) - \theta_{u_i}(m))^2.$$

To penalize $D_{u_i}^{r_i}(m)$ and $A_{u_i}^{r_i}(m)$ when they are not equal to zero, two scalar penalty terms $P_{u_i}^{r_i}(m)$ and $Q_{u_i}^{r_i}(m)$ are defined. The cost of tracking a reference trajectory for unicycle i is given by

$$J_{u_i}^{R_i}(kT_s) = \sum_{m=k+2}^{k+N_p} \left(P_{u_i}^{r_i}(mT_s) D_{u_i}^{r_i}(mT_s) + Q_{u_i}^{r_i}(mT_s) A_{u_i}^{r_i}(mT_s) \right). \tag{5.1}$$

By adding (5.1) to the cost function, the unicycles can track a reference trajectory.

5.3 Avoiding collisions with circular obstacles

To avoid obstacles that are in the same workspace as a unicycle, an obstacle avoidance term should be included in the cost function. In this thesis, the obstacle avoidance term is derived for circular obstacles only. This is done because it is straightforward to calculate the distance between a unicycle and a circular obstacle, which is necessary to detect potential collisions. To detect a collision, the location and radius of a circular obstacle i should be defined as

$$O_i(kT_s) = \begin{bmatrix} x_{o_i}((k+2)T_s) & y_{o_i}((k+2)T_s) & r_{o_i}((k+2)T_s) \\ x_{o_i}((k+3)T_s) & y_{o_i}((k+3)T_s) & r_{o_i}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ x_{o_i}((k+N_p)T_s) & y_{o_i}((k+N_p)T_s) & r_{o_i}((k+N_p)T_s) \end{bmatrix}.$$

The location of the center of a circular obstacle i is indicated by $x_{o_i}(m)$ and $y_{o_i}(m)$. The radius of the obstacle is indicated by $r_{o_i}(m)$. When obstacles are static, so the object's location and radius do not

change over time, it is theoretically possible to guarantee that no collisions occur. This can be done by stopping a unicycle if future inputs will result in a collision with that obstacle. A requirement for this is that v_{u_i} can become zero. After a unicycle has stopped, a new optimization at a next time instant can result in a collision free path. It is also possible to have obstacles that vary their location and radius over time. When obstacles are dynamic, collision avoidance can no longer be guaranteed. However, if the location and/or radius of the obstacles varies slow enough, and the situation is not too crowded, the obstacles will still be avoided. A situation where a unicycle is driving in the presence of a circular obstacle is shown in Figure 5.2.

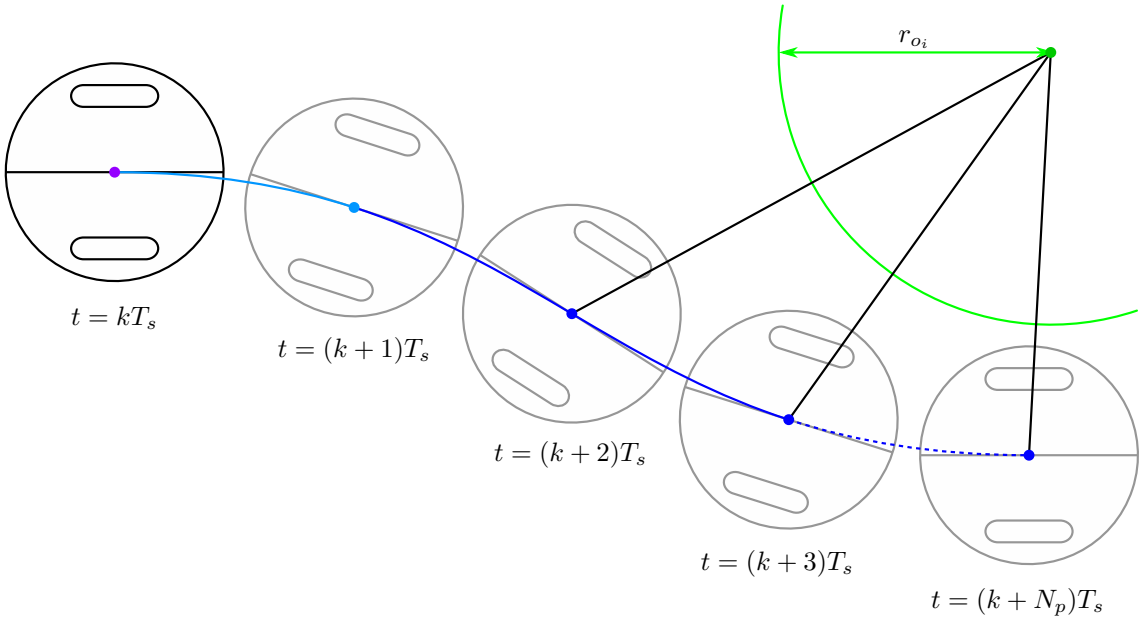


Figure 5.2: Unicycle avoiding collisions with a circular obstacle.

The distance from the center of unicycle i to the center of a circular obstacle j at a future instant m can be calculated as

$$D_{u_i}^{o_j}(m) = \sqrt{\left(x_{u_i}(m) - x_{o_j}(m)\right)^2 + \left(y_{u_i}(m) - y_{o_j}(m)\right)^2}.$$

When the distance between the center of the unicycle and the object is known, it can be checked whether a collision will occur or not. For this, the radii of the unicycle and the obstacle should be known. However, it should also be taken into account what happens between two sampling instants, as can be seen in Figure 5.3. In Figure 5.3(a), a unicycle is depicted at $t = mT_s$ and $t = (m+1)T_s$. In between the instants the unicycle is driving with $v_{u_i}^s = \max(|v_{u_i}^{\min}|, |v_{u_i}^{\max}|)$ and $\omega_{u_i}^s = \max(|\omega_{u_i}^{\min}|, |\omega_{u_i}^{\max}|)$, so the unicycle is driving at maximum velocity and turning with maximum angular velocity. At both $t = mT_s$ and $t = (m+1)T_s$, no collision is detected, while a collision with the obstacle does occur. To avoid this situation, the following analysis is made. It is required that

$$v_{u_i}^s T_s \leq 2r_{u_i}. \tag{5.2}$$

The unicycle is driving on a circular path. The time it takes to complete a full circle can be calculated as

$$t_{u_i}^s = \frac{2\pi}{\omega_{u_i}^s}. \tag{5.3}$$

The circumference of this circle is

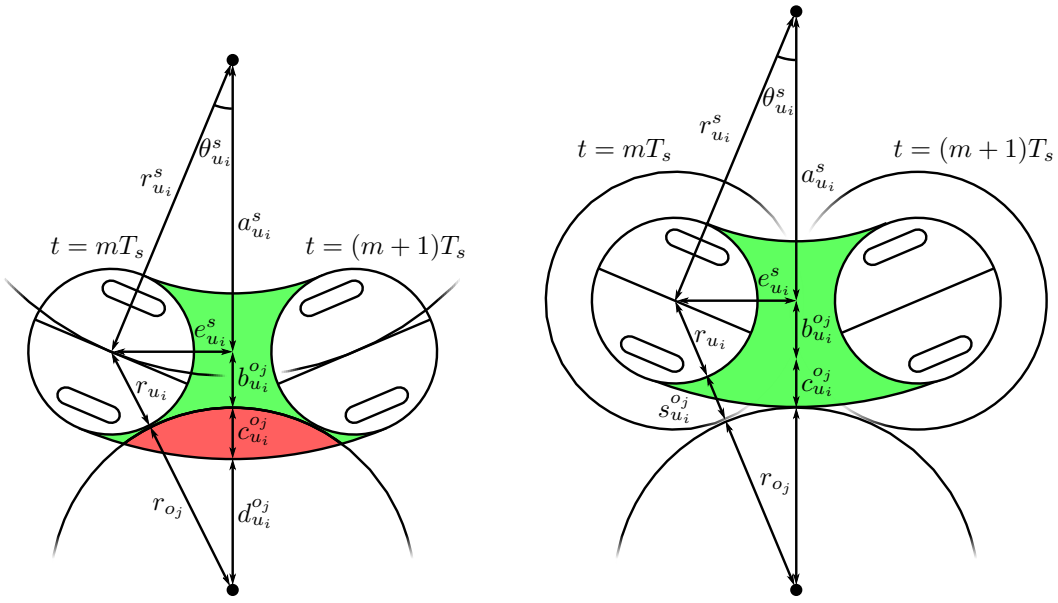
$$C_{u_i}^s = v_{u_i}^s t_{u_i}^s, \tag{5.4}$$

and the radius of the circle is therefore

$$r_{u_i}^s = \frac{C_{u_i}^s}{2\pi}. \tag{5.5}$$

The length of the arc of the circle which connects the center of the unicycle at $t = mT_s$ with the center at $t = (m+1)T_s$ is

$$A_{u_i}^s = v_{u_i}^s T_s. \tag{5.6}$$



(a) Worst case inter-sample collision occurring.

(b) Worst case inter-sample collision avoided.

Figure 5.3: Preventing inter-sample collisions with a circular obstacle.

This arc length is used to determine the angle $\theta^s_{u_i}$ as

$$\theta^s_{u_i} = \frac{2\pi}{2}\frac{A^s_{u_i}}{C^s_{u_i}}. \tag{5.7}$$

Now, the length of line segments $c^{o_j}_{u_i}$ and $bc^{o_j}_{u_i}$ can be calculated as

$$\begin{aligned}
a^s_{u_i} &= r^s_{u_i}\cos(\theta^s_{u_i}), && (5.8)\\
e^s_{u_i} &= r^s_{u_i}\sin(\theta^s_{u_i}), && (5.9)\\
abc^{o_j}_{u_i} &= a^s_{u_i} + b^{o_j}_{u_i} + c^{o_j}_{u_i} = r^s_{u_i} + r_{u_i},\\
bcd^{o_j}_{u_i} &= b^{o_j}_{u_i} + c^{o_j}_{u_i} + d^{o_j}_{u_i} = \sqrt{\left(r_{o_j} + r_{u_i}\right)^2 - {e^s_{u_i}}^2},\\
cd^{o_j}_{u_i} &= c^{o_j}_{u_i} + d^{o_j}_{u_i} = r_{o_j},\\
c^{o_j}_{u_i} &= abc^{o_j}_{u_i} + cd^{o_j}_{u_i} - a^s_{u_i} - bcd^{o_j}_{u_i},\\
bc^{o_j}_{u_i} &= b^{o_j}_{u_i} + c^{o_j}_{u_i} = abc^{o_j}_{u_i} - a^s_{u_i}.
\end{aligned}$$

When the unicycle and its circular path are shifted upwards by $c^{o_j}_{u_i}$, a collision with the obstacle is avoided, as shown in Figure 5.3(b). This shift can be used to calculate a safe distance $s^{o_j}_{u_i}$ which needs to be added to the unicycle radius in order to guarantee that no inter-sample collisions with obstacles occur. This safe distance $s^{o_j}_{u_i}$ can be calculated as

$$s^{o_j}_{u_i} = \sqrt{\left(r_{o_j} + bc^{o_j}_{u_i}\right)^2 + {e^s_{u_i}}^2} - r_{u_i} - r_{o_j}.$$

When this safe distance is added to the original radius of the unicycle, it can be guaranteed that no inter-sample collisions with obstacles occur, assuming that the obstacle is static. It can now be checked if a collision $C^{o_j}_{u_i}(m)$ will occur as

$$C^{o_j}_{u_i}(m) = \begin{cases} 0, & \text{if} \quad D^{o_j}_{u_i}(m) - r_{u_i} - s^{o_j}_{u_i} - r_{o_j} > 0 \quad \text{(no collision)},\\ 1, & \text{if} \quad D^{o_j}_{u_i}(m) - r_{u_i} - s^{o_j}_{u_i} - r_{o_j} \leq 0 \quad \text{(collision)}. \end{cases}$$

The cost of approaching an obstacle, or colliding with it, is specified as

$$J^{o_j}_{u_i}(m) = \begin{cases} \dfrac{1}{D^{o_j}_{u_i}(m) - r_{u_i} - s^{o_j}_{u_i} - r_{o_j}}, & \text{if} \quad C^{o_j}_{u_i}(m) = 0 \quad \text{(no collision)},\\ \dfrac{P^{o_j}_{u_i}(m)}{D^{o_j}_{u_i}(m)}, & \text{if} \quad C^{o_j}_{u_i}(m) = 1 \quad \text{(collision)}. \end{cases} \tag{5.10}$$

Here, a scalar penalty term $P_{u_i}^{o_j}(m)$ is introduced to penalize a collision with a circular obstacle. When no collision will occur, the cost of approaching an obstacle is determined by the first function of (5.10) which uses the distance that is left between the obstacle and the unicycle. As a unicycle moves closer to an obstacle, the cost increases. If a collision will occur, the second function is used to determine the cost. Because $D_{u_i}^{o_j}(m) - r_{u_i} - s_{u_i}^{o_j} - r_{o_j} \leq 0$ in this area, the first function would become negative here, which is not desired. To remedy this, only the remaining distance between the center of the unicycle and the object is used in the denominator. To indicate the difference between the two areas, the cost is multiplied with $P_{u_i}^{o_j}(m)$ to penalize collisions more. The cost returned by (5.10) is finite, except when the distance between the centers is zero. A schematic overview of the cost (5.10) as a function of the distance between the centers, is shown in Figure 5.4.

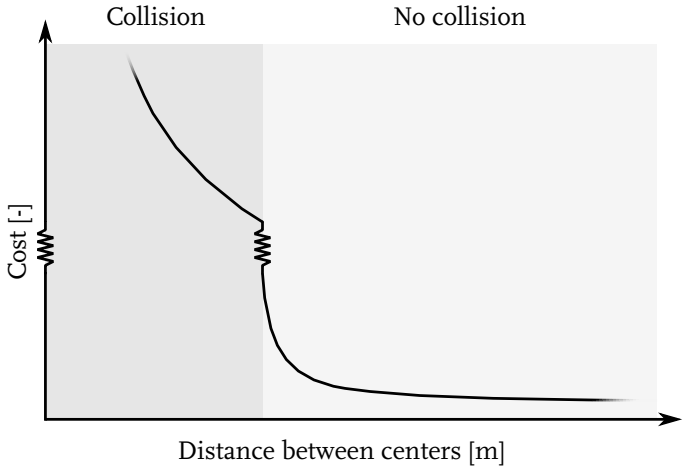


Figure 5.4: Cost of approaching a circular obstacle or another unicycle.

The value of $P_{u_i}^{o_j}(m)$ should be orders larger than penalties on reference tracking and formation keeping. In this way, avoiding collisions will weigh heavier than fulfilling other control objectives. As discussed earlier, it is possible to use a different penalty at a different future instant, in fact here it is even recommended. By choosing $P_{u_i}^{o_j}(m)$ higher at $t = (k+2)T_s$ than at other future time instants, a difference is made between a collision that will happen sooner and a collision that will happen later.

The use of two functions in the different areas leads to a discontinuity when switching between the functions. This means that it can occur that not colliding is penalized more than colliding. However, this does not create a problem as the optimization algorithm, that is discussed further in Chapter 6, uses the gradient and the Hessian to search for a lower cost function value. As can be seen in Figure 5.4, in both areas the gradient and the Hessian will point to a lower cost function value if the unicycle moves further away from an obstacle. When the gradient and Hessian are provided to the optimization algorithm, the predicted input can be changed to prevent a collision with the obstacle. To obtain the cost of avoiding collisions with circular obstacles, (5.10) is combined in one equation, resulting in

$$J_{u_i}^{O_j}(kT_s) = \sum_{m=k+2}^{k+N_p} \left(\frac{1 + C_{u_i}^{o_j}(mT_s)\left(P_{u_i}^{o_j}(mT_s) - 1\right)}{D_{u_i}^{o_j}(mT_s) - \left(1 - C_{u_i}^{o_j}(mT_s)\right)\left(r_{u_i} + s_{u_i}^{o_j} + r_{o_j}\right)} \right). \tag{5.11}$$

By adding (5.11) to the cost function, the unicycles can avoid circular obstacles.

5.4 Avoiding collisions with other unicycles

Avoiding of other unicycles works in a similar way as the avoiding of circular obstacles. The biggest difference is that all the unicycles can be controlled, while the objects can not. Depending on the control method that is used, unicycles have to avoid each other at only the first controllable future time instant, or at every controllable future time instant.

When sequentially decentralized MPC is used, unicycles can plan their trajectories in the order that

results from their respective priorities. A unicycle needs to avoid lower priority unicycles only at the first controllable future instant $t = (k+2)T_s$. Higher priority unicycles need to be avoided at every controllable future time instant, as they already planned their trajectories. The priority of a unicycle i is indicated by $P_{u_i}(kT_s)$. It is assumed that the priorities of the unicycles are never equal to keep the system behavior deterministic. A way in which the priorities of the unicycles can be determined, is discussed in Section 5.7.

If centralized MPC is used, all unicycles plan their trajectories at the same time. This means that a change in trajectory of one unicycle can influence the trajectories of every other unicycle. Because of this, all unicycles should avoid all other unicycles at every controllable future time instant when using this control strategy.

To make unicycle i avoid unicycle j that operates in the same workspace, a unicycle avoidance term should be included in the cost function. When sequentially decentralized MPC is used and $P_{u_i}(kT_s) > P_{u_j}(kT_s)$, the future state of a unicycle j is given by

$$X_j(kT_s) = \begin{bmatrix} x_{u_j}((k+2)T_s) & y_{u_j}((k+2)T_s) & \theta_{u_j}((k+2)T_s) \end{bmatrix}.$$

This state is obtained from the previous optimization result at $(k+3)T_s$. In the case that $P_{u_i}(kT_s) < P_{u_j}(kT_s)$, or if centralized MPC is used, the future states are given by

$$X_j(kT_s) = \begin{bmatrix} x_{u_j}((k+2)T_s) & y_{u_j}((k+2)T_s) & \theta_{u_j}((k+2)T_s) \\ x_{u_j}((k+3)T_s) & y_{u_j}((k+3)T_s) & \theta_{u_j}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ x_{u_j}((k+N_p)T_s) & y_{u_j}((k+N_p)T_s) & \theta_{u_j}((k+N_p)T_s) \end{bmatrix}.$$

A situation where a unicycle is driving in the presence of another unicycle is shown in Figure 5.5.

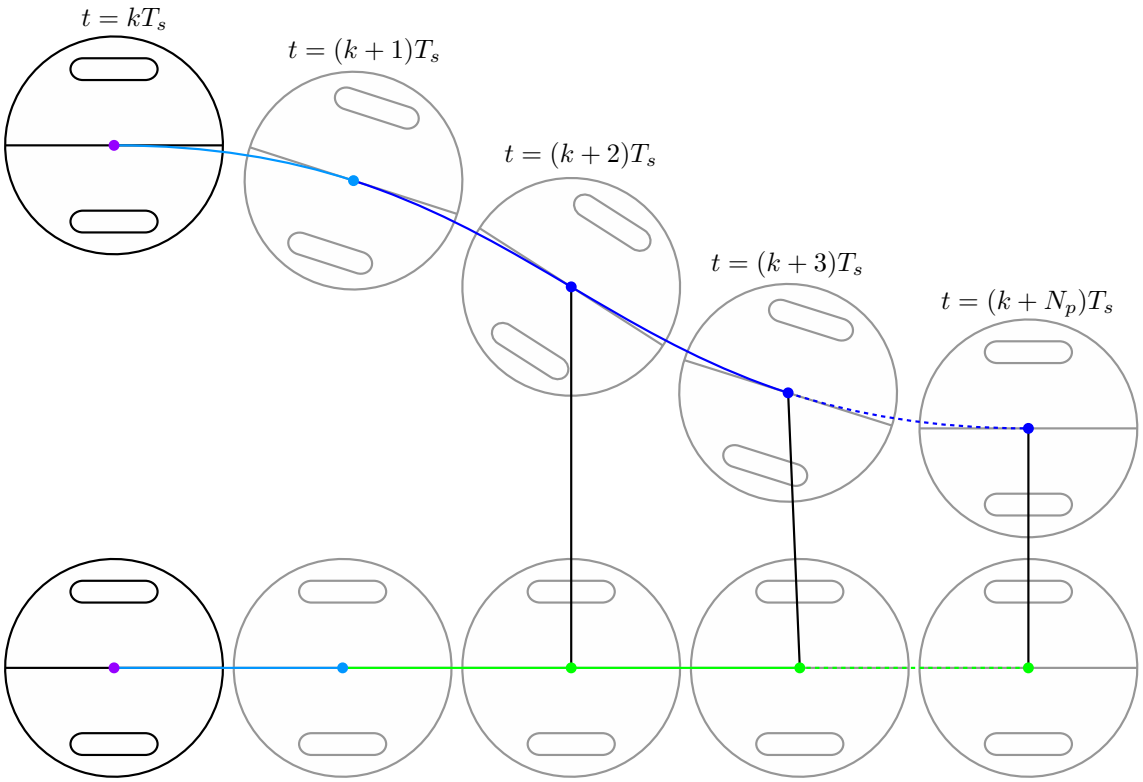
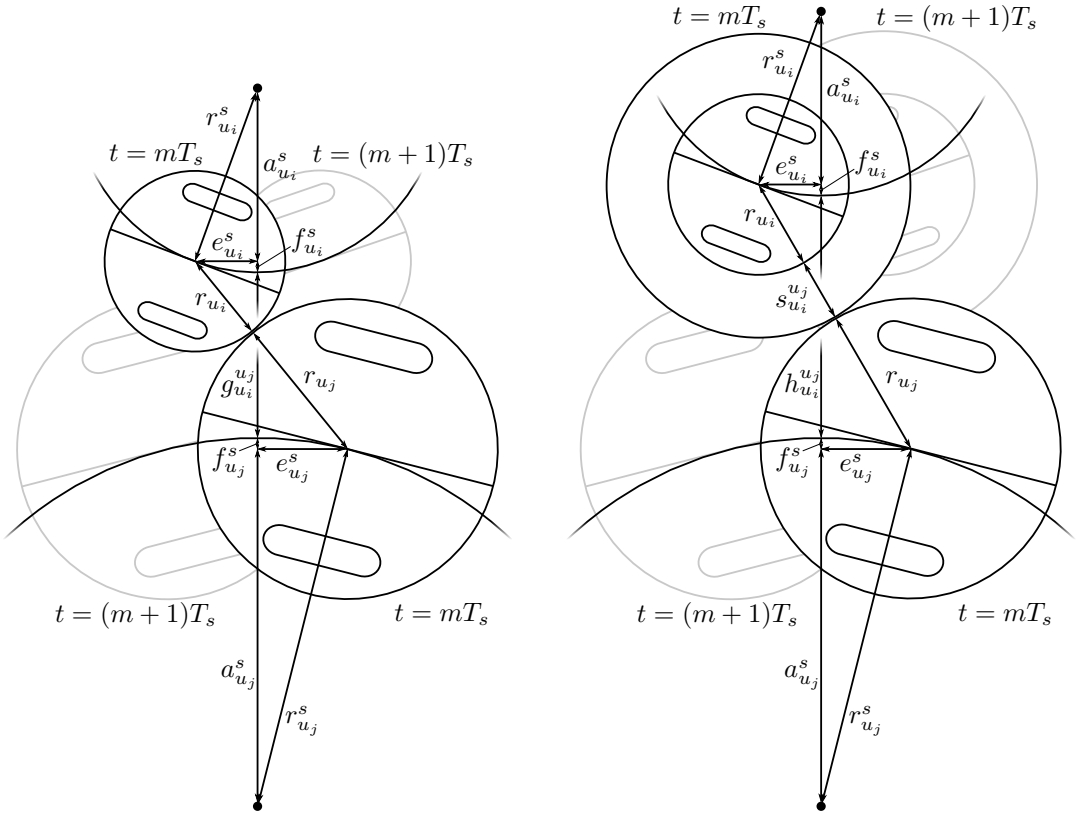


Figure 5.5: Unicycle avoiding collisions with another unicycle.

To avoid other unicycles, the distance from the center of unicycle i to the center of a unicycle j at a future instant m should be known. It can be calculated as

$$D_{ui}^{u_j}(m) = \sqrt{\left(x_{u_i}(m) - x_{u_j}(m)\right)^2 + \left(y_{u_i}(m) - y_{u_j}(m)\right)^2}. \tag{5.12}$$

When the distance between the centers of the unicycles is known, it can be checked whether a collision will occur or not. For this, the radii of the unicycles r_{u_i} and r_{u_j} should be known. However, it should also be taken into account what happens between two sampling instants, as can be seen in Figure 5.6.



(a) Worst case inter-sample collision occurring.

(b) Worst case inter-sample collision avoided.

Figure 5.6: Preventing inter-sample collisions with another unicycle.

In Figure 5.6(a), two unicycles are depicted at $t = mT_s$ and $t = (m+1)T_s$. In between the instants the unicycles are driving with $v^s_{u_i}$, $v^s_{u_j}$ $\omega^s_{u_i}$, and $\omega^s_{u_j}$, so they are both driving at their maximum velocities and turning with their maximum angular velocities. At both $t = mT_s$ and $t = (m+1)T_s$, no collision is detected, while a collision between the unicycles does occur. To avoid this situation, an analysis of the situation is made. For this analysis, it is required that

$$\begin{aligned} v^s_{u_i} T_s &\leq 2r_{u_i}, \\ v^s_{u_j} T_s &\leq 2r_{u_j}. \end{aligned} \tag{5.13}$$

To obtain the length of line segments $r^s_{u_i}$, $r^s_{u_j}$, $a^s_{u_i}$, $a^s_{u_j}$, $e^s_{u_i}$, and $e^s_{u_j}$, (5.3) to (5.9) can be used. At $t = (m + \frac{1}{2})T_s$, the centers of the unicycles are closest to each other. The distance between the centers $g^{u_j}_{u_i}$ at this time instant can be obtained as

$$\begin{aligned} f^s_{u_i} &= r^s_{u_i} - a^s_{u_i}, \qquad f^s_{u_j} = r^s_{u_j} - a^s_{u_j}, \\ fgf^{u_j}_{u_i} &= f^s_{u_i} + g^{u_j}_{u_i} + f^s_{u_j} = \sqrt{(r_{u_i} + r_{u_j})^2 - (e^s_{u_i} + e^s_{u_j})^2}, \\ g^{u_j}_{u_i} &= fgf^{u_j}_{u_i} - f^s_{u_i} - f^s_{u_j}. \end{aligned}$$

The collision depth of the unicycles at this time instant can be calculated as

$$d^{u_j}_{u_i} = r_{u_i} + r_{u_j} - g^{u_j}_{u_i}.$$

With this length known, unicycle i and its circular path are shifted upwards by $d^{u_j}_{u_i}$ as is shown in Figure 5.6(b). Here, $h^{u_j}_{u_i} = g^{u_j}_{u_i} + d^{u_j}_{u_i}$. After this shift, no inter-sample collision can occur anymore.

The shift requires that a safe distance is added to the radius of unicycle i. The safe distance $s_{u_i}^{u_j}$ can be obtained as

$$s_{u_i}^{u_j} = \sqrt{(f_{u_i}^s + h_{u_i}^{u_j} + f_{u_j}^s)^2 + (e_{u_i}^s + e_{u_j}^s)^2} - r_{u_i} - r_{u_j}.$$

When this safe distance is added to radii of the two unicycles, it can be guaranteed that no inter-sample collisions between unicycles occur. It can now be checked if a collision $C_{u_i}^{u_j}(m)$ occurs at a given future sample as

$$C_{u_i}^{u_j}(m) = \begin{cases} 0, & \text{if} \quad D_{u_i}^{u_j}(m) - r_{u_i} - s_{u_i}^{u_j} - r_{u_j} > 0 \quad \text{(no collision)}, \\ 1, & \text{if} \quad D_{u_i}^{u_j}(m) - r_{u_i} - s_{u_i}^{u_j} - r_{u_j} \leq 0 \quad \text{(collision)}. \end{cases}$$

The cost of approaching another unicycle, or colliding with it, is specified as

$$J_{u_i}^{u_j}(m) = \begin{cases} \dfrac{1}{D_{u_i}^{u_j}(m) - r_{u_i} - s_{u_i}^{u_j} - r_{u_j}}, & \text{if} \quad C_{u_i}^{u_j}(m) = 0 \quad \text{(no collision)}, \\ \dfrac{P_{u_i}^{u_j}(m)}{D_{u_i}^{u_j}(m)}, & \text{if} \quad C_{u_i}^{u_j}(m) = 1 \quad \text{(collision)}. \end{cases} \tag{5.14}$$

Here, a scalar penalty term $P_{u_i}^{u_j}(m)$ is introduced to penalize a collision with another unicycle. This penalty term has the same properties and requirements as $P_{u_i}^{o_j}(m)$, as is discussed in Section 5.3. A schematic overview of the cost (5.14) as a function of the distance is shown in Figure 5.4. As before, it is possible to combine (5.14) to obtain an expression for the cost of avoiding another unicycle as

$$J_{u_i}^{U_j}(kT_s) = \sum_{m=k+2}^{n(kT_s)} \left(\frac{1 + C_{u_i}^{u_j}(mT_s)\left(P_{u_i}^{u_j}(mT_s) - 1\right)}{D_{u_i}^{u_j}(mT_s) - \left(1 - C_{u_i}^{u_j}(mT_s)\right)\left(r_{u_i} + s_{u_i}^{u_j} + r_{u_j}\right)} \right). \tag{5.15}$$

Here, $n(kT_s)$ depends on the control strategy and the priority of the unicycle as

$$n(kT_s) = \begin{cases} k+2, & \text{if } P_{u_i}(kT_s) > P_{u_j}(kT_s), \\ k+N_p, & \text{if } P_{u_i}(kT_s) < P_{u_j}(kT_s), \text{ or centralized MPC.} \end{cases} \tag{5.16}$$

As unicycles can not collide with themselves, $J_{u_i}^{U_i}(kT_s) = 0$. By adding (5.15) to the cost function, the unicycles can avoid other unicycles.

5.5 Driving in formation with other unicycles

If it is desired that multiple unicycles drive in a formation, a formation should first be specified. A unicycle can be coupled to other unicycles in three ways, which are depicted in Figure 5.7. It is possible create leader-follower formations, virtual structures, or a mix of these two.

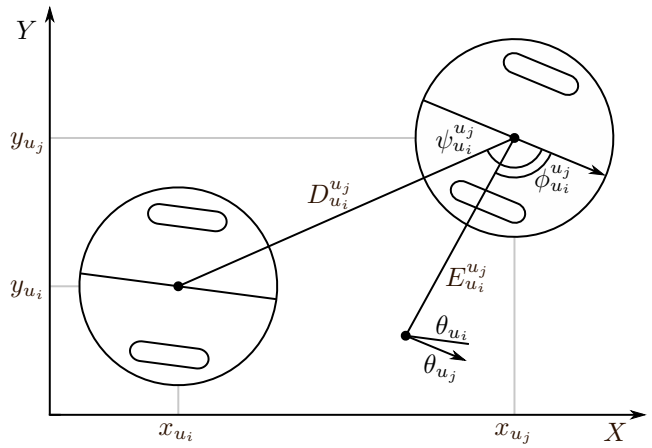


Figure 5.7: Formation that unicycle i should keep from unicycle j.

Here, $D_{u_i}^{u_j}(m)$, given by (5.12), indicates the distance between the centers of unicycle i and j. The desired distance that should be kept between unicycles is indicated with $E_{u_i}^{u_j}(m)$. The angle that

unicycle i has with respect to unicycle j is indicated by $\psi_{u_i}^{u_j}(m)$ and can be calculated as

$$\psi_{u_i}^{u_j}(m) = \operatorname{atan2}\left(y_{u_i}(m) - y_{u_j}(m), x_{u_i}(m) - x_{u_j}(m)\right). \tag{5.17}$$

The desired angle that should be kept between the unicycles is indicated with $\phi_{u_i}^{u_j}(m)$. When $E_{u_i}^{u_j}(m)$ and $\phi_{u_i}^{u_j}(m)$ are specified, the position of the unicycle is fully constrained. However, the angle of the unicycle is not. To constrain this angle, a third coupling term is introduced which is the coupling of θ_{u_i} to θ_{u_j}. This last term can be used to fully constrain the unicycle when it is in the desired position. However, when the unicycle is not yet in position, this coupling overconstrains the formation goals of the unicycle.

To make a unicycle i drive in formation with respect to unicycle j, a formation keeping term should be included in the cost function. When sequentially decentralized MPC is used and $P_{u_i}(kT_s) > P_{u_j}(kT_s)$, the future formation of a unicycle j is given by

$$F_i^j(kT_s) = \begin{bmatrix} E_{u_i}^{u_j}((k+2)T_s) & \phi_{u_i}^{u_j}((k+2)T_s) & \theta_{u_j}((k+2)T_s) \\ E_{u_i}^{u_j}((k+3)T_s) & \phi_{u_i}^{u_j}((k+3)T_s) & \theta_{u_j}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ E_{u_i}^{u_j}((k+N_p-1)T_s) & \phi_{u_i}^{u_j}((k+N_p-1)T_s) & \theta_{u_j}((k+N_p-1)T_s) \end{bmatrix}.$$

This formation can only be specified until $(k+N_p-1)T_s$ because the states of unicycle j are only available up to this time instant. In this case, unicycle i needs to base its decision on the states of unicycle j that was calculated at the previous time instant. These are the future states from $(k+3)T_s$ till $(k+N_p)T_s$ that were calculated at the previous time instant, and are shifted by the sampling time. When $P_{u_i}(kT_s) < P_{u_j}(kT_s)$, or if centralized MPC is used, the future formation is given by

$$F_i^j(kT_s) = \begin{bmatrix} E_{u_i}^{u_j}((k+2)T_s) & \phi_{u_i}^{u_j}((k+2)T_s) & \theta_{u_j}((k+2)T_s) \\ E_{u_i}^{u_j}((k+3)T_s) & \phi_{u_i}^{u_j}((k+3)T_s) & \theta_{u_j}((k+3)T_s) \\ \vdots & \vdots & \vdots \\ E_{u_i}^{u_j}((k+N_p)T_s) & \phi_{u_i}^{u_j}((k+N_p)T_s) & \theta_{u_j}((k+N_p)T_s) \end{bmatrix}.$$

The cost of staying in a formation is given by

$$\begin{aligned} J_{u_i}^{F_j}(kT_s) = \sum_{m=k+2}^{o(kT_s)} \Big(& Q_{u_i}^{u_j}(mT_s)\left(E_{u_i}^{u_j}(mT_s) - D_{u_i}^{u_j}(mT_s)\right)^2 + \dots \\ & R_{u_i}^{u_j}(mT_s)\left(\phi_{u_i}^{u_j}(mT_s) - \psi_{u_i}^{u_j}(mT_s)\right)^2 + \dots \\ & S_{u_i}^{u_j}(mT_s)\left(\theta_{u_j}(mT_s) - \theta_{u_i}(mT_s)\right)^2 \Big) \end{aligned} \tag{5.18}$$

Three penalty terms $Q_{u_i}^{u_j}(m)$, $R_{u_i}^{u_j}(m)$, and $S_{u_i}^{u_j}(m)$ are introduced to penalize deviating from a desired formation. Here, $o(kT_s)$ depends on the control strategy and the priority of the unicycle as

$$o(kT_s) = \begin{cases} k+N_p-1, & \text{if } P_{u_i}(kT_s) > P_{u_j}(kT_s), \\ k+N_p, & \text{if } P_{u_i}(kT_s) < P_{u_j}(kT_s), \text{ or centralized MPC.} \end{cases} \tag{5.19}$$

As a unicycle can not drive solely in a formation, $J_{u_i}^{F_i}(kT_s) = 0$. By adding (5.18) to the cost function, the unicycles can drive in a formation.

5.6 Total cost function

Depending on what control objectives should be fulfilled, and what control method is used, the cost function will be different. Terms that can be included are (5.1), (5.11), (5.15) and (5.18).

5.6.1 Sequentially decentralized MPC

When sequentially decentralized MPC is used, each unicycle has its own cost function. When all control objectives, that are discussed in this chapter, need to be fulfilled, the total cost function $J_{u_i}^{T_o}(kT_s)$

becomes

$$J_{u_i}^{T_o}(kT_s) = J_{u_i}^{R_i}(kT_s) + \sum_{j=1}^{N_o} J_{u_i}^{O_j}(kT_s) + \sum_{j=1}^{N_u} J_{u_i}^{U_j}(kT_s) + \sum_{j=1}^{N_u} J_{u_i}^{F_j}(kT_s). \tag{5.20}$$

Here, N_o indicated the number of circular obstacles and N_u is the number of unicycles that are involved. These cost functions are scalar expressions that depend on the input vectors $I_i(kT_s)$.

5.6.2 Centralized MPC

When centralized MPC is used, the control objectives of all unicycles are combined into a single cost function $J^{T_o}(kT_s)$, which is given by

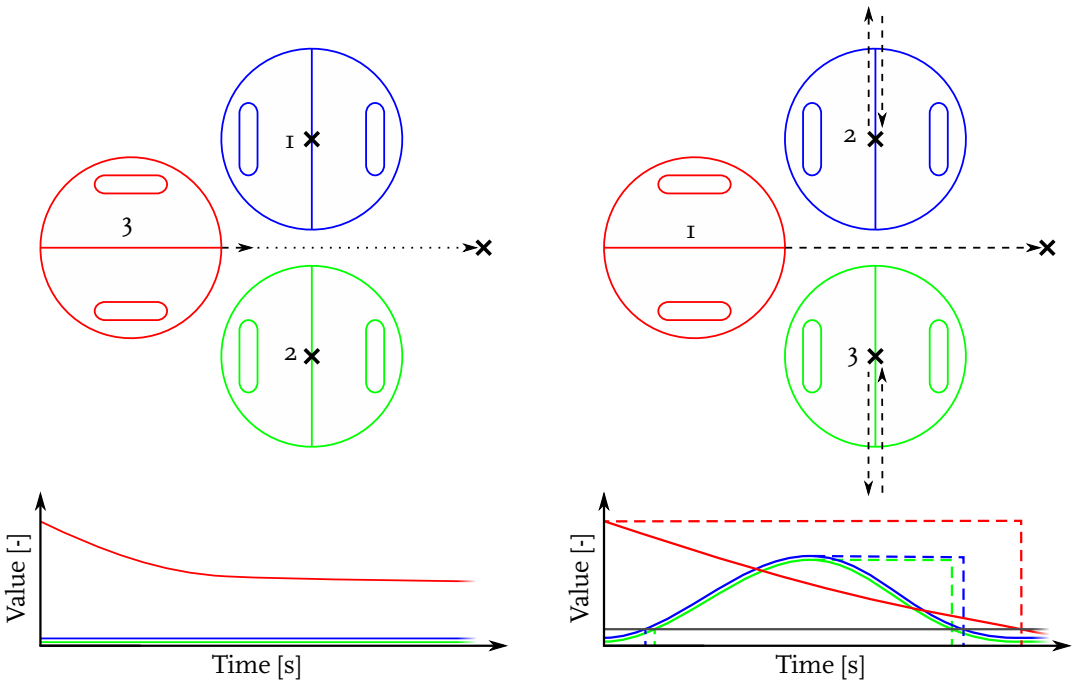
$$J^{T_o}(kT_s) = \sum_{i=1}^{N_u} \left(J_{u_i}^{R_i}(kT_s) + \sum_{j=1}^{N_o} J_{u_i}^{O_j}(kT_s) + \sum_{j=1}^{N_u} J_{u_i}^{U_j}(kT_s) + \sum_{j=1}^{N_u} J_{u_i}^{F_j}(kT_s) \right). \tag{5.21}$$

This cost function depends on a combination of all input vectors, given by

$$I(kT_s) = \begin{bmatrix} I_1^T(kT_s) & I_2^T(kT_s) & \dots & I_{N_u}^T(kT_s) \end{bmatrix}^T.$$

5.7 Priorities based on cost function value

When sequentially decentralized MPC is used, it is necessary to determine the priority values $P_{u_i}(kT_s)$ of the unicycles to determine the order in which the optimization takes place. This order is determined by arranging the priority values from high to low. If two unicycles have the same priority value, the unicycle with a lower priority can plan first. A simple method is to use fixed priorities, which are determined in advance. While this will not cause any issues when unicycles are operating at large distances from each other, this is not always the case when unicycles are operating in close proximity to each other. For example, consider the following scenario, depicted in Figure 5.8, where three unicycles are shown using two different priority strategies.



(a) Conflict resolution with fixed priorities.

(b) Conflict resolution with dynamic priorities.

Figure 5.8: Resolution of a conflict situation with different priority strategies.

The goal of the three unicycles is to drive to their desired locations, which are indicated by the crosses. The blue and green unicycles start from their desired locations, while the red one does not. When a

fixed order is used, depicted by the number on the unicycles in Figure 5.8(a), the blue and green unicycles plan their paths first. As they are already in position, they will remain there. This means that the red unicycle is unable to plan a direct path to its desired location, and will drive as close as it can to the blue and green unicycles, indicated with the short dashed line. It is possible that the unicycle remains in that position, indicated with the dotted line, as local optimization is used in this thesis, which is further discussed in the next chapter.

In the bottom half of Figure 5.8(a), the cost function values of the unicycles from (5.20) are indicated. As can be seen, the values of the blue and green unicycles remain close to zero. The cost function value of the red unicycle initially decreases, but then remains constant.

In Figure 5.8(b), a different strategy is used. Now, the priorities are determined by the cost function values of the unicycles. The idea is that a unicycle that is far from accomplishing its control objectives is more important than a unicycle that has already fulfilled its control objectives. These control objectives do not include avoiding other unicycles. Why this is done is explained below. The priority value of a unicycle can be determined as

$$P_{u_i}(kT_s) = J_{u_i}^{R_i}(kT_s) + \sum_{j=1}^{N_o} J_{u_i}^{O_j}(kT_s) + \sum_{j=1}^{N_u} J_{u_i}^{F_j}(kT_s). \tag{5.22}$$

When these varying priorities are used to determine the order in which the unicycles plan their paths, the red unicycle will be able to plan a path towards its goal as it initially has a higher priority. However, it "pushes" the other unicycles from their desired location, which causes the priority of the these unicycles to rise above the priority of the red one. This leads to priorities that switch back and forth. To resolve this issue, priorities are only allowed to increase or stay the same as they were at the previous time instant. Only if a priority level drops below the base priority level, which is set at a fraction (e.g. 1%) of the highest priority value that has ever occurred with all unicycles, the priority value becomes zero. This base priority level indicates that a unicycle has fulfilled its control objectives. When this strategy is applied, the unicycles will be able to resolve this situation. The red unicycle will plan a path as if the other unicycles are not there, and try to drive to its desired location. The blue and green unicycles will move away, as they see that the planned path of the red unicycle will result in a costly collision. This is why avoiding other unicycles is not included in (5.22), as it would also result in priorities switching back and forth. Finally, after the red unicycle has reached its destination, the blue and green unicycles will return to their desired positions.

The lower part of Figure 5.8(b) shows what happens with the priorities levels of the unicycles. On the vertical axis, the priority values from (5.22) are shown. The base priority level is shown in grey, the current priority value is indicated by the colored lines, and the actual used priorities are indicated by the colored dashed lines. Initially, the blue and green unicycles have a priority of zero as they are below the base priority, while the red one is not. It can therefore plan a path to its desired destination, "pushing" the other unicycles out of the way, resulting in a decreasing priority value. This causes the priority values of the blue and green unicycles to rise above the base priority, and eventually to rise above the current priority of the red unicycle, but not the actually used priority. Eventually, the priorities of all three unicycles drop below the base level. The use of this priority strategy allows conflict situations such as these, or with more unicycles, to be resolved. In the simulation and experimental results, this strategy is always used in combination with sequentially decentralized MPC.

5.8 Conclusions

In this chapter, the cost function is derived, which covers the tracking of a reference trajectory, avoiding collisions with circular obstacles and other unicycles, and driving in a formation. For sequentially decentralized MPC, a priority strategy is developed which can be used to resolve conflicts between unicycles. This strategy is not necessary when centralized MPC is used, as it considers the control objectives of all unicycles simultaneously.

The two control strategies that are discussed in this thesis vary significantly. Whereas sequentially decentralized MPC requires more complex preparations, such as determining priorities, before the simpler optimization takes place, centralized MPC shifts that complexity to the optimization problem.

The length of the input vectors show this clearly, as sequentially decentralized MPC uses two times the number of elements in N_c^i as free input variables. Centralized MPC multiplies this number of variables by the number of unicycles, creating an optimization problem with more freedom, but more complexity as well. A choice for the type of MPC should therefore be based on the quality and speed of the optimization algorithm, which is further discussed in Chapters 6 and 7.

Chapter 6

Optimization of Cost Function

In this chapter, it is discussed how an optimal predicted input can be obtained using the cost function from Chapter 5. To that end, it is first discussed how the gradient and the Hessian of the cost function can be obtained and used in two different optimization methods: the steepest descent and Newton's optimization method. After that, the two methods are augmented with a line search method to obtain a more robust optimization algorithm. Finally, the two augmented optimization methods are compared and some conclusions are drawn.

6.1 Introduction of optimization problem

MPC solves an optimization problem at every time instant to obtain the optimal control inputs that are then sent to the unicycles. The goal of the optimization problem is to find a predicted input sequence that minimizes the cost function. Depending on the control method that is used, the optimization problem can be expressed as

$$\begin{aligned} &\min_{I^f_{(i)}(kT_s)} J^T_{(u_i)}\left(I^f_{(i)}(kT_s)\right), \\ &\text{subject to} \\ &v^{\min}_{u_i} \leq v_{u_i} \leq v^{\max}_{u_i}, \\ &\omega^{\min}_{u_i} \leq \omega_{u_i} \leq \omega^{\max}_{u_i}. \end{aligned} \tag{6.1}$$

Here, the terms between brackets in subscripts ((i) and (u_i)), are used to indicate that this both applies to sequentially decentralized, and centralized MPC. When sequentially decentralized MPC is used, the terms between the brackets are kept. When centralized MPC is used, the terms are ignored. The variable that needs to be optimized, $I^f_{(i)}(kT_s)$, is the free input vector which contains the inputs that are specified at the control instants N^i_c. For example, when $N^i_c = \begin{bmatrix}1 & 3\end{bmatrix}$, the free input vector becomes

Sequentially Decentralized MPC

$$I^f_i(kT_s) = \begin{bmatrix} v_{u_i}((k+1)T_s) \\ v_{u_i}((k+3)T_s) \\ \omega_{u_i}((k+1)T_s) \\ \omega_{u_i}((k+3)T_s) \end{bmatrix}.$$

Centralized MPC

$$I^f(kT_s) = \begin{bmatrix} \begin{bmatrix} v_{u_1}((k+1)T_s) \\ v_{u_1}((k+3)T_s) \\ \omega_{u_1}((k+1)T_s) \\ \omega_{u_1}((k+3)T_s) \end{bmatrix}^T, \ldots, \begin{bmatrix} v_{u_{N_u}}((k+1)T_s) \\ v_{u_{N_u}}((k+3)T_s) \\ \omega_{u_{N_u}}((k+1)T_s) \\ \omega_{u_{N_u}}((k+3)T_s) \end{bmatrix}^T \end{bmatrix}^T.$$

The cost function depends on more variables than $I^f_{(i)}(kT_s)$, but they are not shown in (6.1) for brevity. The constraints in this optimization problem are bounds on the inputs of the unicycles.

It is required that (6.1) is solved at every time instant. When sequentially decentralized MPC is used, this problem should be solved in a specified order that is determined by the priorities of the unicycles. This is required to be able to determine $n(kT_s)$ and $o(kT_s)$ from (5.16) and (5.19). A method to determine the priorities of the unicycles is discussed in Section 5.7. When centralized MPC is used, (6.1) only has to be solved once, as it plans the path of all unicycles simultaneously.

6.2 Local exploration

It is desired that the lowest possible value of the cost function is obtained. This function is nonlinear and there are constraints on the inputs of the unicycles, making it impossible to find an explicit solution for an optimal predicted input sequence. To overcome this problem, it is possible to search for an optimal sequence using local exploration.

When optimality conditions cannot be manipulated to yield an explicit solution, an iterative *procedure can be sought. One may start from an initial point where the function value is calculated and then take a step in a downward direction. where the function value will be lower. To make such a step, one utilizes local information and explores the immediate vicinity of the current point; hence, all iterative methods perform some kind of* local exploration [45].

To obtain the lowest possible cost function value, local exploration is used in this thesis in the iterative search for a local optimum. Two local exploration methods are discussed here: the steepest descent method and Newton's method. These methods make use of local first and second order approximations of the cost function, the gradient and the Hessian respectively.

6.3 Obtaining gradient and Hessian

To obtain the components that make up the gradient, the cost function can be differentiated with respect to its free input variables. The Hessian can be obtained by differentiating the gradient to the free input variables. As the cost function is constructed symbolically, it is possible to differentiate it to the different free input variables. The gradient of the cost function can be obtained as

$$g_{(i)}\left(I^{f}_{(i)}(kT_s)\right) = \frac{\partial J^{T_o}_{(u_i)}\left(I^{f}_{(i)}(kT_s)\right)}{\partial I^{f}_{(i)}(kT_s)}. \tag{6.2}$$

The Hessian can then be calculated by differentiating (6.2) to obtain

$$H_{(i)}\left(I^{f}_{(i)}(kT_s)\right) = \frac{\partial g^{T}_{(i)}\left(I^{f}_{(i)}(kT_s)\right)}{\partial I^{f}_{(i)}(kT_s)}. \tag{6.3}$$

During the differentiation, the penalty and collision terms are treated as constants. Also, the atan2 function that is used in (5.17), is changed to a basic atan function when the gradient and Hessian are derived. After the differentiation, the atan term is changed back to atan2. Because the derivatives of these two functions are identical, and the derivative of the atan function does not contain any atan terms, the derivatives of the cost function remain exact.

6.4 Iterative local optimization

Starting from an initial predicted input sequence $I^{f_0}_{(i)}(kT_s)$, the first problem is to determine a search direction. Depending on the optimization method that is used, the steepest descent method or Newton's method, first and second order approximations of the cost function are used to determine this direction respectively. These local approximations are used to determine iterative procedures to obtain lower cost function values.

6.4.1 Steepest descent optimization method

The steepest descent optimization method uses a first order approximation of the cost function to determine a lower cost function value. This approximation is given by

$$J^{T_o}_{(u_i)}\left(I^{f_{p+1}}_{(i)}(kT_s)\right) = J^{T_o}_{(u_i)}\left(I^{f_p}_{(i)}(kT_s)\right) + g^{T}_{(i)}\left(I^{f_p}_{(i)}(kT_s)\right)\partial I^{f_p}_{(i)}(kT_s),$$

Here, p indicates the current iteration, and the change of inputs is defined as

$$\partial I^{f_p}_{(i)}(kT_s) = I^{f_{p+1}}_{(i)}(kT_s) - I^{f_p}_{(i)}(kT_s). \tag{6.4}$$

A search direction that results in a lower cost function values can be found in the negative direction of the gradient

$$\partial I_{(i)}^{f_p}(kT_s) = -g_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right). \tag{6.5}$$

This allows an iterative procedure to be created by combining (6.4) and (6.5) to get

$$I_{(i)}^{f_{p+1}}(kT_s) = I_{(i)}^{f_p}(kT_s) - g_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right). \tag{6.6}$$

6.4.2 Newton's optimization method

It is also possible to use a second order approximation of the cost function, which is used by Newton's optimization method. This second order approximation is given by

$$\begin{aligned} J_{(u_i)}^{T_o}\left(I_{(i)}^{f_{p+1}}(kT_s)\right) =& J_{(u_i)}^{T_o}\left(I_{(i)}^{f_p}(kT_s)\right) + g_{(i)}^{T}\left(I_{(i)}^{f_p}(kT_s)\right)\partial I_{(i)}^{f_p}(kT_s) + \dots \\ & \frac{1}{2}\partial {I_{(i)}^{f_p}}^{T}(kT_s) H_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right)\partial I_{(i)}^{f_p}(kT_s). \end{aligned}$$

The minimum value of $J_{(u_i)}^{T_o}\left(I_{(i)}^{f_{p+1}}(kT_s)\right)$ is found at the stationary condition

$$g_{(i)}^{T}\left(I_{(i)}^{f_p}(kT_s)\right) + H_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right)\partial I_{(i)}^{f_p}(kT_s) = \mathbf{0}. \tag{6.7}$$

Using (6.4) and (6.7), an iterative procedure can be created, assuming that $H_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right)$ is invertible, resulting in

$$I_{(i)}^{f_{p+1}}(kT_s) = I_{(i)}^{f_p}(kT_s) - H_{(i)}^{-1}\left(I_{(i)}^{f_p}(kT_s)\right) g_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right). \tag{6.8}$$

6.4.3 Advantages and disadvantages

Both these iterative optimization methods have their advantages and disadvantages. The steepest descent method has the advantage that only the gradient of the cost function needs to be available. Newton's method uses the Hessian, which requires more work to derive. Another disadvantage of Newton's method is the Hessian is required to be positive definite to guarantee that an iteration results in a lower function value. If this is not the case, Newton's method may lead to higher function values. It is possible to compensate for this, but this compensation has not been investigated further in this thesis.

A disadvantage of the steepest descent method is that when the input vector approaches a local optimum where input constraints are not active, progress towards the optimum becomes slow, because the gradient decreases in size. This problem can be circumvented by using Newton's method. This is because this method makes use of second order curvature information, leading to faster convergence to the optimum if the input vector is close to a local optimum.

When an input vector is far from a local optimum, it can occur with both methods that a much too large step in the search direction is taken, leading to a higher cost function value. To resolve this problem, the two methods are used in combination with a line search.

6.5 Line search optimization

Line search optimization is a method that searches for a minimum in a descent direction that is determined in advance, and gives more control over the step size that is taken. Another advantage is that the gradient and Hessian don't have to be evaluated at every iteration, leading to shorter computation times. From (6.6) and (6.8), it is possible to determine two descent directions $s_{(i)}^{f_p}(kT_s)$ as

$$s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right) = -g_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right), \tag{6.9}$$

when the steepest descent optimization method is used, and

$$s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right) = -H_{(i)}^{-1}\left(I_{(i)}^{f_p}(kT_s)\right) g_{(i)}\left(I_{(i)}^{f_p}(kT_s)\right), \tag{6.10}$$

when Newton's optimization method is used. When a step size length α^{f_p} is introduced, a new iterative optimization procedure can be created as

$$I_{(i)}^{f_{p+1}}(kT_s) = I_{(i)}^{f_p}(kT_s) + \alpha^{f_p} s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right). \tag{6.11}$$

Now that the search direction is determined, the length of the step size α^{f_p} that will be taken in this direction needs to be determined. The cost function can have a wide range of values, which means that values in the gradient and Hessian can also vary strongly. This particularly happens when unicycles plan a path that approaches other unicycles and obstacles. It can occur that a small shift of the path results in a collision, which can cause a strong variation in the gradient and the Hessian. This can lead to a large shift in the inputs while it would be more optimal to shift the inputs only a little. To resolve this issue, the direction vector is scaled so that the largest size of a component is equal to one, so

$$s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right) = \frac{s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right)}{||s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right)||_\infty}. \tag{6.12}$$

If the search direction vector only contains zeros, this scaling is not applied. The length of the largest step is now fully determined by the step size α^{f_p}. By keeping the initial step size small (e.g. 10% of the range of an input), large changes in inputs can no longer occur in one iteration.

To iteratively obtain a lower cost function value using the search direction, the following steps can be used

1. Determine an initial input vector $I_{(i)}^{f_0}(kT_s)$, the initial step size α^{f_p}, the minimum allowed step size $\alpha_{\min}^{f_p}$, and the maximum number of function evaluations $N_{\max}^p$.
2. Compute the search direction $s_{(i)}^{f_p}\left(I_{(i)}^{f_p}(kT_s)\right)$ according to (6.9) or (6.10), and scale the search direction according to (6.12).
3. Keep taking steps of size α^{f_p} in the search direction as in (6.11) and increasing the iteration index p, until $J_{(u_i)}^T\left(I_{(i)}^{f_p}(kT_s)\right) > J_{(u_i)}^T\left(I_{(i)}^{f_{p-1}}(kT_s)\right)$.
4. Use $I_{(i)}^{f_{p-1}}(kT_s)$ as the result of the line search of step 3, and if the first step taken with a new α^{f_p} already results in a larger cost function value, reduce the step size α^{f_p}.
5. If $\alpha^{f_p} \geq \alpha_{\min}^{f_p}$, and if $p \leq N_{\max}^p$, go to step 2.

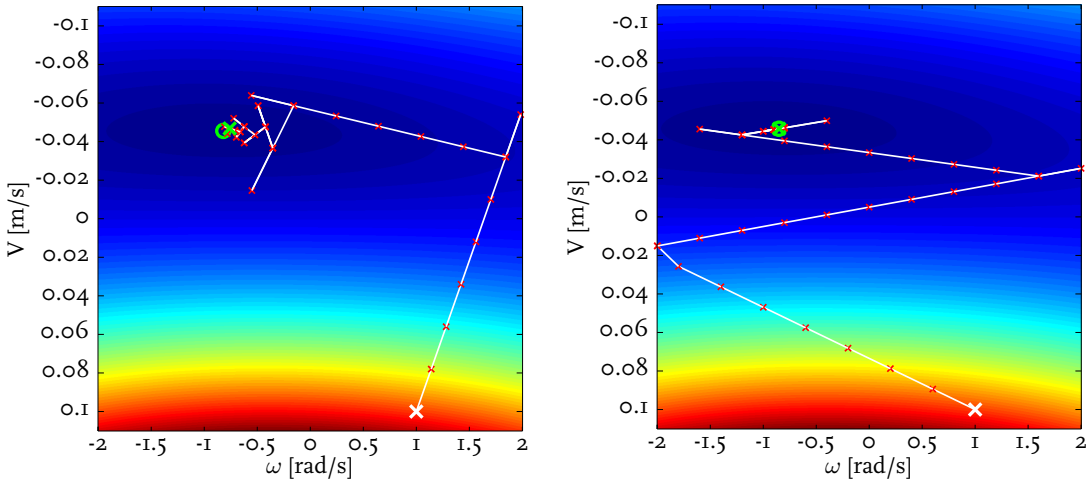
When (6.11) in step 3 results in an input vector that exceeds the constraint on the inputs, the input vector $I_{(i)}^{f_p}(kT_s)$ is bounded and used in step 2. After the search direction is recalculated, it is checked if the direction of the input(s) that are on their bounds can lead to another constraint violation. If this is so, the corresponding part of the search direction is set to zero.

This iterative scheme can result in a predicted input vector that results in a lower cost function value than the initial predicted input vector. It can occur that the optimization results in a path of a unicycle that will collide with an obstacle, or another unicycle. If this occurs, and if $0 \in \left[v_{u_i}^{\min}, v_{u_i}^{\max}\right]$, the unicycle is stopped at the next future time instant. In this way, collisions can be avoided. After that, a next optimization might result in a collision free path.

6.6 Comparison of steepest decent and Newton's method

To illustrate the difference between the two optimization methods that have been implemented, the following examples are presented. In these examples, $N_p = 5$ and $N_c^i = 1$, which facilitates visualization of the optimization process. As the control horizon is equal to one, there are two free control inputs that are kept constant over the prediction horizon. This means that it is possible to show all possible combinations of v_{u_i} and ω_{u_i} with their corresponding cost function values, which is shown in the background in Figures 6.1 and 6.2.

The red areas indicate high cost function values, while the blue areas indicate low values. The result of the optimization process are drawn over the cost function values. The optimization starts at the white cross. From there, red crosses indicate a cost function evaluation at that point. The white lines indicate the search direction. Finally, the most optimal input is indicated with a green circle. The result of the optimization is the green cross. In Figure 6.1, an optimization problem is depicted of a unicycle that has to track a reference trajectory. There are no obstacles or other unicycles present, so the cost function values form a smooth surface.

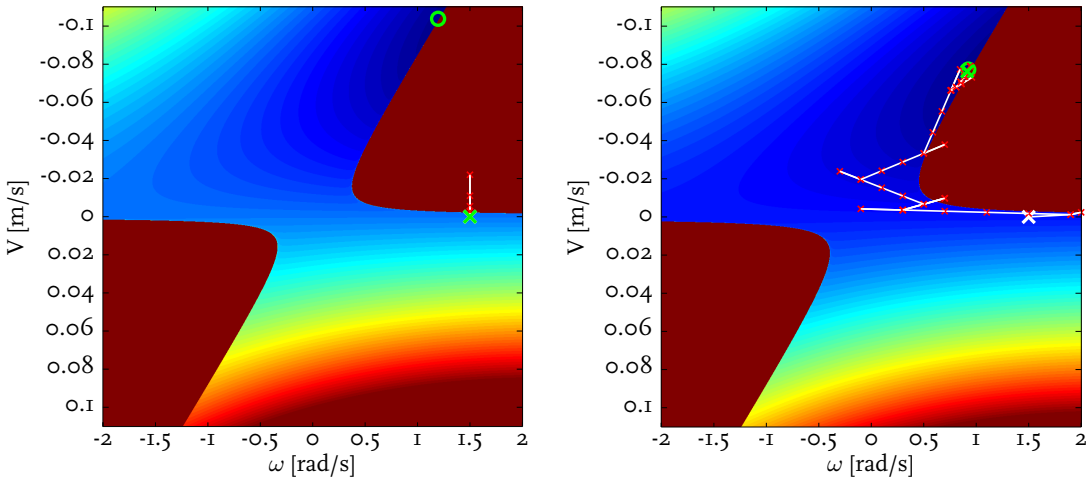


(a) Optimization using the steepest descent method.

(b) Optimization using Newton's method.

Figure 6.1: Unicycle tracking a reference trajectory.

In Figure 6.1(a), the steepest descent method is used to determine an optimal predicted input sequence. As it can be seen, the progress towards the optimum tends to follow a characteristic zig-zagging pattern, which is especially clear when the optimum is approached. Figure 6.1(b) shows the different behavior from Newton's method. As it can be seen, the initial search direction is different than the initial search direction of the steepest descent method. This is caused by the use of second order curvature information. When the optimum is approached, using Newton's method results in a much faster convergence to the optimum.



(a) Optimization using the steepest descent method.

(b) Optimization using Newton's method.

Figure 6.2: Unicycle avoiding a circular obstacle while tracking a reference trajectory.

Another optimization problem is shown in Figure 6.2, which shows the cost function values of a unicycle that again has to track a reference trajectory, but now a circular obstacle is in the way. The cost function values show a distinct pattern here. Within the dark red zones, a given combination of the

inputs will result in a future collision. The cost function values do vary in this area but this is not shown in this figure, as it would reduce the contrast in the areas where no collision will occur. Figure 6.2(a) shows how the steepest descent method deals with this situation. As it can be seen, this method has problems with getting to the desired optimum. This is caused by the fact that the cost function values vary strongly near the edges of the red areas. This results in gradients that are almost perpendicular to the edges, which explains why the combination of inputs that is found by this method is far from the desired optimum. Newton's method, depicted in Figure 6.2(b), performs better in this case, even though it initially moves towards the wrong direction. Finally, it is able to search along the boundary between collision and no collision, resulting in obtaining the most optimal input combination in this situation.

6.7 Conclusions

These two examples show some of the strengths and weaknesses of the steepest descent, and Newton's optimization method. In the first example, both methods are able to find the optimum but they do so in different ways. In this case, using Newton's method is preferred, because the approach of the optimum is more direct than the steepest descent method. The second example also shows that Newton's method outperforms the steepest descent method. Even though these two examples seem to show that Newton's method performs better than the steepest descent method, this is not always the case.

When the control horizon becomes longer than one, Newton's method sometimes has trouble with finding an optimum. This is caused by the fact that the Hessian is not always positive definite. As mentioned earlier, it is possible to compensate for this, but this was not investigated in this thesis. The steepest descent method does not suffer from this drawback, as the gradient always points to a lower cost function value. Even though this method has trouble following a boundary such as the one shown in Figure 6.2, its results prove to be more robust. Therefore, the steepest descent method is the preferred method when a longer control horizon is used.

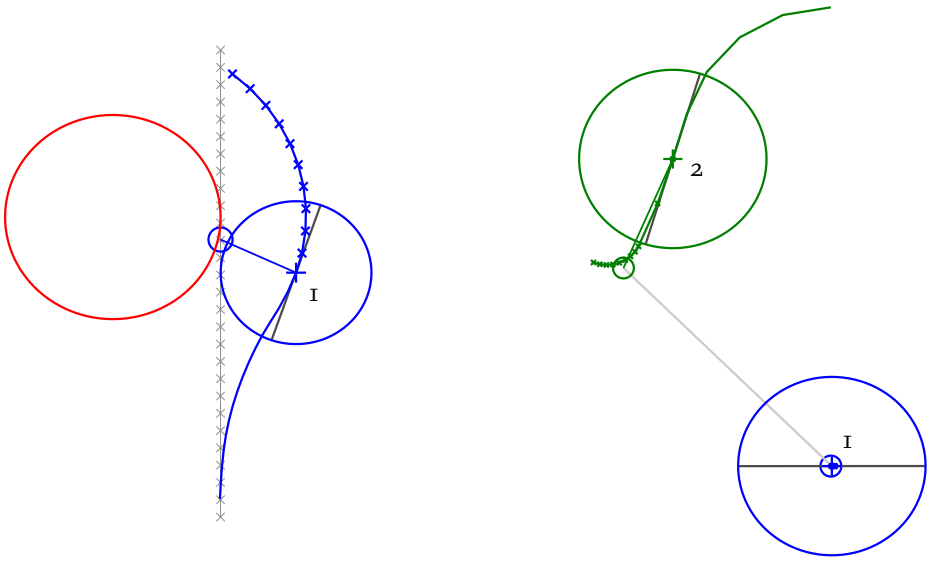
Chapter 7

Simulation Results

In this chapter, simulation results of the controller that was constructed in Chapters 4 to 6 are discussed. The control objectives that are included in the cost function are first demonstrated individually. After that, some additional simulations are performed that showcase various other properties of the controller. Finally, some conclusions are drawn about the simulation performance of the controller.

7.1 Introduction

To determine whether the control objectives that are included in the cost function, can be fulfilled by the designed controller, a number of simulations are performed. The control objectives that are tested here are tracking a reference trajectory, avoiding collisions with circular obstacles and other unicycles, and driving in a formation. The results of these simulations are given in this chapter. Because the simulations often involve multiple moving objects, it is difficult to visualize the results of the simulation in a few images. Therefore, videos of all simulations that are discussed here are made available online. In this chapter, the resulting paths of the unicycles during the simulation are depicted. All simulation videos can be found at www.youtube.com/er47ik and clicking on 'Simulations'. Two examples of snapshots of simulation videos are depicted in Figure 7.1.



(a) Unicycle tracking a reference trajectory.

(b) Two unicycles driving in formation.

Figure 7.1: Snapshots of simulation videos that are available online.

In these two figures, the unicycles are depicted as circles, along with their corresponding numbers. The dark grey lines indicate the current orientations of the corresponding unicycles. The smooth line that is drawn behind the unicycle indicates the path it has taken. The line that is marked with crosses

indicates the result of the optimization algorithm at the current time instant. This is the most optimal path that is found by the optimization algorithm at that time instant. The control objectives of the unicycles are indicated with the small circles that have the same color as the unicycle. It is desired that the center of the unicycle, indicated with the plus sign, is in the center of this small circle. To further clarify this, a thin line is drawn between these points. Figure 7.1(a) shows a unicycle that is tracking a reference trajectory. This trajectory is indicated with the small grey crosses. As a virtual circular red obstacle is preventing the unicycle from driving on the desired trajectory, the unicycle is driving around it. In Figure 7.1(b), two unicycles are depicted that need to form a formation. It is desired that unicycle 2 stays at a certain distance and angle from unicycle 1, which is indicated with the small green circle. The shape of the desired formation is indicated with the light grey line that connects the unicycles.

In the simulations presented in this chapter, parameters of the unicycles are adopted that are similar to system parameters of the e-pucks. In this way, simulation results are achieved that are closer to experimental measurements. These parameters can be found in Table 7.1.

Parameter	r [m]	$v^{\min}$ [m/s]	$v^{\max}$ [m/s]	$\omega^{\min}$ [rad/s]	$\omega^{\max}$ [rad/s]
Value	0.035	-0.11	0.11	-2	2

Table 7.1: System parameters of the e-puck.

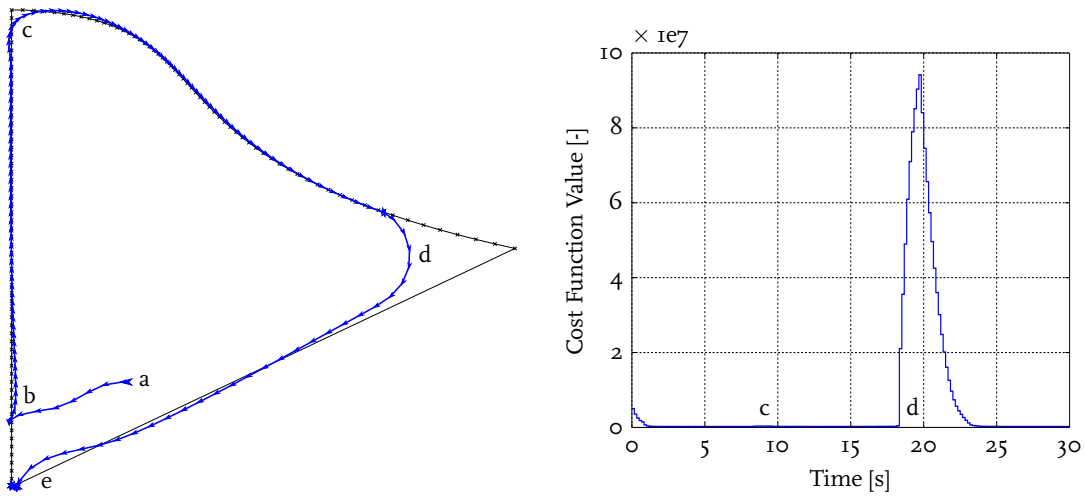
7.2 Tracking a reference trajectory

A reference trajectory has been created to test the tracking performance of the controller. This trajectory consists of a straight line segment, followed by a 90° degree turn into a curved trajectory. At the end of the curved segment, the reference trajectory jumps back to the start. With this trajectory, it can be tested how the controller deals with trajectories that the unicycle can and can not follow due to constraints on its inputs. For this simulation, the parameters from Table 7.2 are used.

Parameter	N_p [-]	N_c^i [-]	T_s [s]	P_u^r [-]	Q_u^r [-]
Value	10	$\begin{bmatrix}1 & 2\end{bmatrix}$	1/6	1e8	1e4

Table 7.2: Reference tracking simulation parameters.

The sampling period is chosen relatively long, when compared to traditional control methods. Here, it is chosen in combination with the prediction horizon to obtain a long enough physical prediction length. A shorter sampling time can be chosen to obtain smoother input signal, but this decreases the physical prediction length. The results of this simulation can be seen in Figure 7.2.



(a) Top-down reference and actual trajectories.

(b) Cost function values.

Figure 7.2: Top-down view and cost function values of the reference tracking simulation.

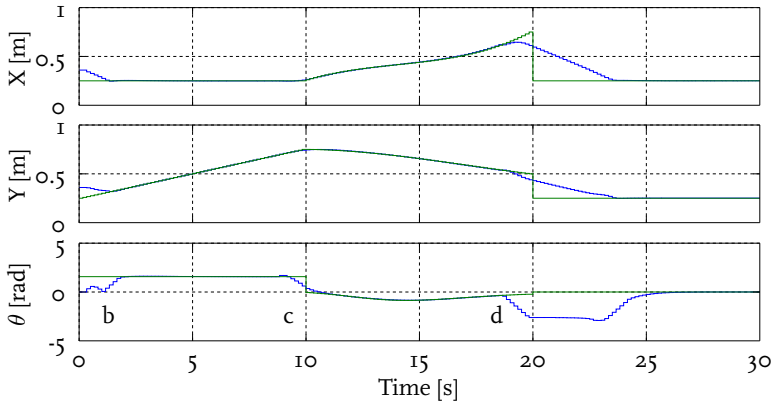
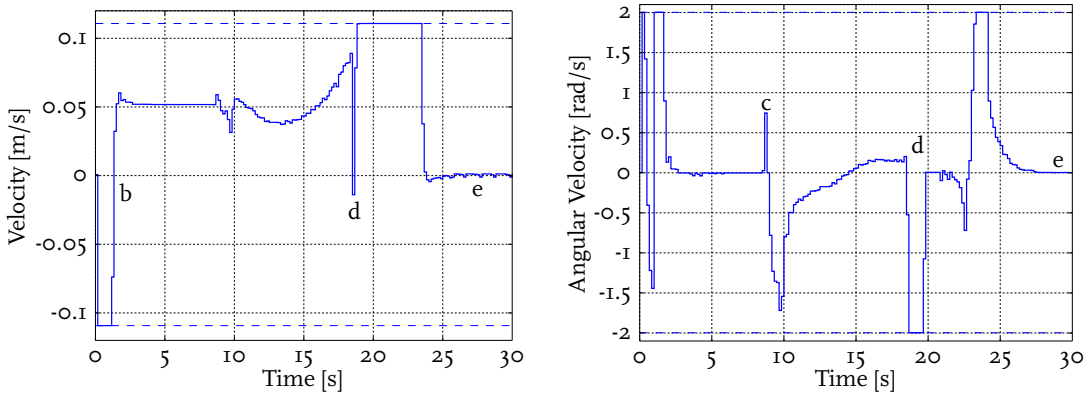


Figure 7.3: Separate state references and actual signals.

The reference and actual trajectories are depicted in Figure 7.2(a). The cost function values are shown in Figure 7.2(b). The separate state references and actual motions in x-, y-, and θ-direction are depicted in Figure 7.3. The green lines indicate the references and the blue lines indicate the actual values. The velocity and angular velocity of the unicycle are depicted in Figure 7.4. Here, the dashed lines indicate the boundaries on the inputs, which are specified in Table 7.1. These figures have letters in them, which connect events in the different figures. A video of the simulation can be found on YouTube.



(a) Velocity during the simulation.

(b) Angular velocity during the simulation.

Figure 7.4: Velocities of the unicycle during the reference tracking simulation.

At the start of the simulation (a), a unicycle starts at a distance from the reference trajectory. This is indicated with the large arrow, which indicates the initial movement direction of the unicycle. It then drives towards the reference trajectory, switches driving direction (b), and continues following the reference trajectory. The unicycle switches direction, because the angular error is penalized. If this would not be the case, it would drive in a different way. The unicycle continues along the reference trajectory until the tip of the prediction horizon reaches the 90° turn (c). The optimization algorithm then determines that it is more optimal to first drive slightly to the left, before turning right into the corner. The curved part of the reference trajectory is then followed, until the reference trajectory returns to the starting position at (e). Because of the prediction horizon of 10, the last part of the smooth section is not tracked anymore. If this behavior is undesired, it is possible to decrease the prediction horizon, decrease the sampling time, or change the weighing factors of future sampling instants. At (d), the unicycle starts turning at the maximum angular velocity. During one sample, it drives backwards, before driving back to (e) at maximum velocity. Finally, after 24 seconds, the unicycle is in position, and the unicycle rotates to get in the desired orientation.

As it can be seen in Figures 7.4(a) and 7.4(b), it takes one sample for the velocities to become nonzero. This is caused by the delay that is introduced in Chapter 3. During the course of the simulation, both inputs reach their input constraints. This causes no problems, as the unicycle is able to follow the reference trajectory as well as is allowed by the bounds on the inputs.

At the end of the simulation, when the unicycle is in position, some small variations can still be seen in the velocity inputs signal (e). This is caused by the fact that the unicycle is not exactly at its desired orientation, meaning that the gradient at its position is nonzero. This results in the fact that the optimization algorithm never stops optimizing and the inputs will probably never become zero, but remain small. How small is determined by the minimum step size that is set in the optimization algorithm. It is also possible that the optimization never stops due to the convexity of the optimization problem. Usually an optimum can be found, but it can not be guaranteed that this is a global optimum.

From this simulation, it can be determined that the unicycle is able to deal with different reference trajectories. It can follow trajectories that allow the unicycles to stay within their input limitations. However, it is no problem if this is not the case, as displayed by the behavior in the 90° turn. It is also possible to specify a location that the unicycle should drive to. The behavior of the unicycle that is observed in this simulation is similar to the behavior that is seen in other simulations. Therefore, the results of the following simulations are not further elaborated.

7.3 Avoiding collisions with circular obstacles

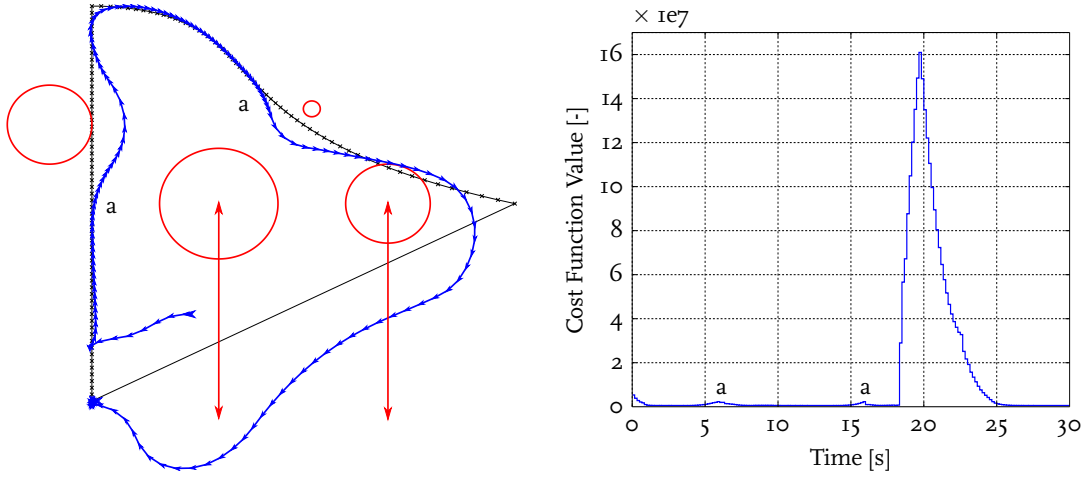
To determine how the controller deals with circular obstacles, the same reference trajectory is used as in the previous section, but four circular obstacles are now included in the same workspace. The parameters that are used in this simulation are the same as in the previously discussed one. The parameters that are related to obstacle avoidance are given in Table 7.3.

Parameter	$P_u^o(mT_s)$ [-] $(m = k+1)$	$P_u^o(mT_s)$ [-] $(m > k+1)$
Value	1e14	1e12

Table 7.3: Obstacle collision avoidance parameters.

As it can be seen, the parameters that penalize the collision with an obstacle are several orders larger than the ones related to the tracking of a reference signal. With these parameters, a collision that will happen at the first controllable time instant is penalized more than a collision that will occur later. The parameters can also be reduced several orders without noticeable differences of the simulation results, but they should remain larger than the penalties on reference tracking and formation keeping.

The results of this simulations are shown in Figure 7.5. The reference and actual trajectories are depicted in Figure 7.5(a). The cost function values are shown in Figure 7.5(b). The separate state references and actual motions in x-, y-, and θ-direction of this simulation are depicted in Figure 7.6. A video of the simulation is available on YouTube.



(a) Top-down reference and actual trajectories.

(b) Cost function values.

Figure 7.5: Top-down view and cost function values of the obstacle collision avoidance simulation.

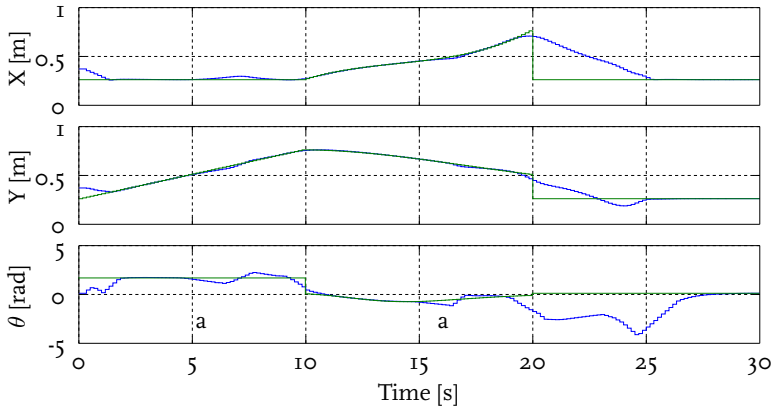


Figure 7.6: Separate state references and actual signals.

Of the four obstacles that are involved in the simulation, two are static, while two vary their radii and positions over time. The dynamic obstacles are indicated with the red arrows. As it can be seen in Figure 7.5(a) and (b), the obstacles that are present can prevent the unicycle from following its reference trajectory. However, the optimization algorithm is able to plan a path around the obstacles, even the dynamic ones. The positions and radii of the obstacles need to be known in advance for this to work. Another requirement is that the obstacles should not move too fast, so that the unicycles have enough space to get out of the way in time. By comparing Figure 7.5(a) with Figure 7.2(a), it can be seen where the unicycle is hindered by the obstacles.

From the resulting trajectory, it can be seen that the steepest descent optimization method is used. When the unicycle approaches the two static obstacles (a), it drives slower. This is because this algorithm has trouble in dealing with steep boundaries in the cost function values. In Figure 6.2(a) a similar illustrative situation is depicted. Here, the optimum can not be reached with the step sizes that are used. Only if step sizes would become very small, a path to the optimum could be found. However, this would lead to long computation times, so in this case a suboptimal solution is sufficient.

It is possible to combine multiple obstacles to form more complex shapes than just circles. It is no problem if obstacles overlap each other, however when multiple obstacles are combined, it can happen that unicycles become stuck behind them as only local optima can be found by the optimization algorithm. If it is desired that the unicycles stay within a certain operating area, obstacles can be used as well. By creating objects with a large radius and placing them at a distance from the desired area, approximately straight lines can be created that can be used to block off certain operating areas.

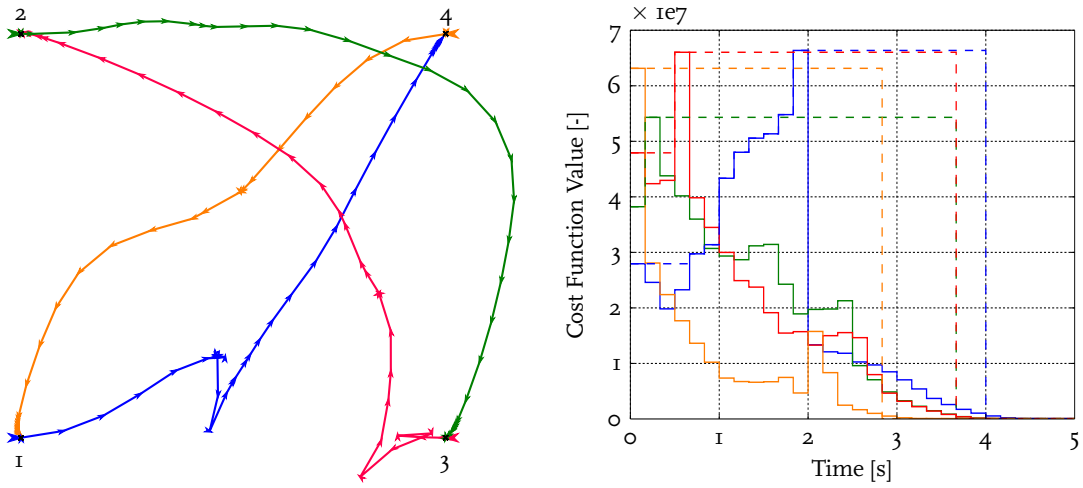
7.4 Avoiding collisions with other unicycles

To test the inter-unicycle collision avoidance, and the priority strategy of Section 5.7, a different simulation is devised. The goal of this simulation is to let the paths of a number of unicycles cross in the same position to see how they all behave. For avoiding other unicycles, parameters from Table 7.4 are used.

Parameter	P_u^r [-]	Q_u^u [-]	$P_u^u(mT_s)$ [-] $(m = k+1)$	$P_u^u(mT_s)$ [-] $(m > k+1)$
Value	1e8	0	1e14	1e12

Table 7.4: Unicycle collision avoidance parameters.

The parameter that penalizes the angular difference is not used here. The parameters that penalize collisions between unicycles have the same value as the penalties that are used with the obstacles. In this simulation, four unicycles start in a square. Their goal is to cross diagonally, so each unicycle would plan a path through the middle if it could. The result of this simulation is shown in Figure 7.7. The trajectories of the unicycles are shown in Figure 7.7(a), and the cost function values and priorities are shown in Figure 7.7(b). A video of the simulation is available on YouTube.



(a) Trajectories of the unicycles.

(b) Cost function values (5.22), and priorities (dashed).

Figure 7.7: Top-down view and cost function values of the unicycles in the simulation of unicycle collision avoidance.

In Figure 7.7(a), the starting positions of the unicycles are indicated by the numbers. The paths that are taken by the unicycles are sometimes quite odd; for instance, unicycle 1 starts moving forward but then moves back. The same behavior can be seen in the path of the third unicycle. This behavior can be explained by the priorities of the unicycles that change over the course of the simulation. First, the modified cost function values of the unicycles from (5.22) are indicated by the solid lines in Figure 7.7(b). The priority values are indicated with the dashed lines. As it can be seen, the priority values can only increase, unless the modified cost function values drop below a threshold. In this simulation, this threshold is set at 1% of the largest modified cost function value that has occurred up to that time instant. The modified cost function values increase and decrease over time, resulting in changes in priorities. Because of these changes in priority, the conflicting situations are resolved quickly, which results in the unicycles reaching their desired positions in approximately four seconds.

In this simulation, the dynamic priorities from Section 5.7 are used. The same simulation is also performed using fixed priorities. In this simulation, the order of the optimization remains constant, so the first unicycle plans a direct path to its destination. Lower priority unicycles needs to avoid higher priority unicycles, meaning that some unicycles have to take detours to get to their destination. Due to the detours, it takes a second longer for the unicycles to drive to their desired locations. Therefore, the priority strategy is used in the formation driving simulation, and during simulations and experiments with more than one unicycle.

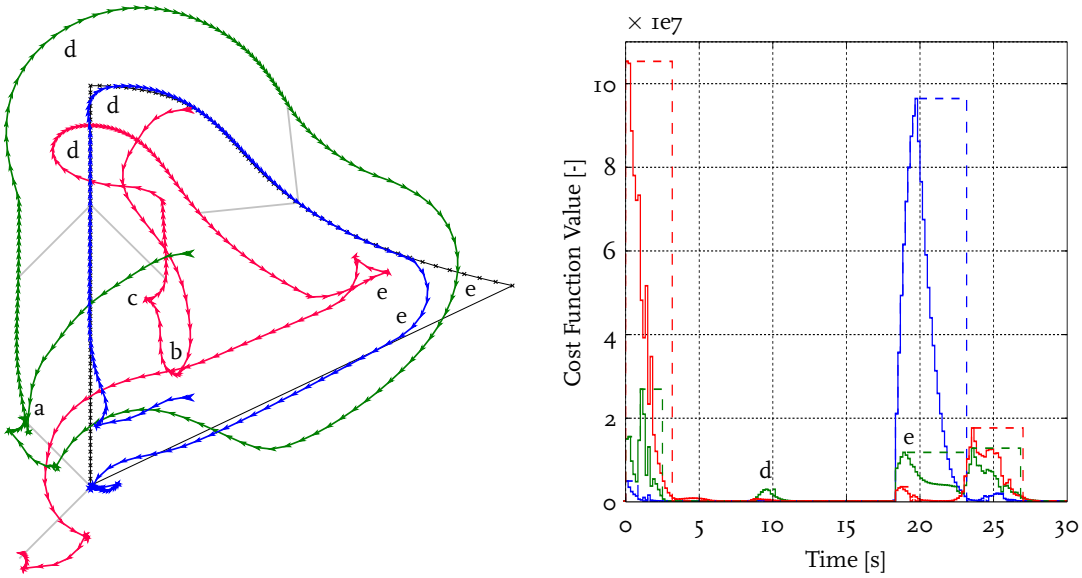
7.5 Driving in formation with other unicycles

To test the formation keeping ability of the controller, one unicycle follows the same reference trajectory as in Section 7.2. There are two other unicycles involved that have to drive behind this unicycle in a V-shaped formation. For this simulation, parameters from Table 7.5 are used. For collision avoidance, parameters from Table 7.4 are used.

Parameter	P_u^r [-]	Q_u^r [-]	Q_u^u [-]	R_u^u [-]	S_u^u [-]
Value	1e8	1e4	1e8	1e6	1e4

Table 7.5: Formation driving simulation parameters.

The penalty values for the formation are similar to the ones for tracking a reference trajectory, as the penalty on distance $P_u^r = Q_u^u$ and the penalty on the angle $Q_u^r = S_u^u$. An additional term R_u^u is added here, which ensures that the reference angles between the unicycles are maintained. These angles are $\pm 0.75\pi$ [rad] according to (5.17). The distance that should be kept between the unicycles is 0.12 [m]. The result of this simulation is shown in Figure 7.8.



(a) Reference and actual trajectories.

(b) Cost function values (5.22), and priorities (dashed).

Figure 7.8: Top-down view and cost function values of the formation driving simulation.

The trajectories of the unicycles are shown in Figure 7.8(a), and the modified cost function values and priorities are shown in Figure 7.8(b). A video of the simulation is available on YouTube.

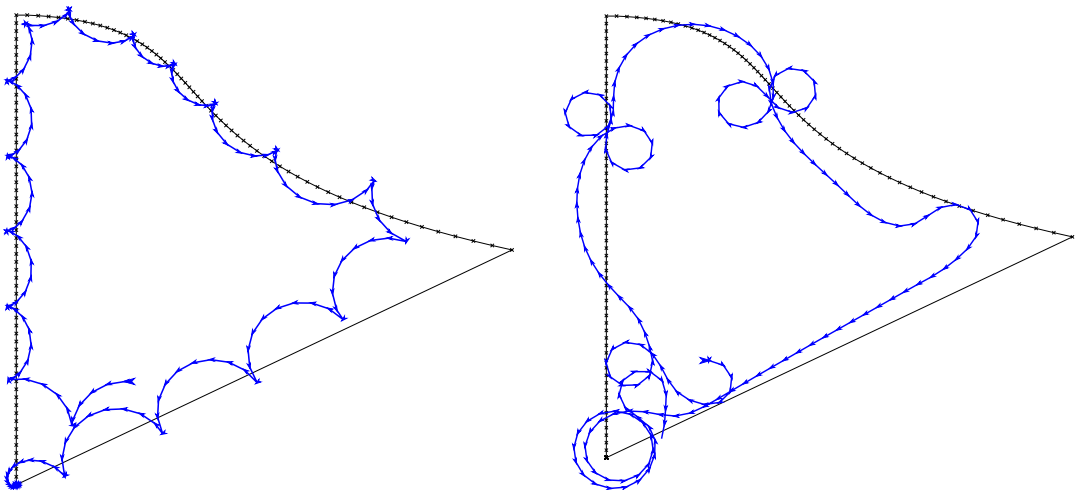
In the simulation, the unicycles start above each other. As the blue unicycle starts following its reference trajectory, the green and red ones get in formation behind the blue one. The green unicycle does so by switching driving direction (a), and the red one does so by making a sharp turn (b). Shortly after this turn, it rotates so that it is driving in the same direction as the blue unicycle (c). At this point, the unicycles are in the desired formation, which is indicated with the grey lines.

When the blue unicycle drives around the first corner (d), the green and red unicycle can not stay exactly in the formation, due to the constraints on their inputs. This can be seen in Figure 7.8(b) as the cost function values of the green and red unicycle increase slightly at (d). The green unicycle runs into the velocity constraint, while the red unicycle is limited by the maximum angular velocity. After the corner, the formation is restored again, until the same input limits become active again at (e). Here, the blue unicycle returns to the start, where the other two finally restore the desired formation.

The formation that is presented here is a leader-follower formation, where the blue unicycle is the leader, and the other two are followers. It is also possible to specify different formations such as a virtual structure. A virtual structure is created by mutually coupling unicycles together. Another possibility is to set up a pursuit-evasion formation. To do this, one unicycle tries to get as close as possible to another unicycle. The other unicycle needs to do the same, but if a negative Q_u^u is used, it will try to get away from the other unicycle. Any formation that is specified can be time-varying, so the desired distances and angles can be different at every time instant. This can be used to create a changing formation, but also to split up one formation to form another.

7.6 Limitations on inputs

As seen in previously discussed simulations, the controller can deal with input constraints that become active. To test this property even further, two additional simulations are performed. In one, the angular velocity of the unicycle is fixed, and in the other, the velocity is fixed. The goal of the unicycles is to follow the same reference trajectory as in Section 7.2. The reference trajectories and the actual trajectories are shown in Figure 7.9. Two videos of the simulations are available on YouTube.



(a) Reference tracking with fixed angular velocity.

(b) Reference tracking with fixed forward velocity.

Figure 7.9: Top-down view of two reference tracking simulations with fixed inputs.

As it can be seen in Figure 7.9(a) and (b), in both cases the unicycles are still roughly able to fulfill their control objectives, despite one input being fixed. Both simulations show distinct behavior. Figure 7.9(a) shows regular switching of velocity direction to stay as close as possible to the reference trajectory. Figure 7.9(b) shows the unicycles making circles to stay in the same position. This is because the velocity of the unicycle is higher than the velocity of the reference trajectory, so it quickly gets ahead of the reference trajectory. By driving in a circle, the unicycle stays roughly in the same position so that it can catch up with the reference trajectory. For this simulation, the values of ω are allowed to be between -4 and 4 [rad/s].

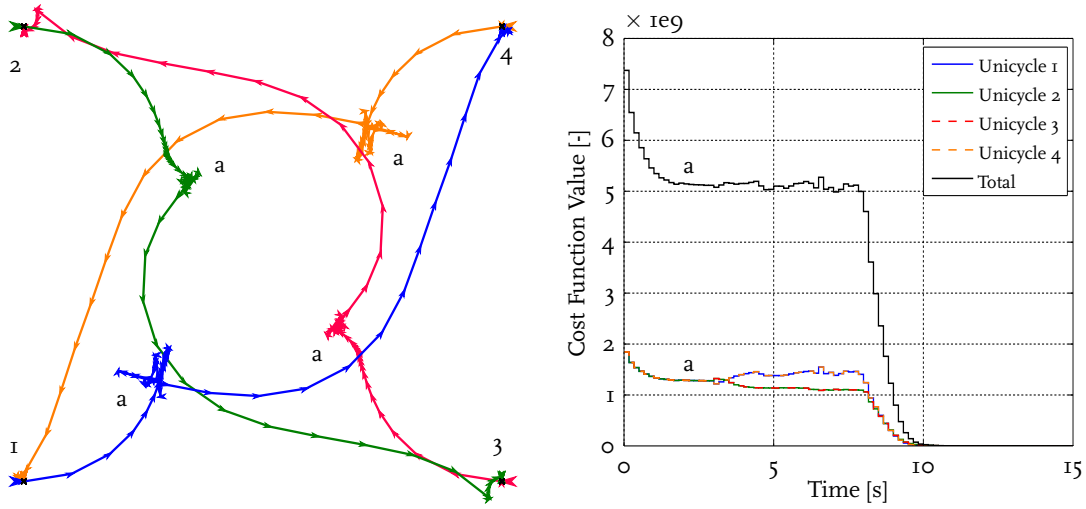
It is also possible to specify less extreme limitations on the inputs, such as inputs that can only have positive values. It is also possible to have time-varying input limitations that are different for each unicycle. If it has to be guaranteed that no collisions occur, the limitations of the velocities should satisfy (5.2) and/or (5.13), depending on the control objectives of the simulation.

7.7 Results with centralized MPC

All simulation that are discussed before, are performed using sequentially decentralized MPC. A number of simulations were performed with centralized MPC as well. The reason that most of these results are not shown here, is because the performance with this control strategy is usually worse. This is caused by the increased complexity of the optimization problem, combined with an optimization algorithm that perhaps is too simple or not suited at all for such an optimization problem.

Problems typically arise when unicycles are in close proximity to each other, allowing their predicted paths cross. There is no longer a predetermined order in which the unicycles plan their paths, so all paths are determined simultaneously. Information from the gradient is used to shift the paths so that a lower cost function value is obtained. A problem that arises during the optimization is that paths are shifted simultaneously. It can happen that the paths of the unicycles are shifted, which will then result in a future collision. This can result in a gradient that points in the opposite direction, leading to paths that are approximately shifted back to their initial form. This process can repeat for a long time, leading to long computation times and poor performance in some cases.

Using centralized MPC can have benefits however. One area where it can have better performance is in resolving conflict situations. This is because the control objectives of all unicycles are treated equally and combined, and the optimization algorithm searches for a more 'global' optimum of unicycle inputs. To test the resolution of conflict situations, the simulation from Section 7.4 is repeated, but now centralized MPC is used. The trajectories of the unicycles are shown in Figure 7.10(a), and the cost function values are shown in Figure 7.10(b). A video of this simulation is available on YouTube.



(a) Trajectories of the unicycles.

(b) Cost function values.

Figure 7.10: Top-down view and cost function values of the unicycles in the simulation of unicycle collision avoidance.

In comparison with the simulation from Section 7.4, there is a significant difference in terms of trajectories that the unicycles take, as it can be seen in Figure 7.10(a). Initially, the unicycles all move towards the center, where they stop to avoid collisions. Then they remain stationary for a while (a), as it can be seen in the constant cost function values in Figure 7.10(b). While they remain stationary, the unicycles keep on rotating. Eventually, after 8 seconds, the optimization algorithm is able to find paths that lead all unicycles to their desired positions. These paths solve this problem in a more optimal way than in Section 7.4.

Even though it takes longer to solve this problem than with sequentially decentralized MPC, this simulation shows that centralized MPC has potential. If the algorithm was able to find these paths directly, the problem would be solved faster than with sequentially decentralized MPC. However, this is not the case. It also takes a fair bit of tuning to obtain this result. Another problem is that the computation times with this control method exceed the sampling time when inputs are calculated, which is further discussed in the next section. Because of this problem, no experiments were performed using centralized MPC.

7.8 Other properties of the controller

In the simulations that are discussed here, each unicycle only has a single control objective. It is also possible to specify multiple control objectives, that may conflict with each other. When a unicycle is faced with this, it will determine what to do based on the penalties that are linked to the control objectives. This usually involves that the unicycle will position itself somewhere in between the control objectives. Furthermore, it is possible to vary many of the parameters and control objectives in the course of the simulation. This is possible because the inputs are determined using an optimization algorithm, which is able to solve a different optimization problem at each sampling instant. The advantage of such a controller is that the control objectives can be specified with a lot of freedom and flexibility.

To keep the behavior of the unicycles deterministic, the new control inputs should always be calculated within one sampling time. However, when the cost function and its gradient and Hessian are numerically evaluated, the calculation times quickly become longer than the sampling time. To resolve this problem, the results obtained in Section 6.3 are simplified according to the rules that are described in Appendix A. The time needed to calculate the control inputs with these simplified results, is shown in Table 7.6. The results were obtained using an Intel Core Duo E6600 Processor, running at 2.4 GHz, which is launched at the end of 2006.

Section of simulation	7.2	7.3	7.4	7.5	7.6(a)	7.6(b)	7.7
Total simulation time [s]	30	30	5	30	30	30	15
Time calculating [s]	0.5	0.7	0.6	4.0	1.6	0.7	22.5
Percentage of T_s used [-]	1.6	2.4	11.4	13.2	5.2	2.4	148.6

Table 7.6: Calculation times.

The table shows the total simulation time, the time it took to calculate the control inputs, and the average percentage that is used of each sampling time. There is some variation in the calculation times at each sampling instant, but the calculation times never exceed the sampling time, except with the centralized controller. As it can be seen, there is quite some room to decrease the minimum step size, add more unicycles or obstacles, or decrease the sampling time.

7.9 Conclusions

In the simulations that are discussed here, the behavior and performance of the sequentially decentralized, and centralized control strategies are discussed. Sections 7.2 to 7.6 present simulations that use sequentially decentralized MPC. The centralized version of the controller is discussed in Section 7.7. Both controllers can fulfill the different control objectives that are implemented in the cost function. Sequentially decentralized MPC outperforms centralized MPC in every way, both considering calculation time, as the time it takes to fulfil control objectives. However, it is clear that centralized MPC has the potential to outperform sequentially decentralized MPC regarding the time required to fulfill the control objectives. However, this requires improvement of the optimization algorithm such as a way to handle the increased complexity of the optimization problem in a less computationally intensive way. Because the optimization algorithm is not developed further, the experiments have all been performed using sequentially decentralized MPC. The experiments are discussed in the next chapter.

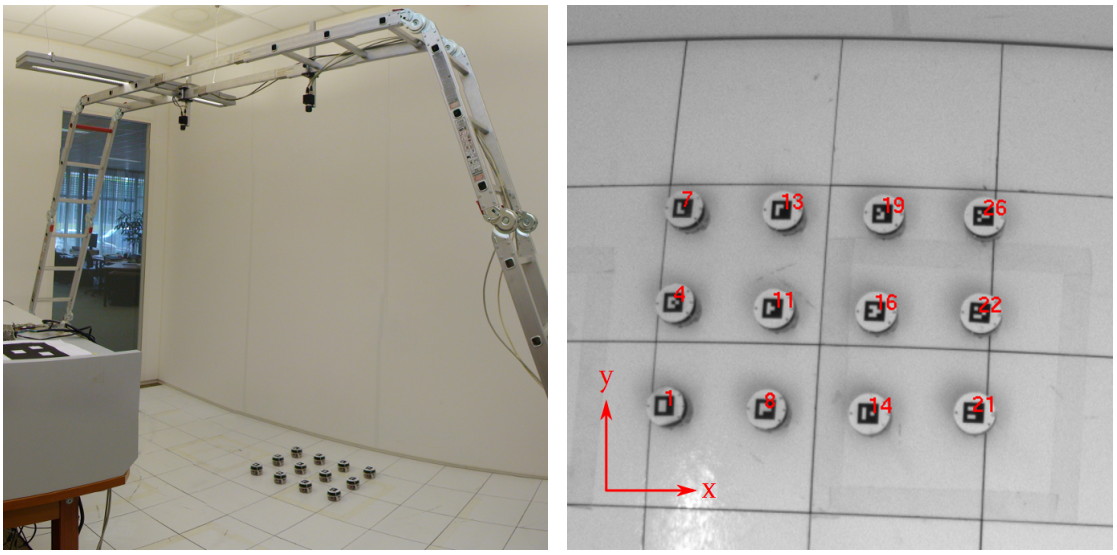
Chapter 8

Experimental Results

In this chapter, the results of the performed experiments are discussed. First, the experimental setup is discussed, followed by a description of how the results are presented. After that, the experimental results are discussed. Finally, a comparison is made between the simulation and experimental results, and potential reasons for the difference are given.

8.1 Experimental setup

The simulation results show that the designed controller performs well and is able to fulfill the implemented control objectives. However, it needs to be verified how the controller performs on real unicycles. Therefore, a number of experiments are performed that use the e-puck nonholonomic mobile unicycle robot, which is depicted in Figure 1.2(a). The e-puck is a small mobile robot, which has been specifically developed for various teaching purposes. More information on the design of the e-puck can be found in [42]. The goal of the experiments is to verify whether the control objectives can be fulfilled as is observed in the simulations. Two different views that show the experimental setup are depicted in Figure 8.1.



(a) Overview of the experimental setup.

(b) Right camera view with recognized e-pucks.

Figure 8.1: Two views of the experimental setup.

The experimental setup with the e-pucks is depicted in Figure 8.1(a). It consists of a total of 12 e-pucks that can be used, a PC, and two cameras. The e-pucks drive around on the white floorboards. Each e-puck is equipped with a unique visual pattern on top. Above the setup, two cameras are mounted, which provide an overview of the e-pucks to a PC, which is located next to the setup. On the PC, the images from the cameras are received and the images are processed. Once the patterns are recognized,

the position and rotation of the e-pucks are available on the PC. An example of a camera image that has been processed is depicted in Figure 8.1(b).

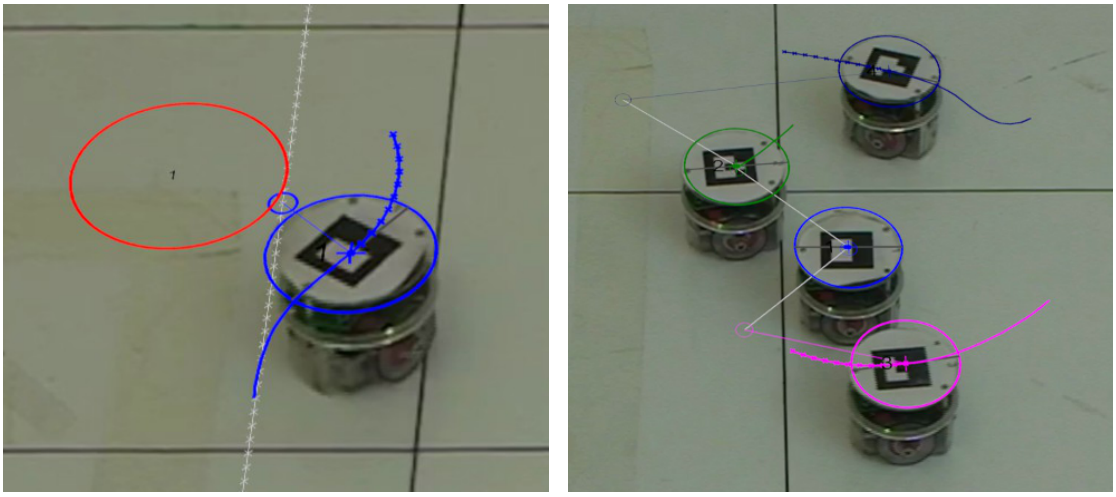
After a camera image is processed and the current orientations of the e-pucks are sent to MATLAB, the optimization algorithm can start calculating new control inputs for each e-puck. When the next sampling instant occurs, the new desired wheel velocities, calculated by (4.1), are sent to the e-pucks. This is done via a Bluetooth adapter that is connected to the PC. Once an e-puck receives new desired wheel velocities, a controller inside the e-puck adjusts the wheel velocities.

In the experiments that are discussed here, a sequentially decentralized control strategy is used, in combination with the steepest descent optimization method. The centralized controller requires to much computation time, and it hard to obtain good performance using this method, so it is not feasible to use this controller. Newton's optimization method is not used here, because it proves to be less robust than the steepest descent method.

The PC that is used with the experiments is different from the one that is used for the simulations. It uses an Intel i7 Quad-core Processor, running at 2.4 GHz, which is launched in 2009. This processor is roughly 30% faster in calculations than the processor in the other PC.

8.2 Introduction

The experimental results are presented in the same way as they are in Chapter 7. This means that in the report, only the trajectories of the unicycles are depicted, while a video of the experiment is available online. The videos from the experiments can be found at www.youtube.com/er47ik and clicking on 'Experiments'. Two examples of snapshots of experimental videos are depicted in Figure 8.2.



(a) Unicycle tracking a reference trajectory.

(b) Four unicycles driving in formation.

Figure 8.2: Snapshots of experimental videos that are available online.

In the videos of the experimental results that are available online, a video of the experiment is combined with the controller visualization that is discussed in the previous chapter. This makes it possible to see a video of the e-pucks, along with the path that it has taken and the planned path at each time instant. The result that is displayed over the video is the same visualization that is used in the videos of the simulation results. A more detailed description of what is displayed, can be found in Section 7.1.

Figure 8.2(a) shows a unicycle that is tracking a reference trajectory, which is indicated with the small grey crosses. As a circular red obstacle is preventing the unicycle from driving on the desired trajectory, the unicycle is driving around it. In Figure 8.2(b), four unicycles are depicted that need to form a formation. It is desired that a V-shaped formation is formed behind the blue e-puck. In the snapshot, the green unicycle is in position, while the blue and pink unicycle are on their way.

8.3 Tracking a reference trajectory

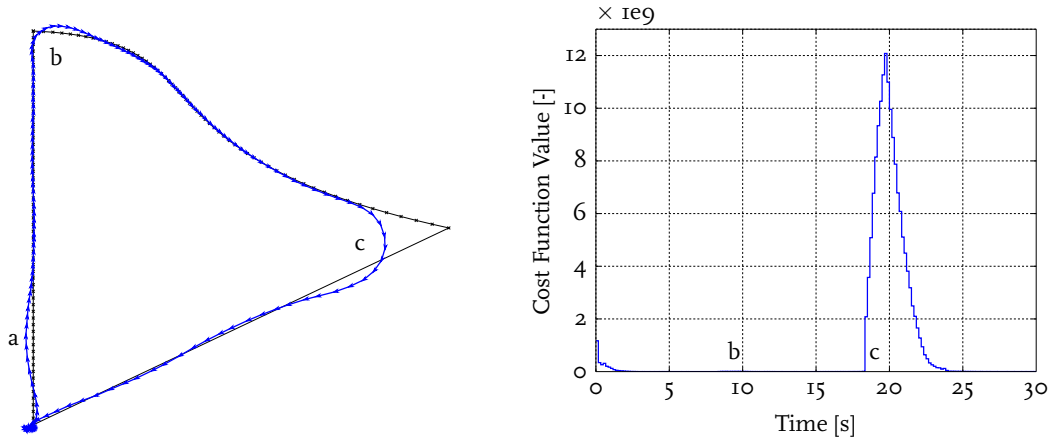
To test the reference tracking ability of the controller in an experiment, the same reference trajectory as in Section 7.2 is used. This trajectory is used to test how the e-puck deals with trajectories that fall within its nonholonomic constraints, and with trajectories that do not have this property. The parameters that are used in this experiment are shown in Table 8.1.

Parameter	N_p [-]	N_c^i [-]	T_s [s]	P_u^r [-]	Q_u^r [-]
Value	10	$\begin{bmatrix}1 & 2\end{bmatrix}$	1/6	1e10	1e6

Table 8.1: Reference tracking experiment parameters.

The penalty values that are used in this experiment are 100 times larger than the parameters in the simulation of the previous chapter. This is done to make the e-puck drive as close as possible to the obstacles in the obstacle avoidance experiment, which is discussed in the next section. The velocity limits of the e-pucks from Table 7.1 are used.

The result of this experiment is depicted in Figure 8.3. The reference and actual trajectories are depicted in Figure 8.3(a). The cost function values are shown in Figure 8.3(b). The separate state references and actual motions in x-, y-, and θ-direction are depicted in Figure 8.4. The green lines indicate the references and the blue lines indicate the actual values. A video of this experiment can be found on YouTube.



(a) Top-down reference and actual trajectories.

(b) Cost function values.

Figure 8.3: Top-down view and cost function values of the reference tracking experiment.

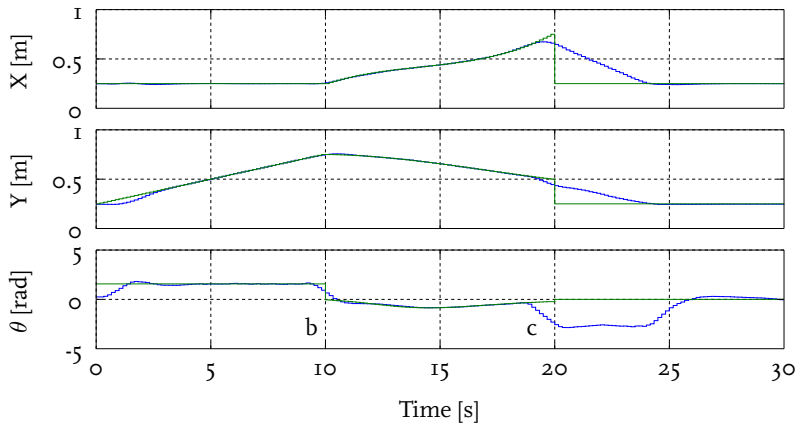


Figure 8.4: Separate state references and actual signals.

As it can be seen from the figures, it is also possible to track a reference trajectory in an experiment. Qualitatively, the experimental results are similar to the simulation result that is discussed in the previous chapter. Overall, the trajectory is followed in a similar way. This is clearly visible when the $90°$ turn is encountered (b), where the e-puck first turns left before turning right.

However, there are several differences with the simulation. The first difference can be seen near the start (a), where the e-puck seems to display overshoot behavior, which is not encountered in the simulation results. The same kind of overshoot is observed at (b) and (c). This indicates that there might be an additional delay active in the system. However, in combination with the relatively long sampling time that is used, this does not cause problems in this experiment.

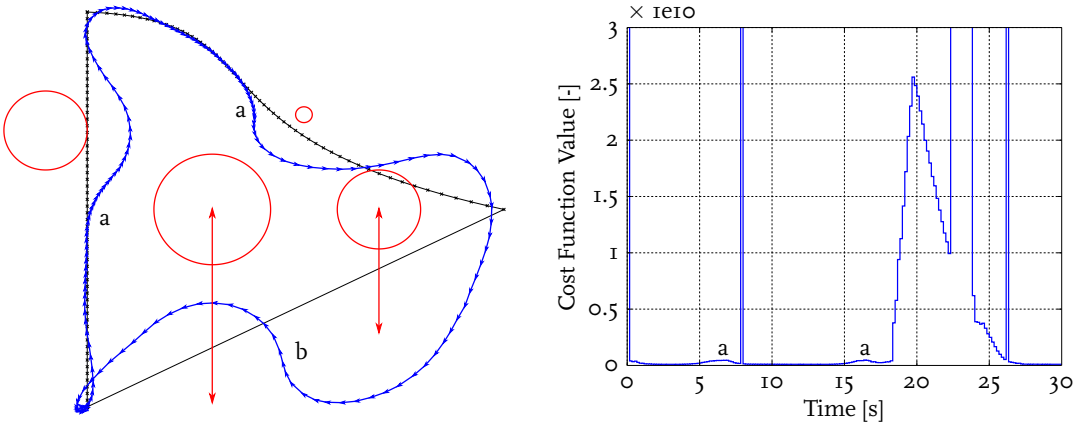
8.4 Avoiding collisions with circular obstacles

To determine how the controller deals with circular obstacles, the same reference trajectory as in the previous section is used, but four circular obstacles are now included in the same workspace. The parameters that are used in this experiment are the same as in the previously discussed one. Additional parameters that are related to obstacle avoidance are given in Table 8.2.

Parameter	$P_u^o(mT_s)$ [-] $(m = k+1)$	$P_u^o(mT_s)$ [-] $(m > k+1)$
Value	1e14	1e12

Table 8.2: Obstacle collision avoidance parameters.

The result of this experiment is depicted in Figure 8.5. The reference and actual trajectories are depicted in Figure 8.5(a). The cost function values are shown in Figure 8.5(b). The separate state references and actual motions in x-, y-, and θ-direction are depicted in Figure 8.6. A video of this experiment can be found on YouTube.



(a) Top-down reference and actual trajectories.

(b) Cost function values.

Figure 8.5: Top-down view and cost function values of the obstacle collision avoidance simulation.

The obstacles that are added to the workspace cause no difficulties for the controller, as the e-puck is able to drive around the static and dynamic obstacles. No collisions occur in this experiment, even though the result of Figure 8.5(b) might suggest otherwise. Due to the increased tracking penalties, the e-puck passes the obstacles with very little room in between. Due to errors in determining the orientation of the e-puck, it seems as though collisions occur. However, due to the safe distance that is used, no collisions occur.

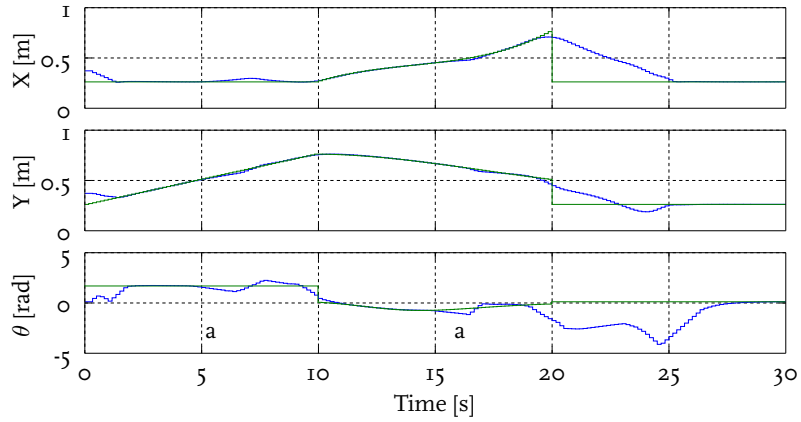


Figure 8.6: Separate state references and actual signals.

The experiment shows the same behavior as the simulation when the static obstacles are approached (a). The e-puck approaches the static obstacles slowly, which is caused by the use of the steepest descent optimization algorithm. This algorithm has trouble finding lower cost function values near the large variations caused by obstacles. An example of this behavior can be seen in Figure 6.2(a). The dynamic obstacles also cause no trouble for the unicycle (b), as the prediction horizon is long enough, and the locations and sizes of the obstacles is already known in advance.

8.5 Avoiding collisions with other unicycles

To test the inter-unicycle collision avoidance and the priority strategy, an experiment is set up that is similar to the simulation of Section 7.4. However, in stead of 4 unicycles, now 12 e-pucks are used that all have to change position. The parameters that are used in this experiment are given in Table 8.3.

Parameter	P_u^r [-]	Q_u^r [-]	$P_u^u(mT_s)$ [-] $(m = k+1)$	$P_u^u(mT_s)$ [-] $(m > k+1)$
Value	1e10	1e6	1e14	1e12

Table 8.3: Unicycle collision avoidance parameters.

The result of this experiment is depicted in Figure 8.7. The trajectories of the e-pucks are depicted in Figure 8.7(a). The modified cost function values and the priorities of the e-pucks are shown in Figure 8.7(b). A video of this experiment can be found on YouTube.

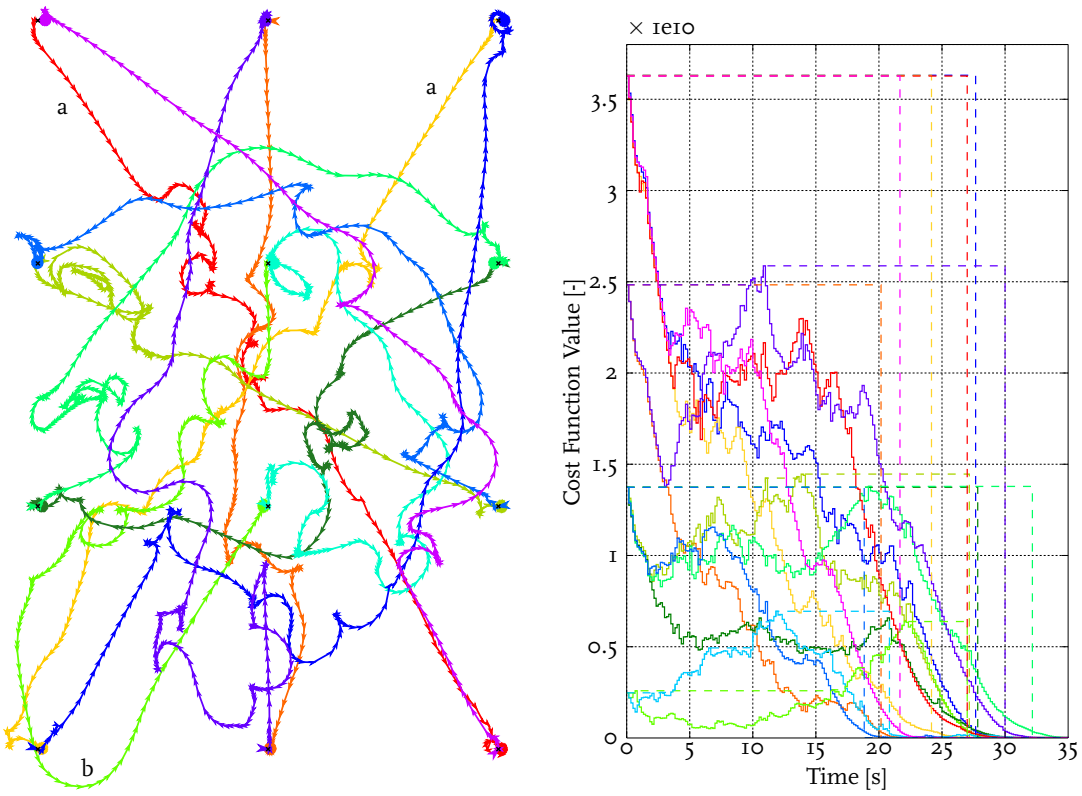
The twelve unicycles are able to switch positions in 35 seconds. Even though the earlier derived safe distances are added to the unicycles, some collisions still occur. However, they are not the result of the planned paths of the controller. Some possible reasons that cause this are presented in Section 8.7.

The paths that the e-pucks take seem chaotic, but when Figures 8.7(a) and 8.7(b) are viewed together, some observations can be made. The four e-pucks that start in the corners (pink, yellow, red, and blue) initially have the largest cost function values, and therefore the highest priorities. These high priorities allow them to drive to their desired orientations in a relatively straight line, which is clearest visible with the trajectories of the red and yellow e-puck (a).

The opposite can be observed with the light green e-puck (b). This e-puck only has to drive a short distance, which means that it starts with the lowest priority value. Because it has such a low priority, it gets pushed out of the way which causes its own priority to rise. Eventually, it is able to reach its desired location.

The other e-pucks are somewhere in between these extremes. Depending on their priority levels, they can "push" others out of the way, or get "pushed" away themselves. The total duration of the experiment is 60 second. The PC spends 21.5 seconds out of the 60 seconds on the calculations of new control inputs. Sometimes, the optimization process almost takes the full sampling time to obtain

new control inputs, but on average, 36% of the total time is used to calculate new control inputs. The other experiments that are discussed in this chapter use less computation time.



(a) Trajectories of the unicycles.

(b) Cost function values (5.22), and priorities (dashed).

Figure 8.7: Top-down view and cost function values of the unicycles in the experiment of unicycle collision avoidance.

8.6 Driving in formation with other unicycles

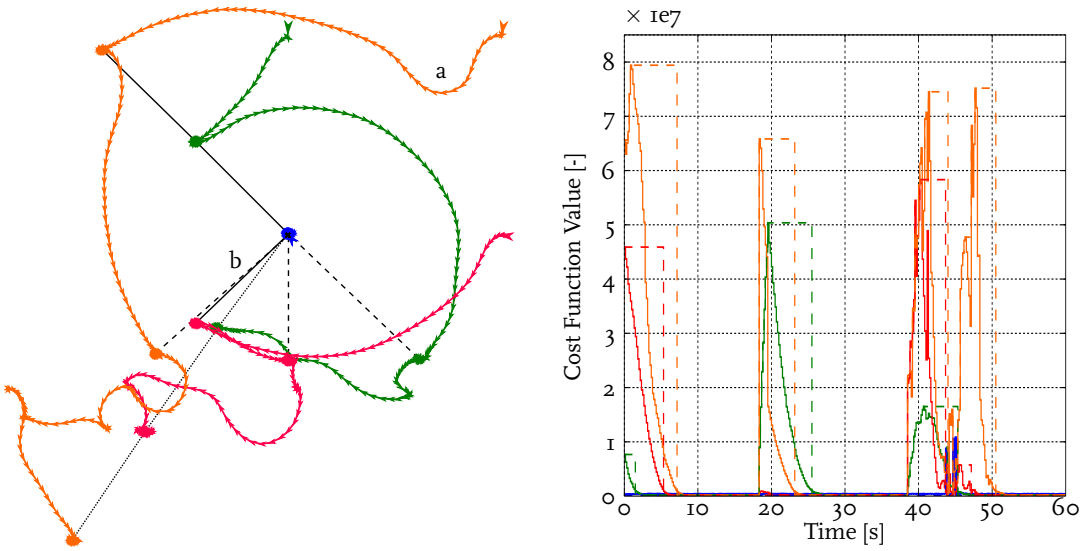
To demonstrate the ability of the controller to deal with time-varying formations, the following experiment is performed. In this experiment, one e-puck remains stationary, while three others form three different formation around it. The first formation needs to be formed from 0-20 seconds, the second from 20-40 seconds, and the third from 40-60 seconds. For this experiment, the parameters from Table 8.4 are used.

Parameter	P_u^r [-]	Q_u^r [-]	Q_u^u [-]	R_u^u [-]	S_u^u [-]
Value	1e10	1e6	1e8	1e6	1e4

Table 8.4: Formation driving experiment parameters.

The results of this experiment are shown in Figure 8.8. The trajectories of the e-pucks are depicted in Figure 8.8(a). The modified cost function values and the priorities of the e-pucks are shown in Figure 8.8(b). A video of this experiment can be found on YouTube.

The three different formations that need to be formed, are depicted in Figure 8.8(a). The first formation is indicated with the solid black lines and requires that the green and red e-puck form a V-shaped formation behind the blue one. The orange e-puck needs to stay at the same distance and angle from the green e-puck. Because of this, the orange e-puck does not initially drive to its desired location (a).



(a) Reference and actual trajectories.

(b) Cost function values (5.22), and priorities (dashed).

Figure 8.8: Top-down view and cost function values of the formation driving simulation.

The next formation need to be achieved from 20-40 seconds. However, the e-pucks start moving sooner than that as it can be seen in Figure 8.8(b). This is caused by the prediction horizon of 10, combined with the sampling time of 1/6 second, resulting in a controller that can look about 1.5 seconds ahead into the future. The second formation couples the three e-pucks directly to the first one. This formation is indicated with the dashed lines. Here, it can be observed that there is a small error in determining the angle of the blue e-puck (b), as the solid and dashed lines should overlap each other.

Finally, the third formation couples the orange to the red, the red to the green, and the green to the blue e-puck. The goal of this coupling is to maintain a given distance and angle from another e-puck. This angle is slightly different from the angle of the two previous formations. In this experiment, the formation changes only three times, but it is also possible to specify a different formation at every time instant.

8.7 Comparison with simulation results

The results that are obtained in the experiments are qualitatively identical to the simulation results. The path that the e-puck takes when tracking a reference trajectory show the same characteristics. Also, when multiple e-pucks are driving in close proximity to each other, the behavior is similar to the simulations with high priority e-pucks planning paths through ones with a lower priority. However, quantitatively the results are not identical, and in general the performance in the experiments is worse. There are most likely a lot of factors that contribute to this decreased performance, some of which are discussed here. It has not been investigated how much each of these factors contribute in widening the gap between simulation and experiment. It is possible to compensate for some of these factors, but this is not further investigated in this thesis.

The first difference between simulation and experiment results from determining the orientation of the e-pucks. In a simulation, this is trivial, but in the experiments an image processing system is used. This system uses two cameras to determine the position and orientation of the e-pucks. The accuracy of this system is tested by determining the position and rotation of a stationary e-puck. It turns out that the orientation along the x-axis (depicted in Figure 8.1(b)) varies by 1.2 mm, the y-axis by 0.4 mm, and the θ-axis by 2.5°. So there are errors in the measurement, especially in the rotation of the e-pucks. A different problem that is caused by the use of a camera, is that it captures images at only 30 Hz. When a request for the orientations of the e-pucks is made in MATLAB, the image processor waits for the camera to capture a frame, which then needs to be processed. After the orientation of the e-pucks is determined, the orientations are sent to MATLAB. This means that no instant orientation information is available. The delay that this process causes is somewhere between 50 and 60 ms.

After the orientation of the e-pucks is sent to MATLAB, and the computation of the next control inputs is complete, they need to be sent to the e-pucks. This is done in parallel using a Bluetooth adapter. This can cause a new set of problems, as the e-pucks are controlled using a networked control system, which can introduce packet dropouts, leading to delays. If a single packet is lost, a delay of the sampling time is introduced. Packet dropouts can clearly be observed when the batteries of the e-pucks are almost empty. When the batteries are full, dropouts can not visually be detected, but they can still be present.

Another source of errors comes from the fact that the e-pucks are modeled using a kinematic model. As the e-puck has mass, it can not instantaneously change velocity, creating a difference between simulation and experiment. Another problem is that the e-puck only has two wheels, and slides over the ground on one side, while the other side sticks approximately 1 mm in the air. When the e-puck stops or changes driving direction, it can tip over and drive on the other side. This can cause the e-puck visual pattern to move, while the e-puck is not driving.

8.8 Conclusions

From the experiments that are performed, it can be concluded that the designed controller is able to let an e-puck track reference trajectories, avoid circular obstacles and other e-pucks, and drive in a formation. The behavior that is observed in the experiments is qualitatively similar to behavior in simulations. However, quantitatively there are some difference with the simulations. The experimental setup introduces time-varying delays when the orientation of the e-pucks is requested, errors when determining the orientation of the e-pucks, and differences between the used model and the e-puck. Even though no attention is payed to trying to compensate for these difference, the designed controller still qualitatively performs as expected.

Chapter 9

Conclusions and Recommendations

In this chapter, a summary of the conclusions of the thesis is given, and finally some general conclusions are drawn. After that, a number of recommendations for future research are given that discuss how aspects of the controller can be improved, or altered.

9.1 Conclusions

In this thesis, a controller is developed which is able to control multiple unicycles simultaneously. This controller is able to make the unicycles track reference trajectories, avoid circular obstacles and each other, and drive in a formation. These control objectives are fulfilled by using model predictive control, of which the general principle is first discussed. This general principle is then adjusted to allow it to be used in systems with short time scale behavior, which is done to keep the system behavior deterministic. Finally, different control strategies are discussed that cover the control of multiple systems that operate in the same environment. Of those two strategies, sequentially decentralized, and centralized MPC are chosen to be further investigated.

The continuous-time model of the unicycle is introduced next, and it is exactly discretized under the assumption that the input remains constant between two sampling instants. A free predicted input vector is introduced to reduce the complexity of the optimization problem. The input vector is used in combination with the exactly discretized model to obtain the predicted outputs of a unicycle.

The predicted outputs are compared to their control objectives in the cost function, which returns a scalar cost function value as a function of the predicted inputs. In the cost function, reference tracking, circular obstacle and other unicycle avoidance, and formation keeping are implemented. For collision avoidance, two worst case scenarios are investigated and safety measures are introduced to prevent inter-sample collisions. To resolve conflicting situations that might occur, a priority strategy is developed that uses present, and past cost function values.

To obtain the lowest cost function values, two local optimization algorithms are introduced: the steepest descent, and Newton's optimization method. These methods use the exact gradient and Hessian of the cost function. Both methods are used in combination with line search to obtain robust optimization algorithms. Newton's method is able to produce good results as long as the Hessian is positive definite. However, as this is not always the case, the steepest descent method performs more consistently.

When all the elements of the controller are combined and put to the test in the simulations, it turns out that all the implemented control objectives can be fulfilled. In simulations with multiple unicycles, it can be guaranteed that no collisions occur as long as there are no dynamic obstacles. Also, conflicting situations can be resolved quickly thanks to priorities that are assigned, based on the cost function values. Already in simulation, it becomes clear that sequentially decentralized MPC performs better than centralized MPC. This is caused by the increased complexity of the optimization problem. However, it can be seen that this control strategy has the potential to outperform sequentially decentralized MPC, if the optimization algorithm is improved.

In the experiments that are conducted, the same behavior is observed as in the experiments. The reference trajectories are followed in the same manner, and when obstacles are encountered, the e-pucks approach them slower, just as in the simulations. The priority system also allows the unicycles to resolve conflicting situations. However, there are multiple factors that decrease the performance of the experiments, such as delays and errors in obtaining the orientation of the e-pucks. Even though the influence of these effects is not investigated further, the control objectives can still be fulfilled.

The controller that is described in this thesis, is able to control multiple nonholonomic mobile agents simultaneously. The controller is flexible as it accepts time-varying control objectives. It is also robust against the disturbances that are caused by the experimental setup. Furthermore, it is able to resolve conflicting situations when multiple unicycles drive in close proximity. The control inputs are also calculated in a fast way, thanks to significant simplification of the symbolic expressions.

9.2 Recommendations

Even though the controller that is designed in this thesis is able to fulfill the specified control objectives, it still has a number of limitations. One problem is that the optimization algorithms that are used, can only find local optima. This means that a unicycle that is stuck behind a number of obstacles, sometimes can not plan a path around them. One solution would be to use global optimization algorithms, but they would drastically increase computation times. A better approach would be to combine the controller with a path planning algorithm, that determines waypoints that a unicycle can follow. In this way, the controller can still be used when the unicycles have navigate a complex environment.

Another issue is that the performance in the simulations is not matched by the experiments. To be able to improve the experimental performance, the setup should be thoroughly investigated. Particular attention should be payed to the delays that occur when the position is determined, the reliability of the Bluetooth connection, and the mechanical limitations of the e-pucks. When the properties of the setup are known, it can be investigated whether measures can be taken to improve performance.

As there is always room for improvement, some ideas are presented here to add or change the functionality of the controller. The following ideas are changes that can be made to the cost function. In the current cost function, two individual penalty terms are used for reference tracking. One term penalizes the distance that a unicycle is from its desired position, while another penalizes the difference between the actual and desired angle of the unicycle. In the simulations and experiments, the penalty on the angle is chosen much smaller than the penalty on the position. However, the penalty on the angle is only used to fully constrain the orientation of the unicycle. When a unicycle is not yet on its position, the penalty on the angle is generally not useful. An idea to resolve this, is to use an angular penalty term that is a function of the distance that the unicycle has to its desired position. This would result in a unicycle that drives to its position, which leads to an increased angular penalty, that leads to a unicycle that rotates to its desired angle. This idea could also be applied to the formation penalty that couples the rotation of the unicycles.

As mentioned earlier in this thesis, a tradeoff needs to be made between the physical prediction length and the sampling time when the prediction horizon is fixed. Because the sampling time is used in the prediction of future outputs, a shorter sampling time leads to a shorter physical prediction length. As it can be seen in Table 7.6, there is room to decrease the sampling time, but this would negatively influence the performance of the controller. A solution to this problem is to use two different sampling times. One shorter sampling time for the controller, and a longer sampling time for the prediction. This change would result in roughly the same performance, with smoother control signals. A difficulty with two different sampling times is that the control objectives need to be specified for both sampling times.

The cost function is designed so that it can be differentiated to the free inputs, leading to the exact gradient and Hessian. This poses some constraints on the functions that can be used in the cost function, as they need to be differentiable. If this requirement is dropped, additional control objectives can be added to the cost function. This would allow the use of non-circular obstacles. When non-circular obstacles are used, it becomes harder, or even impossible to determine an analytical expression for this distance. A solution to this problem is given in [51], where a map that holds the distance to the closest

obstacle at each position, is precomputed before a simulation or experiment takes place. During the simulation, the distance map can be used to quickly determine the shortest distance to an obstacle, which can then be used to determine a cost function value.

If the cost function does not need to be differentiated, it is also possible to use a wide range of functions, such as min, max, abs, sign, and many others. It is then also possible to use if-statements in the cost function. Due to the addition of these functions and the precomputed distance map, the gradient and Hessian need to be approximated. An approximation can be obtained by determining the cost function values close to the current free inputs.

Changes can also be made on the optimization algorithms. The computation times of the controller have been reduced by the simplification that is discussed in Appendix A. However, the controller is still relatively computationally intensive. When more and more unicycles are added to the simulation or experiment, the computational load increases rapidly. This is because the distance between every unicycle is calculated. This is not always necessary, as unicycles that are far from each other can not collide. A reduction of computation times can be achieved by grouping unicycles that are close to each other, and only avoiding other members of these groups. These groups can also be used to create a new sequentially centralized control strategy, in which the paths of one group is determined simultaneously, leading to less complex optimization problems.

Even though the problem description is limited to a system with multiple unicycles, it would be no problem to use different systems. If a number of holonomic mobile robots has to fulfill the same control objectives, the same design framework can be used. The unicycle model is then replaced with a simpler system model. Most of the cost function can then be reused, as well as the optimization algorithm. A very different example where this design framework can be used, is on the motion generation of a robotic arm. The unicycle model can then replaced with a forward kinematic model of the arm. After that, reference tracking of an end defector, or other part of the arm can be implemented. By attaching a chain of circular obstacles to the arm, and to the environment, collision free trajectories can be generated. As long as the cost function remains differentiable to the free inputs, the simplification of Appendix A can be used again to reduce computation times.

Appendix A

Simplification of Obtained Results

The optimization algorithms, that are discussed in Chapter 6, use the gradient and the Hessian of the cost function. The derivation of these is briefly discussed in Section 6.3. The actual derivation is performed using MATLABs Symbolic Math Toolbox. If the results from the derivation would be used in their initial form, it would create a large computational load as the expressions that occur in the gradient and Hessian are very long. Fortunately, it is possible to reduce the computational load by making use of the fact that many terms in the gradient and Hessian occur multiple times. By gathering these terms and ensuring that each term is only calculated once, a large reduction in computation time can be achieved.

This method works by first gathering every sine and cosine term that occurs in the expressions. When a new term is encountered, it is stored in a database. When the same term is encountered again, it is replaced with the corresponding database entry.

After that, the modified terms without sines and cosines are further simplified. Terms that are connected with a plus or minus sign are separated, which starts with the terms that are between the most inner brackets. An example of this simplification is shown in Table A.1, where a term is simplified using these rules. While the term is simplified, its components are stored in a database which is shown on the right. Any subsequent term that is simplified, has access to the terms that are already in the database, and can add new terms.

Step	Equation	Database
	$(\sin(5x) + \cos(2x))^2 + 3x(\sin(5x) + \cos(2x))$	
1	$(D_1 + \cos(2x))^2 + 3x(D_1 + \cos(2x))$	$D_1 = \sin(5x)$
2	$(D_1 + D_2)^2 + 3x(D_1 + D_2)$	$D_2 = \cos(2x)$
3	$D_3^2 + 3xD_3$	$D_3 = D_1 + D_2$
4	$D_4 + 3xD_3$	$D_4 = D_3^2$
5	$D_4 + D_5$	$D_5 = 3xD_3$

Table A.1: Example of simplification of an expression.

Three MATLAB functions are automatically created for use in the optimization process. One returns the cost function value of a predicted input vector, the second one returns the cost function value and the gradient, and the third one returns the cost function value, the gradient, and the Hessian. These functions have been created because it is not always necessary to calculate the gradient and the Hessian. Depending on the optimization method that is used, different functions are called.

After this simplification is applied to the three functions, the computation times of a simulation or experiment are decreased significantly. The speedup is around a factor 10 when a single unicycle is following a reference trajectory when $N_c^i = 1$ and $N_p = 5$. However, when there are nine unicycles that have to exchange positions (as in Section 7.4) in a 60 second simulation with $N_c^i = \begin{bmatrix} 1 & 2 \end{bmatrix}$ and $N_p = 5$, the speedup is even more significant, as it can be seen in Table A.2.

	Calculation time [s]		File size [kB]		
	Steepest	Newton	Cost	Gradient	Hessian
Not simplified	120	1711	21	157	1321
Simplified	6.5	9.3	11	32	106
Factor	18.5	184	1.9	4.9	12.5

Table A.2: Example of simplification of two expressions.

Bibliography

[1] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsly & R. Siegwart. *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*. International Symposium on Distributed Autonomous Robotic Systems, Lausanne, Switzerland, 2010.

[2] A. Alvarez Aguirre. *Discrete-time model of a unicycle mobile robot*. Eindhoven, 2009.

[3] T. Arai, E. Pagello & L. E. Parker. *Editorial: Advances in Multi-Robot Systems*. Robotics and Automation, Volume 18, Issue 5, pp. 655-661, 2002.

[4] A. M. Bloch. *Nonholonomic Mechanics and Control*. Springer-Verlag, New York, 2003.

[5] T. H. A. van den Broek, N. van de Wouw & H. Nijmeijer. *A Virtual Structure Approach to Formation Control of Unicycle Mobile Robots*. IEEE Conference on Decision & Control, Shanghai, CH, pp. 8328-8333, 2009.

[6] T. H. A. van den Broek. *Formation Control of Unicycle Mobile Robots*. Master's thesis, Eindhoven University of Technology, 2008.

[7] G. Chaloulos, P. Hokayem & J. Lygeros. *Distributed Hierarchical MPC for Conflict Resolution in Air Traffic Control*. American Control Conference, Baltimore, USA, pp. 3945-3950, 2010.

[8] J. Chena, D. Sun, J. Yang & H. Chen. *A Leader-Follower Formation Control of Multiple Nonholonomic Mobile Robots Incorporating a Receding-horizon Scheme*. The International Journal of Robotics Research, Volume 29, pp. 727-747, 2010.

[9] Y. Chen & Z. Wang. *Formation Control: A Review and A New Consideration*. International Conference on Intelligent Robots and Systems, Edmonton, Canada, pp. 3664-3669, 2005.

[10] M. Defoort, A. Kokosy, T. Floquet, W. Perruquetti & J. Palos. *Motion planning for cooperative unicycle-type mobile robots with limited sensing ranges: A distributed receding horizon approach*. Robots and Autonomous Systems, Volume 57, pp. 1094-1106, 2009.

[11] J. P. Desai, J. Ostriwski & V. Kumar. *Controlling formations of multiple mobile robots*. International Conference on Robotics & Automation, Leuven, Belgium, pp. 2864-2869, 1998.

[12] D. V. Dimarogonas, S. G. Loizou, K. J. Kyriakopoulos & M. M. Zavlanos. *A Feedback Stabilization and Collision Avoidance Scheme for Multiple Independent Non-point Agents*. Automata, Volume 42, pp. 229-243, 2006.

[13] K. D. Do & J. Pan. *Nonlinear formation control of unicycle-type mobile robots*. Robotics and Autonomous Systems, Volume 55, Issue 3, pp. 191-204, 2007.

[14] W. B. Dunbar & R. M. Murray. *Receding Horizon Control of Multi-Vehicle Formations: A Distributed Implementation*. IEEE Conference on Decision and Control, Paradise Island, Bahamas, pp. 1995-2002, 2004.

[15] Embedded Systems Institute. *Falcon, Research Topic: System Performance and Reliability*. http://www.esi.nl/frames.html?/projects/falcon.html.

[16] H. A. van Essen & H. Nijmeijer. *Non-linear model predictive control of constrained mobile robots*. European Control Conference, Porto, Portugal, pp. 1157-1162, 2001.

[17] W. Evers & H. Nijmeijer. *Practical stabilization of a mobile robot using saturated control*. Proceedings of the Conference on Decision and Control, San Diego, USA, pp. 2394-2399, 2006.

[18] F. Fahimi, C. Nataraj & H. Ashrafiuon. *Real-time obstacle avoidance for multiple mobile robots*. Robotica, Volume 27, pp. 189-198, 2008.

[19] R. Fierro & F.L. Lewis. *Control of a Nonholonomic Mobile Robot: Backstepping Kinematics into Dynamics*. Proceedings of the Conference on Decision & Control, New Orleans, USA, pp. 3805-3810, 1995.

[20] M. Fliess, J. Levine, P. Martin & P. Rouchon. *Flatness and Defect of Nonlinear Systems: Introductory Theory and Examples*. International Journal of Control, Volume 61, pp. 1327-1361, 1995.

[21] E. Franco, L. Magni, T. Parisini, M. M. Polycarpou & D. M. Raimondo. *Cooperative Constrained Control of Distributed Agents With Nonlinear Dynamics and Delayed Information Exchange: A Stabilizing Receding-Horizon Approach*. IEEE Transactions on Automatic Control, Volume 53, Issue 1, pp. 324-338, 2008.

[22] J. B. Froisy. *Model predictive control: Past, present and future*. ISA Transactions, Volume 33, pp. 235-243, 1994.

[23] H. Fukushima, K. Kon & F. Matsuno. *Distributed Model Predictive Control for Multi-Vehicle Formation with Collision Avoidance Constraints*. Proceedings of the Conference on Decision and Control, Seville, Spain, pp. 5480-5485, 2005.

[24] J. Ghommam, H. Mehrjerdi, M. Saad & F. Mnif. *Formation path following control of unicycle-type mobile robots*. Robotics and Autonomous Systems, Volume 58, pp. 727-736, 2010.

[25] Grand Cooperative Driving Challenge. http://www.gcdc.net/.

[26] D. Gu & H. Hu. *A Stabilizing Receding Horizon Regulator for Nonholonomic Mobile Robots*. IEEE Transactions on Robotics, Volume 21, Issue 5, pp. 1022-1028, 2005.

[27] D. Gu & H. Hu. *Receding Horizon Tracking Control of Wheeled Mobile Robots*. IEEE Transactions on Control Systems Technology, Volume 14, Issue 4, pp. 743-749, 2006.

[28] D. Gu & H. Hu. *Unicycle-like Vehicle Parking via Receding Horizon Control*. Proceedings of the IEEE Conference on Robotics and Biomimetics, Shenyang, China, pp. 379-384, 2004.

[29] Y. Kanayama, Y. Kimura, F. Miyazaki & T. Noguchi. *A Stable Tracking Control Method for an Autonomous Mobile Robot*. IEEE Conference on Robotics and Automation, Cincinnati, USA, pp. 384-389, 1990.

[30] T. Keviczky, F. Borelli, K. Fregene, D. Godbole & G. J. Balas. *Decentralized Receding Horizon Control and Coordination of Autonomous Vehicle Formations*. IEEE Transactions on Control Systems Technology, Volume 16, Issue 1, pp. 19-33, 2008.

[31] D. E. Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, New York, USA, 1998.

[32] I. Kolmanovsky & N. H. McClamroch. *Developments in Nonholonomic Control Problems*. IEEE Control Systems Magazine, Volume 15, Issue 6, pp. 20-36, 1995.

[33] F. Kühne & J. M. G. da Silva Jr. *Mobile Robot Trajectory Tracking Using Model Predictive Control*. Proceedings of the IEEE Latin American Robotics Council, São Luís, Brazil, pp. 1-6, 2005.

[34] L. Lapierre, R. Zapata & P. Lepinay. *Simultaneous Path Following and Obstacle Avoidance Control of a Unicycle-type Robot*. IEEE Conference on Robotics and Automation, Rome, Italy, pp. 2617-2622, 2007.

[35] J-P. Laumond et al. *Robot Motion Planning and Control*. Laboratory of Analysis and Architecture of Systems, Toulouse, France, 1999.

[36] Q. Li & Z. Jiang. *Formation Tracking Control of Unicycle Teams with Collision Avoidance*. IEEE Conference on Decision and Control, Cancun, Mexico, pp. 496-501, 2008.

[37] H. Lim, Y. Kang, J. Kim & C. Kim. *Formation Control of Leader Following Unmanned Ground Vehicles using Nonlinear Model Predictive Control*. IEEE Conference on Advanced Intelligent Mechatronics, Singapore, pp. 945-950, 2009.

[38] H. Lim, Y. Kang, C. Kim, J. Kim & B. You. *Nonlinear Model Predictive Controller Design with Obstacle Avoidance for a Mobile Robot*. IEEE Conference on Mechatronic and Embedded Systems and Applications, Bejing, China, pp. 494-499, 2008.

[39] S. Majima. *On a Conventional Receding Horizon Regulator for a Nonholonomic Mobile Robot*. IEEE International Conference on Control and Automation, Christchurch, New Zealand, pp. 139-144, 2009.

[40] S. Mastellone, D. M. Stipanovic, C. R. Graunke, K. A. Intlekofer & M. Spong. *Formation Control and Collision Avoidance for Multi-agent Non-holonomic Systems: Theory and Experiments*. The International Journal of Robotics Research, January pp. 107-126, 2008.

[41] D. Mellinger, M. Shomin, N. Michael & V. Kumar. *Cooperative Grasping and Transport using Multiple Quadrotors*. Philladephia, USA, 2010.

[42] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. Zufferey, D. Floreano & A. Martinoli. *The e-puck, a Robot Designed for Education in Engineering*. Conference on Autonomous Robot Systems and Competitions, Volume 1, pp. 59-65, 2009.

[43] R. M. Murray & S. S. Sastry. *Nonholonomic Motion Planning: Steering Using Sinusoids*. IEEE Transactions on Automatic Control, Volume 38, pp. 700-716, 1993.

[44] P. A. Niño-Suarez, E. Aranda-Bricaire & M. Velasco-Villa. *Discrete-time sliding mode path-tracking control for a wheeled mobile robot*. IEEE Conference on Decision & Control, San Diego, USA, pp. 3052-3057, 2006.

[45] P. Y. Papalambrso & D. Wilde. *Principles of Optimial Design*. Cambridge University Press, Cambridge, UK, 2000.

[46] L. E. Parker. *Current state of the art in distributed autonomous mobile robots*. Distributed Autonomous Robotic Systems, Volume 4, pp. 3-12, 2000.

[47] Z. Qu. *Cooperative Control of Dynamical Systems: Applications to Autonomous Vehicles*. Springer-Verlag, London, 2009.

[48] G. P. Roussos, G. Chaloulos, K. J. Kyriakopoulos & J. Lygeros. *Control of multiple non-holonomic air vehicles under wind uncertainty using model predictive control and decentralized navigation functions*. IEEE Conference on Decision and Control, Cancun, Mexico, pp. 1225-1230, 2008.

[49] I. Rañó & C. Burbridge. *Motion camouflage for unicycle robots using optimal control*. 11th Conference Towards Autonomous Robotic Systems, Plymouth, UK, pp. 219-225, 2010.

[50] J. A. Reeds & L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. Pacific Journal of Mathematics, Volume 145, pp. 367-393, 1990.

[51] M. Saska. *Trajectory planning and optimal control for formations of autonomous robots*. Dissertation, Würzburg, Germany, 2009.

[52] P. S. Schenker, T. L. Huntsberger, P. Pirjanian, E. T. Baumgartner & E. Tunstel *Planetary Rover Developments Supporting Mars Exploration, Sample Return and Future Human-Robotic Colonization*. Autonomous Robots 14, pp. 103-126, 2003.

[53] J. Shin & H. J. Kin. *Nonlinear Model Predictive Formation Flight*. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, Volume 39, Issue 5, pp. 1116-1125, 2009.

[54] J. Snape, J. van den Berg, S. J. Guy & D. Manocha. *Smooth and Collision-Free Navigation for Multiple Robots Under Differential-Drive Constraints*. IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, pp. 4584-4589, 2010.

[55] D. Sun, C. Wang, W. Shang & G. Feng. *A Synchronization Approach to Trajectory Tracking of Multiple Mobile Robots While Maintaining Time-Varying Formations*. IEEE Transactions on Robotics, Volume 25, pp. 1074-1086, 2009.

[56] H. J. Sussmann & G. Tang. *Shortest Paths For The Reeds-Shepp Car: A Worked Out Example Of The Use Of Geometric Techniques In Nonlinear Optimal Control*. SYCON Report, 1991.

[57] Tech United Eindhoven. http://www.techunited.nl/nl/home.

[58] J. Verriet. *Autonomous Vehicles in a Distribution Center*. Vanderlande Industries, 2008.

[59] H. Voos. *Model Predictive Collaborative Motion Planning and Control of Mobile Robots including Safety Aspects*. International Conference on Advanced Robotics, Munich, Germany, pp. 1-6, 2009.

[60] W. Xi & J. S. Baras. *MPC Based Motion Control of Car-like Vehicle Swarms*. Mediterranean Conference on Control & Automation, Athens, Greece, pp. 1-6, 2007.

[61] F. Xiao, L. Wang, J. Chen & Y. Gao. *Finite-time formation control for multi-agent systems*. Automata, Volume 45, pp. 2605-2611, 2009.

[62] F. Xie & R. Fierro. *First-State Contractive Model Predictive Control of Nonholonomic Mobile Robots*. American Control Conference, Washington, USA, pp. 3494-3499, 2008.

[63] F. Xie & R. Fierro. *On Motion Coordination of Multiple Vehicles with Nonholonomic Constraints*. Proceedings of the American Control Conference, New York City, USA, pp. 1888-1893, 2007.

[64] Y. Zhu & Ü. Özgüner. *Constrained Model Predictive Control for Nonholonomic Vehicle Regulation Problem*. Proceedings of the 17th IFAC, Korea, pp. 9552-9557, 2008.