Obstacle avoidance using limit cycles

A. Aalbers

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Obstacle avoidance using limit cycles

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

A. Aalbers

October 10, 2013

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Recent years have shown an increased interest in the use of autonomous vehicles. These vehicles must find a path towards a desired location while avoiding obstacles. Since finding the optimal solution is computationally expensive, approximation techniques are often used. In the last decades several path planning techniques have been proposed.

One novel technique considers mobile robots and uses the limit cycle strategy to avoid static obstacles (D. Kim and J. Kim, 2003). The main advantage of this method is the low computational cost, which makes this method especially useful for small robots with limited computational power. In short, the method relies on creating a circular limit cycle that a robot should follow in the proximity of an obstacle to avoid collision.

In this thesis a new algorithm incorporating ellipsoidal limit cycles is presented for avoiding static 3D obstacles by using 3D limit cycles. The ellipsoidal limit cycles represent a safety zone around an obstacle. By using ellipsoidal limit cycles, the shape of the obstacles can be better represented. Ellipsoidal limit cycles are also useful when there is a preferred direction in which the obstacle should be avoided, e.g., to prevent passing in front of the obstacle.

Further, the limit cycle method is combined with the velocity obstacle approach in order to avoid moving obstacles in 2D and 3D. To ensure that the robot chooses the optimal rotation direction with multiple and/or moving obstacles, a tree search is performed yielding the globally (resolution) optimal rotation direction around each obstacle.

Table of Contents

List of Figures

List of Tables

Preface

This document is part of my Master of Science graduation thesis. In the fall of 2012, I was finishing the necessary courses and started to look for an interesting graduation project. One of the available thesis proposals was to investigate collision avoidance using limit cycles. This awoke my interest because of its practical application, and because I had worked on motion planning before in the final project of my bachelor's study. I discussed the thesis proposal with dr.ir. Mernout Burger, and decided to choose this topic for my graduation project. I have been working on this project since then. And now, almost a year later, the project is finished and the results are written down in this thesis.

Acknowledgements
I am thankful to Mernout, who helped me to initiate the project and who, as a MSc-thesis-advisor, helped me during the first part of my project work. Around July 2012, Mernout left *DCSC*, and dr.ir. Anna Sadowska took over as my thesis advisor. I thank Anna for her support during the final stages of my project work, and especially for her assistance with writing this thesis. I am grateful to prof.dr.ir. Bart De Schutter for his feedback when writing this thesis, and for being the chairman of the examining committee. I also thank dr.ir. Manuel Mazo and dr. Raffaella Carloni for being part of the graduation committee. Finally I thank all other people who helped during this graduation project.

Delft, University of Technology — A. Aalbers
October 10, 2013

"The question of whether computers can think is like the question of whether submarines can swim."

— *Edsger W. Dijkstra*

Chapter 1

Introduction

1-1 Historical context of robot motion planning

Robots control much of our lives. Although most robots are not visible in our everyday lives, they are abundantly present and we could not live the way we do without them. Robots manufacture most of our goods, feed and milk our cattle, handle much of our transportation, and are our friends or foes in computer games. All these robots can move, either in reality or virtually in games and simulators.

The motion planning problem is to find a collision-free path that connects an initial and final configuration of a robot. This may seem to be a simple problem, but it is not. The computational complexity of the best known complete (see Section 1-2-2) path planning algorithms grows exponentially with the number of degrees of freedom of the robot.

The earliest work on robot motion planning was done in the late sixties and early seventies [51]. This research dealt with high level planning using symbolic reasoning that was popular at that time within the Artificial Intelligence (AI) community. Geometry was not often explicitly considered in early robot planners, in part because it was not clear how to represent geometric constraints in a computationally plausible manner.

The introduction of the configuration space in the early eighties initiated a surge in path planning research. Geometric path planning methods were introduced, such as the exact and approximate cell decomposition methods (see Section 2-2). These methods were too computationally expensive to be used in most practical cases, which led to the development of potential field approaches.

Potential field methods became very popular and are still widely used today. To escape the local minima problem of potential fields, probabilistic methods were introduced (see Section 2-3). Probabilistic methods are increasingly popular because of their low computational costs and their completeness.

A more recent method uses limit cycles to avoid obstacles [29]. This method was first used in the context of robot soccer. The major advantages of this method are the extremely low

computational costs and the smooth trajectories that are generated by using this method. These properties make this method especially useful for autonomous vehicles that have limited computational power, such as small unmanned aerial vehicles and other small robots. On the other hand, the limitations of the method are that so far it only worked in two dimensions and with static obstacles. In this thesis a new algorithm is presented that uses limit cycles to avoid obstacles in 3D. The Limit Cycle Method is also combined with the velocity obstacle approach in order to avoid multiple moving obstacles in 2D and 3D.

1-2 Properties of the motion planning problem

Today many different motion planning algorithms exist and they can be characterized by their problem description and algorithm properties. To this end some basic concepts and properties are discussed in the following sections. The terminology used in this report is based on [20, 34, 35]. This report is focused on local collision avoidance, which is a subproblem of the motion planning problem, and uses the same terminology. In what follows, the problem space of the motion planning problem is described (Section 1-2-1). Then, several properties of motion planning algorithms are discussed (Section 1-2-2). The problem statement and contributions are given in Section 1-3 and Section 1-4, respectively.

1-2-1 The problem space

All robots, obstacles and targets are confined in the workspace $\mathcal{W}$, also called world. In this thesis *robots* are objects that move towards a target. They can be described using polygonal (2-Dimensional) or polyhedral (3-Dimensional) or algebraic models [24, 34]. The configuration q of a robot is a set of parameters that defines the position of every point of the robot. The minimum number of configuration parameters is called the degrees of freedom (DOF). The total set of all configurations is called the configuration space, $\mathcal{C}$. For example: a robot that moves on a plane can translate in two directions and rotate in one direction, $\mathcal{C} = \mathbb{R}^2 \times S^1$, also denoted as $SE(2)$, where $\mathbb{R}^2$ represents the Cartesian plane, and S^1 represents the unit circle.

If vehicle dynamics and kinematics are taken into account, it does not suffice to only define the configuration. In these cases the state x is used, which consists of the configuration together with first or higher order time-derivatives of the configuration. The set of all possible states is called the state space.

The workspace $\mathcal{W}$ is divided into two disjoint subsets: the *obstacle space* and the *free space* [20]. The obstacle space is created by all obstacles combined, in which obstacles are understood as bodies in $\mathcal{W}$ that cannot be crossed by the robot. When vehicle constraints are taken into consideration, the obstacle space includes the region of inevitable collision. The free space is the subset of $\mathcal{W}$ in which the robot can move freely without touching any obstacles. Consequently, the task of the motion planning algorithm is to drive the robot from the initial configuration q_{init} to the goal configuration q_{goal} while avoiding obstacles.

1-2-2 Algorithm properties

In this section we discuss some of the properties of algorithms for solving the path planning problem, which refers to the control problem of finding a collision-free path. A trajectory is a path that includes the time parametrization along the path, where each point is assigned to a time instant at which the robot assumes the configuration associated with that point. Generating trajectories is called motion planning, tracking trajectories is called trajectory tracking [8, 20]. In some literature the term motion planning is used for either path planning or trajectory planning [24], however, in this thesis the term motion planning refers to trajectory generating.

Two important properties of motion planning algorithms are the *completeness* (exact or heuristic) and the *scope* (global or local) of the algorithm. Exact algorithms find a path from start to goal if and only if such a path exists, and otherwise prove that there is no solution. Two other, weaker forms of completeness are *probabilistic completeness* and *resolution completeness*. An algorithm is called probabilistically complete if the probability of finding a path approaches one (100%), while an algorithm is considered resolution complete if it finds an existing solution in finite time by choosing smaller spatial discretization steps. Exact algorithms are often *computationally expensive*; this means that it requires a lot of computation power to find a solution. In contrast, heuristic algorithms try to find a solution in a short time, but there is no guarantee for finding a solution; hence they are not complete.

An algorithm is *optimal* if it computes the optimal path with respect to some criterion, e.g., minimal time or minimal distance. *Probabilistic optimality* and *resolution optimality* are similarly defined as *probabilistic completeness* and *resolution completeness*. Optimality implies exactness. Algorithms can only be optimal if they work on a global scope. Global algorithms assume knowledge of the complete workspace, and compute a path or trajectory from start to goal. On the other hand, local algorithms react to obstacles in the direct vicinity of the robot and are in principle never optimal. Local algorithms are often incorporated in global algorithms for emergency obstacle avoidance, when the obstacle appears unexpectedly.

Algorithm complexity

A perfect algorithm would compute an optimal trajectory that satisfies the kinematic and dynamic constraints of the vehicle. Such an algorithm does not exist because of the complexity of the motion planning problem. Even the basic path planning problem is PSPACE hard [24, 45]. This means that in practice it is impossible to find a solution. In some special cases, such as a limited number of obstacles in a planar world, the optimal solution can be found in polynomial time [32]. However, these algorithms usually cope badly with a changing and uncertain workspace model. Due to these complexities, most algorithms are approximate algorithms, which comes at the cost of losing completeness and optimality. Since the complexity increases rapidly with the dimension of $\mathcal{W}$ and $\mathcal{C}$, most algorithms are limited to only a few dimensions. If the motion planning algorithm takes the constraints into account, the complexity is typically very high [20, 24].

Constraints of the robot

The motion of a robot is restricted by dynamic and kinematic constraints. These constraints are also called differential constraints [36], and they restrict the allowable velocities at each

point in $\mathcal{C}$. Kinematic constraints arise from the geometry of the robot, and they are bounds on the velocity. If these constraints can be integrated into position constraints, they are called holonomic constraints. Holonomic constraints constrain the motion of a system away from a certain region of its configuration space. If velocity constraints cannot be integrated, they are called non-holonomic constraints [41, 44], meaning that the time derivatives of the state variables cannot be removed by integration. Nonholonomic constraints arise mostly from underactuated systems, which means that the number of control variables is less than the dimension of $\mathcal{C}$ [36]. The allowable velocity state vector $\dot{q}$ is a function of the current state and control action u,

$$\dot{q} = f(q, u). \tag{1-1}$$

A typical example of a vehicle with nonholonomic constraints is a car, which cannot move sideways but can still reach every configuration in the workspace by maneuvering. Another example is a falling cat, when dropped from an upside down configuration it is able to land on its feet using a combination of maneuvers [40]. In motion planning these constraints often imply that a robot has a minimum turning radius.

Dynamic constraints are typically constraints on the acceleration of the robot. The robot motor has a maximum torque and the robot wheels have a maximum braking force, resulting in a bound on maximal and minimal acceleration. The dynamic constraints can be written as the differential equation

$$\ddot{q} = h(\dot{q}, q, u). \tag{1-2}$$

Using a new state vector x, which incorporates the time-derivative of q, the kinematic and dynamic constraints can be written in a state update equation,

$$\dot{x} = f(x, u). \tag{1-3}$$

Most of the path planning algorithms ignore the kinematic and dynamic constraints of the robot, with the exception of methods like the dynamic window approach and curvature methods [7, 48]. Ignoring the constraints of the robot can lead to intractable paths with sharp edges, that would result in discontinuous motion. These paths are *smoothed* to make them *feasible*, i.e., they are modified such that they satisfy the aforementioned constraints. Some different techniques that smooth out a path are mentioned in [7]. Note that the risk of smoothing is that the new, smooth path might not be collision-free. However, if the constraints are incorporated in the motion planning algorithm, there is no need for smoothing and consequently no risk of collision due to smoothing, which is why most of the recent motion planning algorithms incorporate constraints in the design of the trajectory.

To distinguish between algorithms, three kinds of vehicle models are considered: a point vehicle, a rigid vehicle, and a general vehicle. A point vehicle is a robot with no constraints and a point dimension. A rigid vehicle is a robot of a certain size and shape (often circular or polygonal), but with no kinematic or dynamic constraints. A general vehicle is a vehicle with dimensions and constraints. If an algorithm can handle general vehicles, then it can handle point vehicles and rigid vehicles as well, since these are simplifications of a general vehicle.

Uncertainties and moving obstacles

Although most motion planning algorithms do not consider uncertainties directly, they are

important when computing a collision-free path. There are two types of uncertainty: uncertainty in sensing and uncertainty in predictability [34]. Uncertainties in sensing are due to imperfect sensors and impose uncertainties on the configuration of the robot and on the position of the obstacles. Uncertainties in *configuration predictability* originate from uncertainties in the state update equation (1-3). An algorithm is called *sound* if a robot will reach the goal position and stop there without hitting any obstacles despite uncertainties in sensing and control. If the environment of the robot is dynamic (with moving obstacles), another uncertainty comes from the unknown trajectory of the obstacles; this is called uncertainty in *environment predictability*.

Moving obstacles with an unknown trajectory form a major challenge in motion planning. If the trajectory of the obstacle is known, a collision-free trajectory can be computed by adding a time-dimension to the configuration space and recasting the dynamic motion planning problem into a static motion planning problem [44]. Other approaches include the velocity space and the visibility graph approach [14]. Conversely, if the trajectories of the obstacles are unknown, most motion planners fail to find a collision-free trajectory. Nevertheless, some algorithms can handle these unknown obstacle trajectories, such as the dynamic window approach [15], the genetic-fuzzy approach [43], randomized path planning algorithms [37], and Virtual Force Field algorithms [57]. These algorithm, though, are non-optimal and often not complete.

1-3 Problem statement

In this thesis, we study the motion planning problem with collision avoidance using the Limit Cycle Method. Note that up to now, the Limit Cycle Method only existed for obstacle avoidance with static, two dimensional obstacles. Nevertheless, the computationally inexpensive Limit Cycle Method would be especially useful in 3D, since the complexity of most conventional obstacle avoidance algorithms increases rapidly when applied in higher dimensions. Also, the existing method was unable to handle moving obstacles, which made the Limit Cycle Method not suitable in many practical applications. Therefore, the goal of this graduation project is to extend the existing Limit Cycle Method in a twofold manner. First, an algorithm based on the Limit Cycle Method that can be used in 3D (e.g. for aircraft, submarines) is to be developed. Second, the concept of the Limit Cycle Method is to be enhanced to allow multiple moving obstacles, thus extending the practical application of the method.

1-4 Contributions and outline

In this thesis an overview of several motion planning strategies is given in Chapter 2. The most widely used methods are discussed in Section 2-1 up to and including Section 2-4. The existing Limit Cycle Method for 2D static obstacles is reviewed in more detail in Section 2-5.

The contributions of this graduation project to the existing Limit Cycle Method are:

- The Limit Cycle Method has been extended to 3D.
- The Velocity Obstacle Method is incorporated in the Limit Cycle Method in order to avoid moving obstacles.

- A tree search procedure has been developed that can be used to find the (resolution) optimal rotation direction with multiple, static or moving obstacles.

Based on the Limit Cycle Method, a new algorithm is developed for avoiding static ellipsoidal obstacles in three dimensional space. The details and implementation of this method are discussed in Chapter 3. Special attention is given to the choice of the rotation axis in Section 3-3. The chapter is concluded with a case study.

In order to be able to avoid moving obstacles, the Limit Cycle Method is combined with the Velocity Obstacle Method. This is described in Chapter 4 for the two dimensional case (Section 4-1-2) and for the three dimensional case (Section 4-1-3). To ensure that the robot chooses the globally optimal rotation direction around each moving obstacle, a tree search is performed in order to find the optimal rotation direction. The theory behind this tree search, its implementation, and a case study are discussed in the second part of Chapter 4. Finally some conclusions and recommendations are given in Chapter 5.

Chapter 2

Solutions to the Collision-free Motion Planning Problem

The field of robot motion planning has been well studied, and researchers have come up with a wide variety of motion planning methods. This chapter provides an overview of the main methods found in literature. An extensive textbook that covers the Motion Planning problem is Planning Algorithms [35]. This chapter groups the different methods into their general strategies which are:

- Potential field based methods, covered in Section 2-1
- Sampling based methods, covered in Section 2-2
- Heuristic methods, covered in Section 2-3
- Other methods, covered in Section 2-4
- The Limit Cycle Method, covered in Section 2-5

These methods are not necessarily mutually exclusive, and many combinations of different methods exist. A comparison between the different methods is given in Section 2-6.

2-1 Potential field based methods

This section covers the motion planning methods that are based on Artificial Potential Fields (APF) introduced by Khatib [28] in 1985 and Virtual Force Field (VFF) introduced by Borenstein and Koren in 1989 [4]. APF uses the gradient of potential fields to compute virtual forces on the robot, VFF defines these forces directly using some function. These methods have been widely used in manipulator control and robot motion planning because of their low computational costs, mathematical elegance, and the smooth trajectories that these methods create. They combine trajectory generation and trajectory tracking into one method. The APF and VVF methods are similar in the sense that the gradient descent forces in APF can be considered as the virtual forces in VFF.

The Artificial Potential Fields method

The APF method uses a repulsive potential field around an obstacle to push the robot away from the obstacle, and an attractive potential field at the goal to pull the robot towards the goal [28]. These potential fields are added, which leads to the artificial potential field

$$U_{\text{art}}(x) = U_{x_\text{d}}(x) + U_\text{O}(x), \tag{2-1}$$

where U_{x_d} is the attractive potential to goal position x_d, and U_O is the repulsive potential from the obstacle. In the case of multiple obstacles, their repulsive potentials are added. $U_{\text{art}}(x)$ is then differentiated with respect to the position x, which gives the artificial force

$$F = F_{x_\text{d}} + F_\text{O}, \tag{2-2}$$

where the forces are the gradients of the respective potentials:

$$F_{x_\text{d}} = -\nabla U_{x_\text{d}}(x), \tag{2-3}$$
$$F_\text{O} = -\nabla U_\text{O}(x). \tag{2-4}$$

The attractive potential is usually chosen as

$$U_{x_\text{d}}(x) = \frac{1}{2}k(x - x_\text{d})^2, \tag{2-5}$$

where k is a scaling factor. The repulsive potential $U_\text{O}(x)$ is chosen such that the potential is high close to the obstacle, and small further away from the obstacle, as illustrated in Figure 2-1. The function $U_\text{O}(x)$ can be anything as long as $U_{\text{art}}(x)$ is a positive continuous and

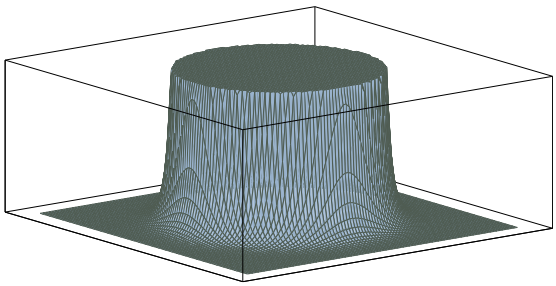


Figure 2-1: Repulsive potential around a circular obstacle.

differentiable function, that attains its minimum at the goal location.
The major disadvantage of the APF method is the possibility to get trapped in a local minimum. This happens especially in environments that are cluttered with obstacles. There are four common conditions in which local minima occur [57]:

1. When an obstacle is located between the robot and the goal, and the centers of the robot, obstacle, and goal are on a line.

2. When the goal is within the active region of an obstacle such that the repulsive force of the obstacle will push the robot away from the goal.

3. When the robot encounters a non-convex (e.g. U-shaped) obstacle.

4. When the robot passes through a narrow passage between two obstacles, this can lead to oscillations.

Some methods based on APFs can overcome the problem of local minima using a range of techniques. For example, a recent method based on VFF called Enhanced Virtual Force Field (EVFF) [57] uses improved functions and detour forces to overcome the local minima problem. The main drawback of these techniques is, however, that while avoiding local minima, they have many heuristic parameters that need to be adjusted for each problem.

2-2 Sampling based methods

Before APF methods existed, sampling based methods were widely used in motion planning. Sampling based methods decompose the workspace into cells (cell decomposition) or a network of lines (roadmap). Each cell or line-segment can be represented by a node in a graph. This reduces the motion planning problem to a graph search problem. Graph search problems can be solved by algorithms such as Dijkstra's algorithm and the A^* algorithm [24]. Other graph search methods are developed specifically for robot motion planning, such as the D++ algorithm [10], which combines Dijkstra's algorithm with local workspace knowledge. The *simplical Dijkstra's* algorithm and A^* algorithms described by Yershov [54] can compute control values at every point in space using interpolation and run at a low computational cost. Sampling based methods can be grouped into one of three categories:

- **Exact cell decomposition.** The workspace is discretized such that the union of the cells covers the free space exactly. The obstacles are represented as convex or non-convex polytopes. Among these methods, we could classify methods relying on the decomposition into convex regions (Meadow Map), decomposition into generalized cones, and vertically or horizontally slicing of the workspace. The major advantage of exact cell decomposition is the low computational costs. On the other hand, the disadvantage of this method is the inability to consider constraints, and its complexity when applied in higher (more than two) dimensions.
- **Approximate cell decomposition.** Approximate cell decomposition uses cells with uniform geometry that fill the workspace. Cells that are in the free space are labeled empty, cells that are completely covered by an obstacle are labeled full, all other cells are labeled mixed. A graph is constructed to search for the shortest path, where the nodes in the graph represent the empty cells, and two nodes are connected if their corresponding cells are adjacent. Common approximate cell decomposition methods are the Regular Occupancy Grid and the Quadtree mapping (illustrated in Figure 2-2). The main advantage of these methods is that they are resolution optimal, and can handle complicated robot shapes and obstacles. Also, they can be applied for any number of dimensions, but at an increasing computational cost. The main disadvantage of these methods is, however, their inability to handle dynamic and kinematic constraints.
- **Roadmaps.** In the roadmap approach, the free configuration space is retracted, reduced to, or mapped onto a network of one-dimensional lines [53]. This approach is also

called retraction, skeleton, or highway approach. The difference with the cell decomposition method is that the cell decomposition approaches decompose the free space into discrete areas, whereas the roadmap approaches decompose the free space into a set of partial possible paths–a roadmap. The motion planning algorithm combines the paths to create a path from robot to goal position. The two most well-known roadmap methods are the Visibility Graph and the Voronoi Diagram. A Visibility Graph creates an optimal path for polygonal obstacles in 2D. It uses the fact that the shortest path grazes polygonal obstacles at their vertices, and builds a roadmap of lines connecting each vertex with all vertices visible from its position [20]. A Visibility Graph can create non-feasible optimal paths (due to constraints), it works only in 2D, requires perfect workspace knowledge, and it can only handle polygonal obstacles. The Voronoi roadmap is the opposite of the Visibility graph in the sense that it keeps the maximum distance to the obstacles. The Voronoi approach builds a roadmap using lines that are equidistant to obstacle edges. The algorithms based on the Voronoi approach are complete, computationally efficient, and can be extended to multiple dimensions, but they are not optimal [1]. Another method, the Velocity Obstacle Method [14], plans a trajectory in the velocity space, and is therefore able to handle moving obstacles with known and constant velocities. Other roadmap methods are the Silhouette method, the Freeway method, and Canny's Roadmap [35, 53].

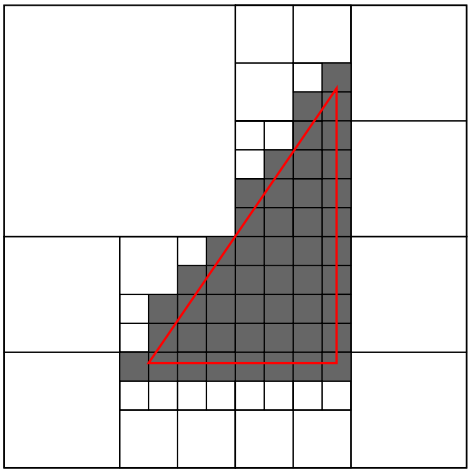


Figure 2-2: Quadtree decomposition with a triangular obstacle. Close to the obstacle the cells are chosen smaller to make it possible for the robot to pass the obstacle at a small distance. This is easily done by splitting up the partially filled cells into four new cells.

Sampling based methods are resolution optimal, but computationally expensive and they do not incorporate dynamic constraints. The computational costs of sampling based methods increase rapidly with the dimensions of the workspace, and therefore some probabilistic methods have been developed. These are covered under heuristic methods in Section 2-3.

2-3 Heuristic methods

To overcome the computational costs and local minima of traditional methods, algorithms have been developed that are probabilistic in nature. In the last decade the vast majority of research done in motion planning has been in heuristic algorithms [53]. Two categories in heuristic methods are: Heuristic algorithms and Metaheuristic algorithms.

Heuristic algorithms use random sampling to create a graph that can be searched. These methods include Rapidly exploring Random Tree (RRT) and Probabilistic Roadmap (PRM). The RRT algorithm was first introduced by Kuffner and Lavalle [31] in 2000, and has been quite popular due to its excellent experimental performance [17]. Instead of sampling the whole free space, only a small random part is sampled, which brings down the computational costs. A naive random tree can be constructed by randomly selecting a vertex from the tree, and a random control input that will add an edge of a certain length to the tree. This will result in a tree that has a strong bias toward places already explored, as shown in Figure 2-3. In contrast, the RRT chooses a random point in the workspace, and tries to connect that point to the closest point in the tree. As a result the tree explores the workspace fast, hence the name of the algorithm. An extension to this algorithm, described in [37], is able to incorporate dynamic and kinematic constraints. The main advantages of most heuristic methods are their low computational costs (for few dimensions), ability to handle constraints, and their probabilistic completeness. However, heuristic algorithms are non-optimal, the choice of the metric (measure of performance, e.g., travel time) is difficult for nonholonomic systems, and they have difficulties in handling changing environments.

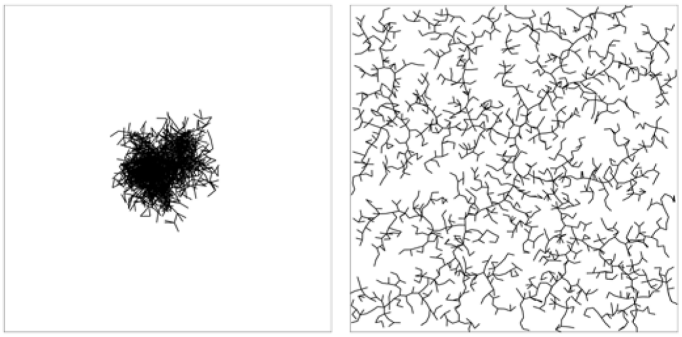


Figure 2-3: Naive random tree (left) versus rapidly exploring random tree (right); each tree has 2000 vertices [37].

Metaheuristic algorithms use random changes to improve the path planning algorithm itself, according to some objective function. These methods include Genetic Algorithms (GA), Neural Networks (NN), Simulated Annealing, Ant Colony Optimization, Stigmergy, Wavelet Theory, Tabu Search, and Fuzzy Logic [52, 53, 58]. Heuristic and metaheuristic algorithms are used not only in motion planning, but in a wide variety of problems that include a Traveling-Salesman-like problem.

Metaheuristic algorithms can be used to tune a motion planner off-line. Any motion planner that has a tuneable variable can be tuned using metaheuristic algorithms. Among these methods are fuzzy-logic controllers [43], neural systems [23], and Virtual Force Fields [30].

Fuzzy logic controllers are increasingly popular in robot motion planning [53], in combination with soft computing techniques such as neural networks, genetic algorithms, and their hybrids. The main advantage of metaheuristic algorithms is that they are tuned off-line and have very low computational costs online. Unlike most motion planning algorithms, most metaheuristic algorithms can handle moving obstacles, since the relative velocity can be used as an input. Most metaheuristic algorithms are used for local obstacle avoidance in combination with a global path planner.

2-4 Other methods

Although most of the motion planning methods can be classified as a variant of one of the methods discussed in the previous sections, there are still motion planning methods that are somewhat different. Some methods draw inspiration from nature such as the bug algorithm, others are derived from language like symbolic planning. Other methods compute trajectories for multiple robots simultaneously, for robots moving in formations, or for multiple robots with different tasks in the same workspace. In this section, some of the main categories that are not (yet) widely used for motion planning are discussed. These methods are:

- **Mathematical methods** Mathematical programming treats the motion planning problem as a constrained optimization problem. Obstacle avoidance is accomplished by representing the obstacles with a set of inequalities on the configuration parameters. The motion planning problem is then formulated as a mathematical optimization problem that finds a path or trajectory from initial to goal configuration. The equations of motion (1-3) can be used as constraints to create feasible trajectories. Mixed Integer Linear Programming (MILP) [46, 55], Discrete Mechanics and Optimal Control (DMOC) [39] and nonlinear programming are some of the popular methods in this field. The main advantage of the MILP method and other constrained optimization methods is the existence of highly-optimized commercially available software that can be used to solve these problems. However, in practice there is an unmodeled sensor uncertainty and prediction uncertainty, which necessitates the use of an additional controller to reduce the impact of the unmodeled effects. Also, the cost functions typically have a number of local minima, thus, whether the global optimum will be found depends highly on the initial guess [20]. Furthermore, the constrained optimization methods are computationally expensive, especially in higher dimensions with many obstacles. The global solution will not necessarily be found in polynomial time, therefore an approximation is often used to solve the optimization problem.
- **Reactive methods** Reactive planners generally only use local knowledge to compute a (partial) trajectory. The sensory information is directly translated to a control action via some transfer function, thus, the robot reacts directly to the environment, hence the name of this method. Reactive planners are seldom used as a sole trajectory generating method, due to their inability to take the global planning problem into consideration. Examples of reactive methods are the Velocity Obstacle Method [14], the Beam Curvature Method [48], The Dynamic Window Approach [15], the Follow The Gap Method [47], and the Limit Cycle Method [29].

- **Bug algorithms** The bug algorithms are one of the earliest and simplest reactive planners [42]. Like the name suggests the algorithm behaves like a bug, with zero range vision (the robot only knows about an obstacle when it actually hits it). Many bug algorithms exist, of which the Bug1 algorithm and Bug2 algorithm are the most well known. The Bug1 algorithm is executed as follows: a robot moves towards the goal. If an obstacle is encountered, the robot turns left around the obstacle; once the obstacle has been encircled completely, the robot returns to the point at which it is closest to the goal and moves towards the goal from there. This repeats until the goal has been reached, at which point the robot stops. Bug algorithms are straightforward to implement, and are complete under the assumptions that the obstacles are either polygons, smooth curves, or some combination of curved and linear parts [35]. However, the bug algorithms are not often used in practice because of the long paths, the inability to incorporate kinematic and dynamic constraints and sensitivity to uncertainties.

- **Symbolic planning** One of the goals of symbolic programming is to create an environment in which the naive user can specify a task for his domestic robot using a high-level language that results in provable correct robot control laws. Symbolic motion planning uses language to specify objectives. Such specifications, consisting of logical and temporal statements, are translated to formulas of temporal logic. A language satisfying such a formula is in general accepted by an *automaton*, which can be seen as a generalization of a graph. A string of words in this language is translated to a finite set of discrete control actions or strategies. Several specification languages have been proposed, such as linear temporal logic (LTL) and computation tree logic (CTL) [2]. Symbolic control can be achieved using *control quanta*, which are discrete control actions that can be combined from a library to move a robot. Another way of symbolic programming is to employ *motion primitives*, which are elementary trajectories that can be combined to create a trajectory towards the goal position. Using such a concept, it was possible to demonstrate completely automated acrobatic flight of miniature helicopters [19]. In real-world applications, the open-loop strategy of motion primitives should be made more robust to uncertainties and disturbances by using feedback control. Discretization based on motion primitives provides a flexible and fast method to create feasible trajectories. However, the main drawbacks are the difficulty in establishing resolution completeness and the challenge to make symbolic planning generally applicable.

2-5 The Limit Cycle Method

The Limit Cycle Method was already mentioned in Section 2-4 as a reactive method. It uses position information to compute a control action via a limit cycle function. Using limit cycles in motion planning was proposed by Kim and Kim in 2001 [29]. They used it in the context of robot soccer to retrieve the ball while avoiding other robots. This method can be used for obstacle avoidance, which is a subproblem of the motion planning problem. In this section we study the limit cycle approach to obstacle avoidance in more detail. We first give a mathematical analysis of limit cycles in Section 2-5-1. Then we describe how limit cycles are currently used in obstacle avoidance in Section 2-5-2.

2-5-1 Theory behind limit cycles

Limit cycles can arise in 2nd-order nonlinear systems and are closed orbits that are stable or unstable. Physically, stable limit cycles represent self sustained oscillations. One of the best known limit cycles stems from the Van der Pol oscillator, which is discussed in Appendix A-1. There are three kinds of limit cycles:

- **Stable limit cycles:** all trajectories in the vicinity of the limit cycle converge to it as $t \to \infty$
- **Unstable limit cycles:** all trajectories in the vicinity of the limit cycle diverge from it as $t \to \infty$
- **Semi-stable limit cycles:** some of the trajectories in the vicinity converge to the limit cycle, while other diverge from it as $t \to \infty$,

where t is the time. From now on only stable limit cycles will be considered, because these stable limit cycles are used in obstacle avoidance. The limit cycles used for motion planning are described by the equations:

$$\begin{aligned} \dot{x}_1 &= \gamma x_2 + x_1(1 - x_1^2 - x_2^2) \\ \dot{x}_2 &= -\gamma x_1 + x_2(1 - x_1^2 - x_2^2). \end{aligned} \tag{2-6}$$

The direction of rotation is given by γ, where

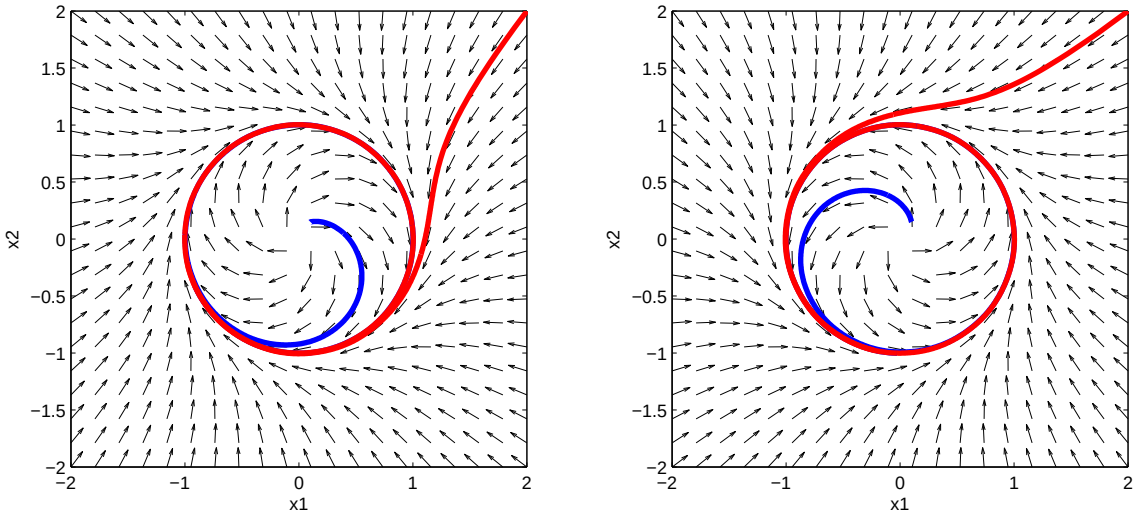
$$\gamma = \begin{cases} 1 & \text{if the trajectories converge towards the limit cycle in a clockwise rotation,} \\ -1 & \text{if the trajectories converge towards the limit cycle in a counterclockwise rotation.} \end{cases} \tag{2-7}$$

The resulting trajectories for $\gamma = 1$ and $\gamma = -1$ can be seen in Figure 2-4. The stability of the limit cycles (2-6) can be shown using the direct method of Lyapunov stability analysis. This is presented in Appendix A-2.

Other limit cycles are found in different research areas, such as the flow equations in a Hopfield model for neural networks [11], and the Wien-bridge oscillator in electrical circuits [13]. A limit cycle can be constructed for any closed planar curve whose polar equation is known [12]. Since many of these curves exist, limit cycles of many different shapes can be constructed and used for obstacle avoidance. In this thesis however, only ellipsoidal limit cycles are considered.

2-5-2 Obstacle avoidance using limit cycles

Limit cycles are used to avoid obstacles in the Limit Cycle Method. The Limit Cycle Method for motion planning as proposed by Kim and Kim [29] considers the following problem. Suppose we have a robot with center position (R_x,R_y) that needs to avoid an obstacle with center position (Q_x,Q_y) to get to a goal position (G_x,G_y), see Figure 2-5. Then, limit cycle algorithm for obstacle avoidance uses the following steps:



(a) Clockwise limit cycle, $\gamma = 1$.

(b) Counter clockwise limit cycle, $\gamma = -1$.

Figure 2-4: Clockwise and counterclockwise limit cycle with trajectories starting from (0.1;0.15) in blue and from (2;2) in red.

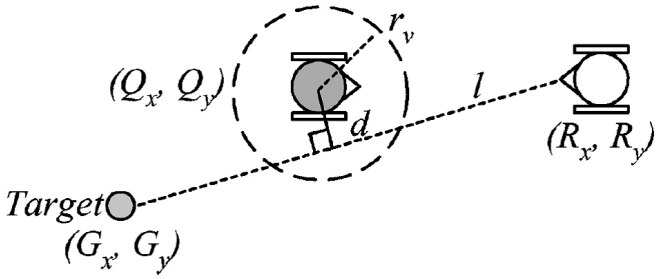


Figure 2-5: Decision of rotational direction based on distance d.

1. Draw a line from the robot to the target in a global coordinate frame $\sum OXY$; the line l is represented by
$$l : ax + by + c = 0. \tag{2-8}$$
2. Treat any obstacle as a disturbing obstacles O_{d} if the line l crosses them, else, threat it as a non-disturbing obstacle O_{n}.
3. Move towards the target if there is no disturbing obstacle O_{d}.
4. Otherwise, follow the limit cycle trajectory that is generated around the closest disturbing obstacle. Repeat steps (1)-(4) until the destination is reached.

To generate the limit cycle trajectory, the direction of rotation around the obstacle must be known. This direction of rotation is based on the distance d. Referring to Figure 2-5, we

calculate the distance d from the center of the nearest disturbing obstacle to the line l using

$$d = \frac{aQ_x + bQ_y + c}{\sqrt{a^2 + b^2}}. \tag{2-9}$$

We then calculate the desired direction of the robot at each position using a generalized version of Equation (2-6),

$$\begin{aligned} \dot{\bar{x}} &= \frac{d}{|d|}\bar{y} + \bar{x}(r_v - \bar{x}^2 - \bar{y}^2) \\ \dot{\bar{y}} &= -\frac{d}{|d|}\bar{x} + \bar{y}(r_v - \bar{x}^2 - \bar{y}^2), \end{aligned} \tag{2-10}$$

where $\bar{x}$ and $\bar{y}$ are the relative position values to the center of the obstacle. The robot avoids the obstacle clockwise if d is positive and counterclockwise if d is negative. The obstacles and robot are assumed to be circular, with radii r_o and r_r, respectively. Using a safety margin δ, the radius r_v is

$$r_v = r_r + r_o + \delta. \tag{2-11}$$

This makes the robot in effect a point vehicle. An example of using this algorithm is given in Appendix A-3.

The algorithm described thus far fails when obstacles are overlapping. When two obstacles overlap, as shown in Figure 2-6, a local minimum occurs and the robot will get stuck at the local minimum. In this case, the two obstacles can be treated as one bigger obstacle and a new central position of the obstacles is calculated. With n overlapping obstacles the new central position with coordinates (Q_x,Q_y) of the overlapping obstacles is calculated by

$$Q_x = \frac{1}{n}\sum_{k=0}^{n} Q_{xk}, \qquad Q_y = \frac{1}{n}\sum_{k=0}^{n} Q_{yk}, \tag{2-12}$$

where (Q_{xk},Q_{xk}) are the coordinates of the kth obstacle. Using the coordinates (Q_x,Q_y) a new value of d is calculated; this determines the rotational direction around all n overlapping obstacles. If the robot does not have global knowledge of the workspace, the robot can get stuck in a loop when avoiding overlapping obstacles [21]. This can be avoided by choosing a fixed direction of rotation until the robot has passed the overlapping obstacles.

Limit cycles are also used in robot soccer to position the robot behind the ball such that the robot faces the opponent goal [29]. The orientation of the robot at the goal position is called *posture* and in most motion planning algorithms it is not possible to control the desired posture directly. The Limit Cycle Method accomplishes posture control by placing two virtual obstacles on a line between the opponent goal and the target position as shown in Figure 2-7.

The dynamic and kinematic constraints can be met for most vehicle models by using the minimum turning radius of the vehicle for the limit cycles. This radius can be velocity dependent [29]. Instead of a fixed goal position, the Limit Cycle Method can be extended to handle moving targets [13, 13]. When all obstacles are cleared using the Limit Cycle Method, some ODEs can be used to track the target [21, 50].

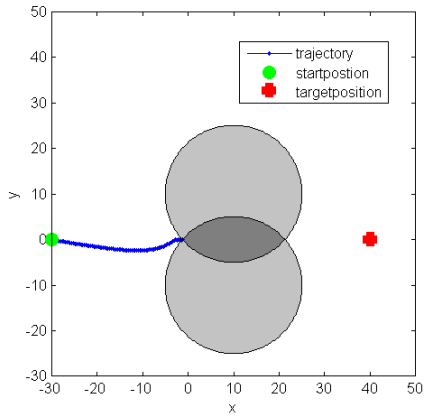


Figure 2-6: Local minimum with two overlapping obstacles.

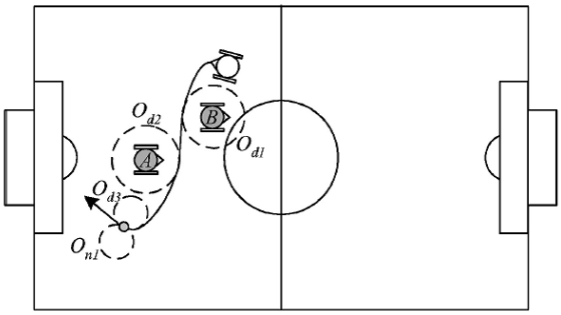


Figure 2-7: Virtual obstacles are placed on either side of the ball to control posture. The two virtual obstacles are placed adjacent to the line from the ball to the opponent's goal. The lower virtual obstacle is associated with a counterclockwise rotating limit cycle, the upper virtual obstacle with clockwise rotating limit cycle. The robot will then reach the ball, facing the opponent's goal [29].

Limit cycles are also used in a multi-agent setting in [18, 22, 56]. When multiple robots converge to a limit cycle with the same relative phase angle, this can be used for surveillance [56]. Local obstacle avoidance using limit cycles in combination with vector fields is described by Jie et al [25].

Ellipsoidal limit cycles

Ellipsoidal limit cycles can be used to better represent oblong obstacles and moving obstacles. The differential equations corresponding to ellipsoidal limit cycles are given by [9, 50]

$$\begin{aligned}\dot{\bar{x}} &= \frac{d}{|d|}\frac{\bar{y}}{b^2} + \alpha\bar{x}(r_v - \frac{\bar{x}^2}{a^2} - \frac{\bar{y}^2}{b^2}),\\ \dot{\bar{y}} &= -\frac{d}{|d|}\frac{\bar{x}}{a^2} + \alpha\bar{y}(r_v - \frac{\bar{x}^2}{a^2} - \frac{\bar{y}^2}{b^2}),\end{aligned} \tag{2-13}$$

where $\bar{x}$ and $\bar{y}$ are the relative positions to the origin of the limit cycle. The speed of convergence is determined by α; a larger α results in a faster convergence to the limit cycle.

The resulting limit cycle is characterized by the general equation of an ellipse, with semi-major axis a and semi-minor axis b:

$$\left[\frac{\bar{x}\cos\phi + \bar{y}\sin\phi}{a}\right]^2 + \left[\frac{-\bar{x}\sin\phi + \bar{y}\cos\phi}{b}\right]^2 = 1, \tag{2-14}$$

where ϕ is the angle between the ellipse's semi-major axis and the global horizontal axis. This angle can be time dependent to represent rotating objects. An example of an ellipsoidal limit cycle is shown in Figure 2-8.

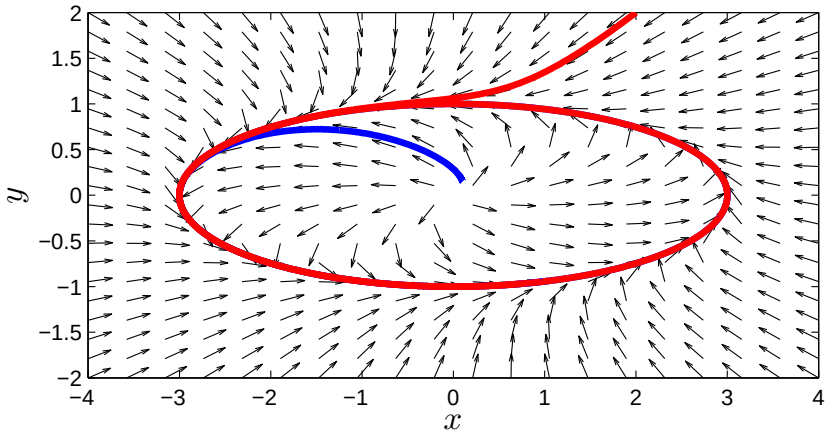


Figure 2-8: An ellipsoidal limit cycle, with semi-axes $a = 3$ and $b = 1$.

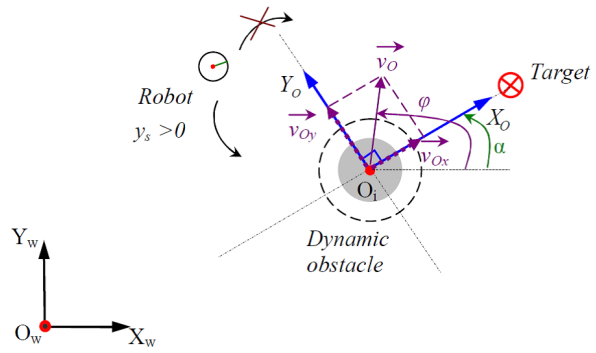


Figure 2-9: Avoiding a moving obstacle [3].

Moving obstacles

The basic limit cycle algorithm can be used to avoid obstacles that move slowly with respect to the maximum velocity of a robot. Because the algorithm updates its trajectory very often, the velocity of slow obstacles can be neglected. The order of avoiding the disturbing obstacles can then be chosen based on the closest-point of approach [38]. This means that the closest obstacle is not chosen based on the current distance to each obstacle, but on the time at which a collision would occur. Ellipsoidal limit cycles can be used with the long axis aligned

with the obstacle's direction [9]. Fast obstacles, however, still pose a problem for the Limit Cycle Method. For instance, if the robot were to pass in front of a fast moving obstacle, it could result in a collision.

A way to solve this problem is to let the robot pass behind the obstacle, such that the robot does not cut off the obstacle's trajectory [3]. The obstacle's velocity v_O can be decomposed into a velocity vector V_{O_x} pointing towards the target and a perpendicular velocity vector V_{O_y}, see Figure 2-9. If V_{O_y} is positive the obstacle will intersect the direct path between the robot and the target, and the robot will avoid the target counterclockwise, thus behind the obstacle. If V_{O_y} is negative, the obstacle will be avoided clockwise.

2-6 Conclusions and comparison

Since an exact solution to the motion planning problem is not computable online in most cases, an exact algorithm for all situations does not exist. The best choice of an algorithm depends on the environment in which the algorithm is used. Each algorithm is designed with a specific goal in mind, and has different priorities on different performance criteria. If the workspace is known with little uncertainty, and the robot has sufficient computational power, a search algorithm like RRT can be used. If the robot has little computational power, a fuzzy controller can be used.

Table 2-1 gives an overview of the advantages and disadvantages of several methods. It is important to note that this table is not conclusive, for many algorithms variants exist that benefit one aspect of the algorithm, often at the cost of something else. For example: the harmonic potential functions are complete, in contrast to standard Artificial Potential Fields methods, but are computationally expensive. A more extensive list of several sub-algorithms that are not discussed in this report can be found in [20].

Method	Main Advantages	Main Disadvantages	Vehicle Model	Obstacle Model
Potential fields methods	inexpensive, smooth trajectories	not complete	general	any
Sampling based methods				
Exact cell decomposition	complete	expensive	rigid	all polytopes
Approximate cell decomposition	resolution complete, sound	expensive	point	any
Road maps	complete	expensive	point	any/all polygons
Heuristic methods				
RRT	inexpensive, probabilistically complete	not optimal	general	any
PRM	probabilistically complete	not-optimal, slow convergence	point	any
GA and NN	inexpensive (off-line training)	not complete	general	any
Other methods				
Mathematical methods	resolution optimal	expensive	general	(convex) polytopes
Reactive methods	inexpensive	not complete	general	any
Bug algorithms	complete	not-optimal, not sound	point	any
Symbolic	inexpensive, feasible	not complete	general	any
Limit cycle method	very inexpensive, smooth trajectories	not complete, only 2D	general	ellipse

Table 2-1: Comparison among motion planning methods

Chapter 3

Avoiding Static Obstacles in Three Dimensional Space

The Limit Cycle Method discussed in Section 2-5 is designed for obstacles in two dimensional space. This method works well for a robot that moves on a plane, e.g., a wheeled robot such as a soccer robot. For robots that do not move on a plane, such as UAVs (Unmanned Aerial Vehicles) or submarines, the method is unsuitable. To make the Limit Cycle Method applicable to robots that move in three dimensions, we develop a new algorithm that uses 3D ellipsoidal limit cycles. In Section 3-1 the 3D limit cycles are discussed and their stability is proven using the direct Lyapunov method. The algorithm for avoiding obstacles in three dimensional space is explained in Section 3-2. In order to use this algorithm, a rotation axis needs to be computed first. Section 3-3 discusses several methods of finding a rotation axis. The algorithm is then illustrated with a case study in Section 3-4.

3-1 Limit cycles in 3D and their stability

The three dimensional shape that is used to describe the obstacles is an ellipsoid. This ellipsoid does not represent the exact shape of the obstacle, but rather a safety zone around the obstacle, where the shape of the ellipsoid can be chosen such as to assign a greater safety distance from certain points on the obstacle, e.g., a greater safety distance at front of the obstacle can be achieved by choosing an oblong ellipsoid. The ellipsoid in the canonical form is characterized by:

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1, \tag{3-1}$$

where a, b, and c represent the lengths of the independent semi-axes of the ellipsoid. In the canonical form, the axes of the ellipsoid are aligned with the coordinate axes and the ellipsoid is centered at the origin. If the lengths of two axes are the same ($a = b$), the ellipsoid is called a spheroid. If the third axis is larger than the other two axes ($a = b < c$), this spheroid is

called a prolate spheroid. If the third axis is smaller than the other two axes ($a = b > c$), the spheroid is called an oblate spheroid. If all three axes are the same, it is a sphere.

A limit cycle for a triaxial ellipsoid in the canonical form can be constructed as follows. In 3D, the stable set towards which the trajectories converge is called an *attractor* instead of a limit cycle. In fact, the term attractor is used for stable sets in any number of dimensions, whereas the limit cycles as described in (2-6) and (2-13) are a special class of attractors, namely attractors in 2D. The proposed attractor is defined with the differential equations:

$$\begin{aligned}
\dot{x} &= -S_3\frac{y}{b^2} + S_2\frac{z}{c^2} + \gamma x\left(1 - \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2 - \left(\frac{z}{c}\right)^2\right), \\
\dot{y} &= \quad S_3\frac{x}{a^2} - S_1\frac{z}{c^2} + \gamma y\left(1 - \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2 - \left(\frac{z}{c}\right)^2\right), \\
\dot{z} &= \underbrace{-S_2\frac{x}{a^2} + S_1\frac{y}{b^2}}_{\text{Part I}} + \underbrace{\gamma z\left(1 - \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2 - \left(\frac{z}{c}\right)^2\right)}_{\text{Part II}},
\end{aligned} \tag{3-2}$$

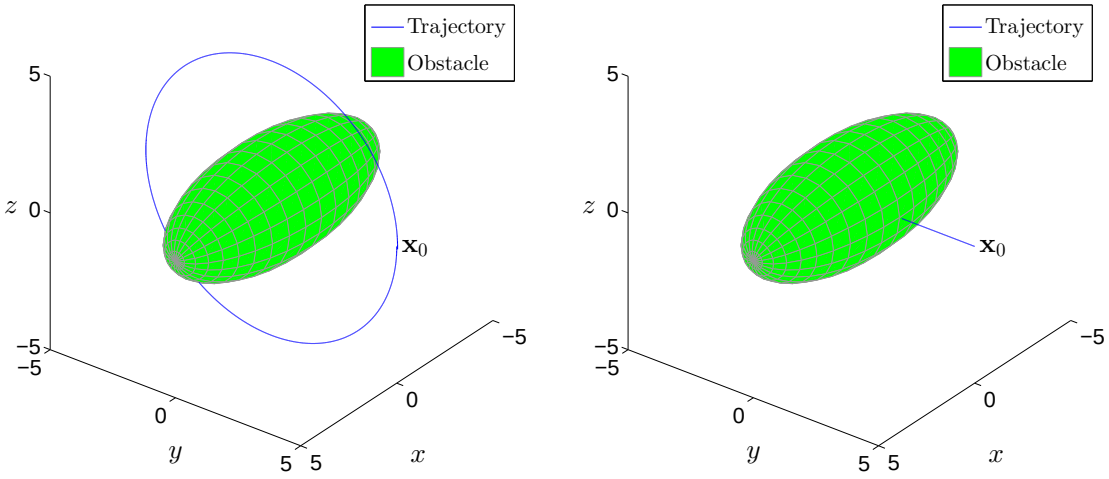
where S_1, S_2, and S_3 are scalars that determine the rotation direction. To gain more insight into these equations, they are split into two parts: Part I represents an elliptic curve part and Part II represents the convergence term. In particular, Part I of (3-2) defines an elliptic curve that lies on a plane. This plane is defined by its normal vector, or *rotation axis*, $A_\mathrm{r} = [S_1, S_2, S_3]^T$ and the center of the ellipsoid. The elliptic curve is a scaled version of the curve that results from the intersection between the plane and the ellipsoid. This curve rotates around A_r. In contrast, Part II of (3-2) makes the trajectories converge towards the attractor. The factor $\gamma > 0$ determines the convergence rate. An illustration of the two parts in these equations can be seen in Figure 3-1. In Figure 3-1a the convergence factor $\gamma = 0$, which leads to an elliptic curve with no convergence to the ellipsoid, in Figure 3-1b, however, γ is chosen very high ($\gamma = 10^6$), which makes the trajectory converge to the surface without encircling the ellipsoid.

All trajectories converge towards the surface of the ellipsoid, which implies that this surface is a stable limit set. The stability of the attractor can be shown using the direct method of Lyapunov stability analysis [16]. The direct Lyapunov method begins with the intuitive notion that one measure of the state of a physical system is the total energy stored in the system. For a stable system, the energy decreases over time and obtains a minimum at a stable equilibrium point $\mathbf{x}_\mathrm{e}$ (e.g., a damped oscillator). Here $\mathbf{x}$ is a vector containing x, y, and z. A scalar function $V(\mathbf{x})$ of the states, called a Lyapunov function candidate, is defined having the following properties [16]:

- $V(\mathbf{x}_\mathrm{e}) = 0$,
- $V(\mathbf{x}) > 0$, $\forall \mathbf{x} \neq \mathbf{x}_\mathrm{e}$,
- V is continuous and has continuous derivatives with respect to all components of $\mathbf{x}$.

The Lyapunov function candidate is considered a Lyapunov function if the time derivative along the trajectories of the system is less than or equal to zero for all states,

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}}\dot{\mathbf{x}} \leq 0. \tag{3-3}$$



(a) Part I of (3-2) results in a elliptic curve around the ellipsoid, starting at $\mathbf{x}_0$. In this plot $A_\mathrm{r} = [1, 0, 0]^T$.

(b) Part II of (3-2) converges the trajectory to the surface of the ellipsoid.

Figure 3-1: The two parts of the ellipsoidal limit cycle equations.

If a Lyapunov function can be found for a system, then the equilibrium is stable. If $V(\mathbf{x}) < 0$ for all $\mathbf{x} \neq \mathbf{x}_\mathrm{e}$ and $V(\mathbf{x})$ is radially unbounded ($||\mathbf{x}|| \to \infty \Rightarrow V(\mathbf{x}) \to \infty$), then the equilibrium is globally asymptotically stable [16], meaning that all trajectories will converge towards the equilibrium point as $t \to \infty$.

For the system (3-2) consider the Lyapunov function candidate

$$V(\mathbf{x}) = \left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2. \tag{3-4}$$

The time derivative of $V(\mathbf{x})$ is

$$\begin{aligned}
\dot{V}(\mathbf{x}) &= 2\frac{x}{a^2}\dot{x} + 2\frac{y}{b^2}\dot{y} + 2\frac{z}{c^2}\dot{z} \\
&= 2\frac{x}{a^2}\left(-S_3\frac{y}{b^2} + S_2\frac{z}{c^2} + \gamma x\left(1 - V(\mathbf{x})\right)\right) \\
&\quad + 2\frac{y}{b^2}\left(S_3\frac{x}{a^2} - S_1\frac{z}{c^2} + \gamma y\left(1 - V(\mathbf{x})\right)\right) \\
&\quad + 2\frac{z}{c^2}\left(-S_2\frac{x}{a^2} + S_1\frac{y}{b^2} + \gamma z\left(1 - V(\mathbf{x})\right)\right) \\
&= \left(2\gamma\left(\frac{x}{a}\right)^2 + 2\gamma\left(\frac{y}{b}\right)^2 + 2\gamma\left(\frac{z}{c}\right)^2\right)\left(1 - V(\mathbf{x})\right) \\
&= 2\gamma V(\mathbf{x})\left(1 - V(\mathbf{x})\right).
\end{aligned} \tag{3-5}$$

The time derivative of $V(\mathbf{x})$ is positive for $V(\mathbf{x}) < 1$ and negative for $V(\mathbf{x}) > 1$. Hence, on the level surface $V(\mathbf{x}) = c_1$, with $0 < c_1 < 1$, all trajectories will be moving outward, while on the level surface $V(\mathbf{x}) = c_2$, with $c_2 > 1$, all trajectories will be moving inward. Similar to

the 2D case in Appendix A-2, this shows that all trajectories converge towards the ellipsoid (3-1). From (3-3) and (3-5) it follows that γ must be a positive number.

3-2 Algorithm for avoiding obstacles

The stable ellipsoidal attractor introduced in the previous section can be used to avoid obstacles in three dimensions with the following algorithm:

1. Draw a line from the robot to the target in a global coordinate frame $\sum OXYZ$; the line l is represented by
$$l : ax + by + cz + d = 0. \tag{3-6}$$
2. Treat any obstacle as a disturbing obstacles O_{d} if the line l crosses them, else, threat it as a non-disturbing obstacle O_{n}.
3. Move towards the target if there is no disturbing obstacle O_{d}.
4. Otherwise, compute a rotation axis, and follow the attractor trajectory that is generated around the closest disturbing obstacle. Repeat steps (1)-(4) until the destination is reached.

There are some differences between this algorithm and the algorithm used in Section 2-5-2. The major difference is that in step 4 the choice for direction of rotation has infinitely many possibilities, whereas the 2D algorithm has the binary choice of clockwise or counterclockwise rotation. Another difference occurs when calculating whether obstacles are disturbing obstacles. In 2D, only the shortest distance d from the line l to the center of the obstacle needs to be computed and compared to the radius of the obstacle to know whether an obstacle is a disturbing obstacle. In 3D, this method works well with spheres, but not with general ellipsoids. To check for disturbing obstacles, the intersection points between the robot's trajectory and the obstacle's surface are computed. In practice, the ellipsoids are often translated and/or rotated with respect to a global coordinate system. Instead of using the canonical form (3-1), it is therefore more convenient to use the equation for an arbitrarily oriented ellipsoid,

$$(\mathbf{x} - \mathbf{p})^T R\Sigma^2 R^T (\mathbf{x} - \mathbf{p}) = 1, \tag{3-7}$$

where $\mathbf{x} = [x, y, z]^T$ represents the position vector, and R is a 3×3 matrix whose columns are orthogonal unit vectors in the directions of the ellipsoid's semi-axes. The matrix Σ is a diagonal matrix, with the entries $1/a$, $1/b$, and $1/c$. The ellipsoid is centered at $\mathbf{p}$. Given the current robot position R_{pos} and the target position T_{pos}, the desired velocity of the robot V_{R} can be calculated as $V_{\mathrm{R}} = (\mathrm{normc}(R_{\mathrm{pos}} - T_{\mathrm{pos}}))R_{\mathrm{speed}}$, where normc stands for the column normalization operation (which is used to make a vector a unit vector), and $R_{\mathrm{speed}} > 0$ is the desired speed of the robot. The speed of the robot is assumed to be constant. Then, the intersection points (if any) can be calculated by using

$$\mathbf{x}_{\mathrm{int}} = R_{\mathrm{pos}} + V_{\mathrm{R}} \cdot t, \tag{3-8}$$

where t is the time of intersection between the robot moving on the line l and the ellipsoid. Substituting $\mathbf{x}_{\mathrm{int}}$ from (3-8) into (3-7) results in

$$(R_{\mathrm{pos}} + V_{\mathrm{R}} \cdot t - \mathbf{p})^T R\Sigma^2 R^T (R_{\mathrm{pos}} + V_{\mathrm{R}} \cdot t - \mathbf{p}) = 1. \tag{3-9}$$

This equation can be solved for t to yield the time(s) of collision between the robot and the obstacle. Note that (3-9) can have two complex solutions, one real solution, or two real solutions. When there are no real solutions for t, the obstacle is a non-disturbing obstacle. When only one real solution for t exists, the line l is a tangent line to the ellipsoid and the robot will only touch, but not intersect, the ellipsoid. Since the ellipsoid represents a safety zone around an obstacle, the robot will not actually touch the obstacle, but it only touches the safety zone. Therefore, in the case of one real solution for t, the obstacle is also regarded as a non-disturbing obstacle. With two real solutions for t, namely t_1 and t_2, where $t_1 < t_2$, there are 3 scenarios:

- $t_1 < 0, \quad t_2 \leq 0$: the robot is already beyond the obstacle, so the obstacle is regarded as non-disturbing obstacle,
- $t_1 < 0, \quad t_2 > 0$: the robot is inside the obstacle, and the obstacle is regarded as a disturbing obstacle. This will not happen in practice, however, because the algorithm will steer the robot around the obstacle,
- $t_1 \geq 0, \quad t_2 > 0$: the obstacle is in between the robot and the target, hence the obstacle is regarded as a disturbing obstacle. Two intersection points A and B are calculated using (3-8).

Importantly, if t_1 is larger than some threshold T, the obstacle is regarded as a non-disturbing obstacle to prevent the robot from avoiding an obstacle that is very far away. This saves computing power and mimics the effect of a sensor with a limited range.

Once the robot encounters a disturbing obstacle, the positions of the obstacle, the robot, and the target are transformed to a local Cartesian coordinate system $\hat{X}\hat{Y}\hat{Z}$ using rotation matrix R^T and position $\mathbf{p}$. In the resulting local coordinate system, the ellipsoid is located at the origin, with the semi-axes aligned with the $\hat{x}, \hat{y}, \hat{z}$ axes. Consequently, the velocity components of the robot can be calculated using (3-2), and translated back to the global coordinate system using R. The components are then scaled to match R_{speed}. However, in order to use equations (3-2), the axis of rotation A_{r} must be calculated first. We elaborate on the choice of the rotation axis A_{r} in the following section.

3-3 Choice of rotation axis

Unlike the 2D algorithm, the 3D algorithm permits infinitely many directions in which to rotate around an obstacle. This gives a great freedom when implementing the algorithm, but it also makes the computation of the rotation direction more involved. One way of choosing the rotation plane is to use the plane that contains the origin of the ellipsoid and the intersection points. This is analogous to the procedure of finding the rotation direction in the 2D algorithm. Although this method is simple and works well for spheres, for ellipsoids this method leads to unnecessary long paths. Another option is to use a fixed rotation direction as will be described in Section 3-3-1, which can lead to unnecessary long paths as well, whereas it is clear that ideally the robot would pass the obstacle using an optimal path. For ellipsoids of revolution (oblate or prolate spheroids), a resolution-optimal path on its surface can be found using tools from geodesy, as described in Section 3-3-2. Since a literature survey did

not bring up any method of finding the shortest path around a general triaxial ellipsoid, only obstacles with a spheroidal shape ($a = b$) are considered in this section.

3-3-1 Using a fixed direction of rotation

The choice of rotation axis depends on the situation in which the obstacle avoidance algorithm is used. In a traffic-like scenario, it might be required to always pass the obstacle in the same direction. For example, it might be desirable to always pass behind the obstacle. The axis of rotation can then be calculated as $A_\mathrm{r} = -R_1 \times V_\mathrm{R}$ where $\times$ denotes the cross product and vector R_1 is the first column of rotation matrix R, which is aligned with the orientation of the obstacle. This example is illustrated in Figure 3-2. In the special case that R_1 is perpendicular to V_R, their cross product is zero and A_r can be chosen as one of the other two columns of R to pass the obstacle around the top or around the side.

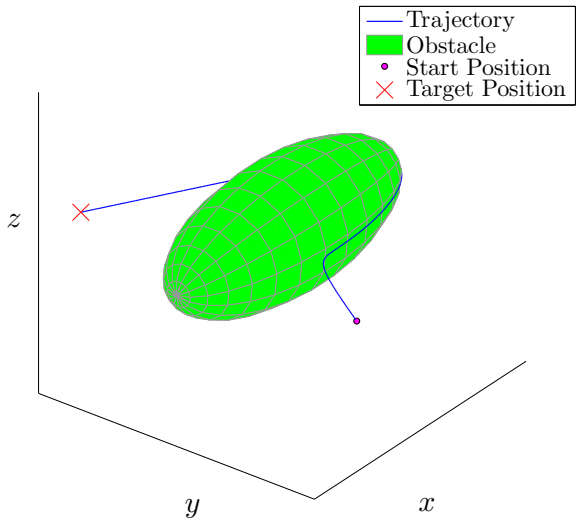


Figure 3-2: The rotation axis A_r is chosen such that the robot passes behind the obstacle, which might not yield the shortest path.

3-3-2 Using the inverse geodetic problem

Another option is to find the resolution optimal path around the surface of an obstacle. The shape of the obstacle is represented by a spheroid. The problem of finding the shortest path around a spheroid is well known in aviation, where it is beneficial to find the shortest path between airports. Since the earth is not a perfect sphere, but an oblate spheroid due to its rotation, finding the shortest path is not trivial. This problem is known as the second geodetic problem or inverse geodetic problem [26]. An approximate solution to this problem can be found numerically, but the accuracy decreases over distance and for increasing eccentricity ϵ, defined as

$$\epsilon = \sqrt{1 - \frac{c^2}{a^2}}. \tag{3-10}$$

On the surface of the spheroid, the resolution optimal path can be computed using the two intersection points described in Section 3-2. When the robot has not yet reached the surface of the obstacle, the path is not optimal due to the convergence term in (3-2). To find the shortest path between the intersection points, the direction of the path at the first intersection point must be calculated.

In geodesy the shortest path between two points on the earth, which is an oblate spheroid, is called a geodesic. The angle at the start position between the north direction and the direction of the geodesic is called the azimuth α, see Figure 3-3. The inverse geodetic problem is to

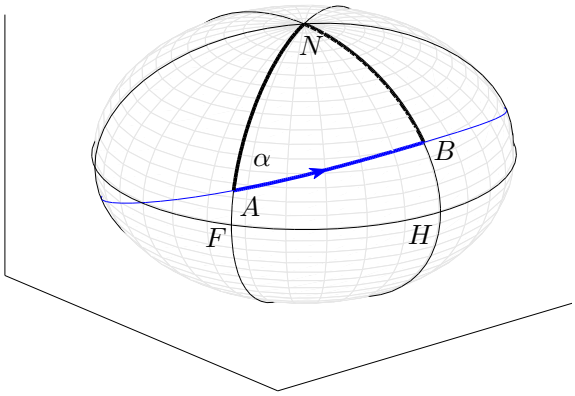


Figure 3-3: Azimuth α is the angle in triangle NAB, A and B are the coordinates of the intersection points, and N is the north pole. NAF and NBH are meridians.

find the azimuth. Given the geographic coordinates (the latitude and the longitude) of the start and end points, the azimuth can be approximated using several different algorithms. In this project the algorithm devised by Karney [26] is used. This algorithm runs at low computational costs and can handle oblate ($a = b > c$) and prolate ($a = b < c$) spheroids. Another advantage of this method is the higher accuracy at increasing eccentricity, compared to standard methods such as Bessel's method [49] (which is used in the standard *azimuth* function in *MATLAB*). The two intersection points calculated in (3-9) are translated first to the local coordinate frame of the obstacle and subsequently to geographic coordinates. Karney's algorithm is then used to find the azimuth.

Once the azimuth has been computed, a direction vector V_{AB} can be obtained. This vector points from the point A in the direction of the geodesic. To determine V_{AB}, first the vector from A to the north pole N is projected onto the tangential plane of the ellipsoid at A, resulting in the vector V_{AN}. This vector is rotated in the tangential plane by an angle equal to the azimuth, which gives the direction vector V_{AB}, as shown in Figure 3-4. The cross product of the direction vector V_{AB} with the direction vector of the robot V_{R} gives the rotation axis A_{r}, see Figure 3-5.

Two observations can be made on the resulting path:

- The computation of the shortest path is based on the two intersection points. The robot generally does not move via these intersection points because the robot starts encircling

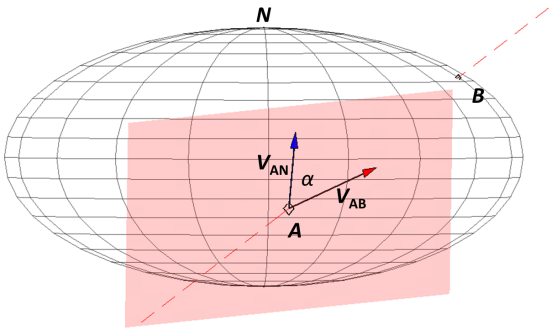


Figure 3-4: In the tangential plane at A, V_{AN} is the unit vector projection of a vector pointing towards N. This vector is rotated α around the normal to the tangential plane, which gives the direction vector V_{AB}.

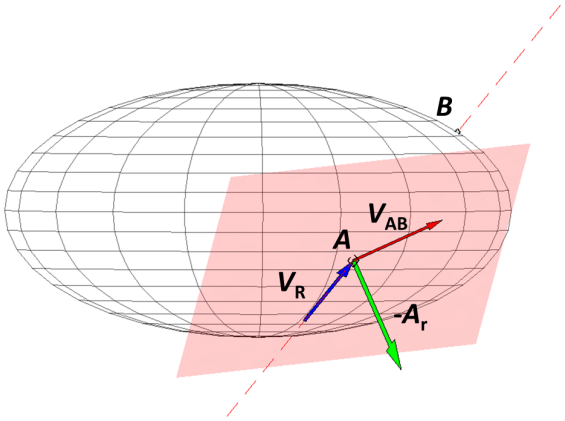


Figure 3-5: The rotation axis A_{r} is computed as the cross product of V_{R} and the vector V_{AB}.

the obstacle when it has not yet reached the surface of the obstacle, and the robot leaves the obstacle before it reaches the second intersection point.

- The convergence factor γ influences the length of the resulting path. A large γ steers the robot quickly towards the obstacle and only when the robot is close to the surface it will start encircling the obstacle, which leads to a very small turning radius close to the ellipsoid. On the other hand, a small γ will make the robot encircle the obstacle, without converging towards the obstacle. If the robot is not yet fully converged to the surface of the obstacle when it leaves the obstacle, the resulting path will be not smooth, as can be seen in Figure 3-8b. In both cases, the resulting path will be longer than when a more appropriate γ is used. An appropriate γ will give a smooth, converging trajectory around the obstacles and must chosen depending on the application of the algorithm and the size of the obstacles. The larger the obstacles are, the smaller the convergence rate needs to be. As a rule of thumb, the convergence rate γ in (3-2) can be chosen as $\gamma = 1/a^2$, where a is the length of the semi-minor axis of the ellipsoid.

The robot will then rotate around the computed rotation axis A_{r} on the elliptic curve defined by Part I in (3-2). In contrast, the convergence term in (3-2) is independent of the rotation

axis and is zero at the surface of the obstacle. When the robot is not yet on the surface of the ellipsoid, its path is not optimal due to the convergence term. Close to the surface of the ellipsoid this term reduces to zero and the rest of the path on the ellipsoid is resolution optimal. The optimality can be increased by decreasing the time steps and by decreasing the error tolerance when approximating α.

Removing detour induced by convergence

The convergence term in (3-2) introduces a detour in the path of the robot when it has not yet reached the surface of the obstacle, as visible in Figure 3-6 and in 2D in Figure 3-8a. In

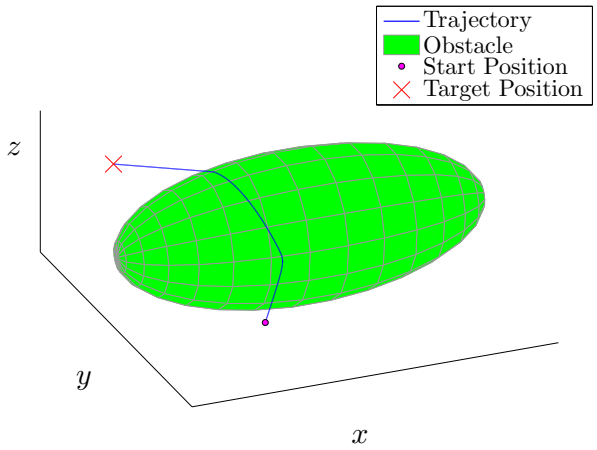


Figure 3-6: The convergence term in (3-2) introduces a detour towards the center of the obstacle. The same start and target positions are used as in Figure 3-2.

fact, the convergence term steers the robot towards a point on the obstacle that has the same geographic coordinates as the geographic coordinates of the current robot position on a scaled version of the ellipsoid. This problem also occurs when using the existing 2D Limit Cycle Method. Since (3-2) does not contain information on the direction of the robot, the robot has no means of converging towards the ellipsoid in the direction of the target position. The detour problem is solved using a different set of differential equations describing the attractor. Specifically, instead of (3-2), we propose to use the differential equations

$$\begin{aligned}
\dot{x} &= -S_3\frac{y}{b^2} + S_2\frac{z}{c^2} + \gamma(x-A_1)\left(1-\left(\frac{x}{a}\right)^2-\left(\frac{y}{b}\right)^2-\left(\frac{z}{c}\right)^2\right),\\
\dot{y} &= \quad S_3\frac{x}{a^2} - S_1\frac{z}{c^2} + \gamma(y-A_2)\left(1-\left(\frac{x}{a}\right)^2-\left(\frac{y}{b}\right)^2-\left(\frac{z}{c}\right)^2\right),\\
\dot{z} &= -S_2\frac{x}{a^2} + S_1\frac{y}{b^2} + \gamma(z-A_3)\left(1-\left(\frac{x}{a}\right)^2-\left(\frac{y}{b}\right)^2-\left(\frac{z}{c}\right)^2\right),
\end{aligned} \tag{3-11}$$

where A_1, A_2, and A_3 are respectively the x, y, and z coordinates of point A, i.e., the first intersection point, which lies on the surface of the ellipsoid. The purpose of using (3-11) is to ensure that the convergence is in the direction of the target in order to create a shorter path.

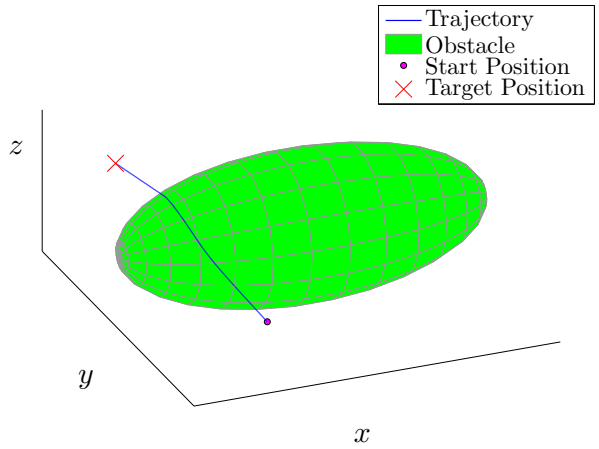


Figure 3-7: The new convergence term in (3-11) creates a shorter path by converging in the direction of the target. In the case illustrated, the travel time is 10.51 seconds versus 11.49 seconds using the naive approach in Figure 3-6.

The intersection point A is used instead of the target position in (3-11) because A is in the same direction as the target and is on the surface of the attractor. The constraint that A is on the surface of the attractor makes (3-11) stable. In order to use (3-11) for obstacle avoidance, these equations must be semi-stable (see Section 2-5-1), i.e., all trajectories starting outside the attractor must converge towards the surface of the attractor; however, trajectories starting inside the attractor do not need to converge towards the surface of the attractor. Since the robot will never end up inside the attractor, the stability for trajectories starting inside the attractor is not taken into consideration. The semi-stability of (3-11) can be shown using the same Lyapunov function candidate as (3-4). The time derivative of $V(\mathbf{x})$ is

$$\begin{aligned}
\dot{V}(\mathbf{x}) &= 2\frac{x}{a^2}\dot{x} + 2\frac{y}{b^2}\dot{y} + 2\frac{z}{c^2}\dot{z} \\
&= 2\frac{x}{a^2}\left(-S_3\frac{y}{b^2} + S_2\frac{z}{c^2} + \gamma(x - A_1)\left(1 - V(\mathbf{x})\right)\right) \\
&\quad + 2\frac{y}{b^2}\left(S_3\frac{x}{a^2} - S_1\frac{z}{c^2} + \gamma(y - A_2)\left(1 - V(\mathbf{x})\right)\right) \\
&\quad + 2\frac{z}{c^2}\left(-S_2\frac{x}{a^2} + S_1\frac{y}{b^2} + \gamma(z - A_3)\left(1 - V(\mathbf{x})\right)\right), \\
&= 2\gamma\left(\frac{x}{a^2}(x - A_1) + \frac{y}{b^2}(y - A_2) + \frac{z}{c^2}(z - A_3)\right)\left(1 - V(\mathbf{x})\right).
\end{aligned} \tag{3-12}$$

The last part $((1-V(\mathbf{x}))$ of (3-12) is smaller than zero for all points outside the attractor, zero at the surface of the attractor, and larger than zero for points starting inside the attractor. All trajectories starting outside the attractor will be converging to the ellipsoid, provided that the first part of (3-12) is larger than zero, i.e.,

$$f(\mathbf{x}) = \frac{x}{a^2}(x - A_1) + \frac{y}{b^2}(y - A_2) + \frac{z}{c^2}(z - A_3) \tag{3-13}$$

must be larger than zero for all $\mathbf{x} \neq \mathbf{x}_e$. The optimum of this function for trajectories starting outside the ellipsoid can be found by solving the constrained optimization problem

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad g(\mathbf{x}) = 1 - \left(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} \right) \leq 0. \end{aligned} \tag{3-14}$$

The minimum can be computed explicitly using Karush-Kuhn-Tucker conditions [6]. This minimum occurs for $\mathbf{x} = A$, where $f(\mathbf{x}) = 0$; a derivation of this minimum is given in Appendix B-1. Hence, on the level surface $V(\mathbf{x}) = c_2$, with $c_2 > 1$, all trajectories will be moving inward. Similarly to the argument in Appendix A-2, this proves that the differential equations (3-11) create a semi-stable attractor shaped as an ellipsoid. Figure 3-7 shows the same situation as Figure 3-6 but now with using (3-11) instead of (3-2).

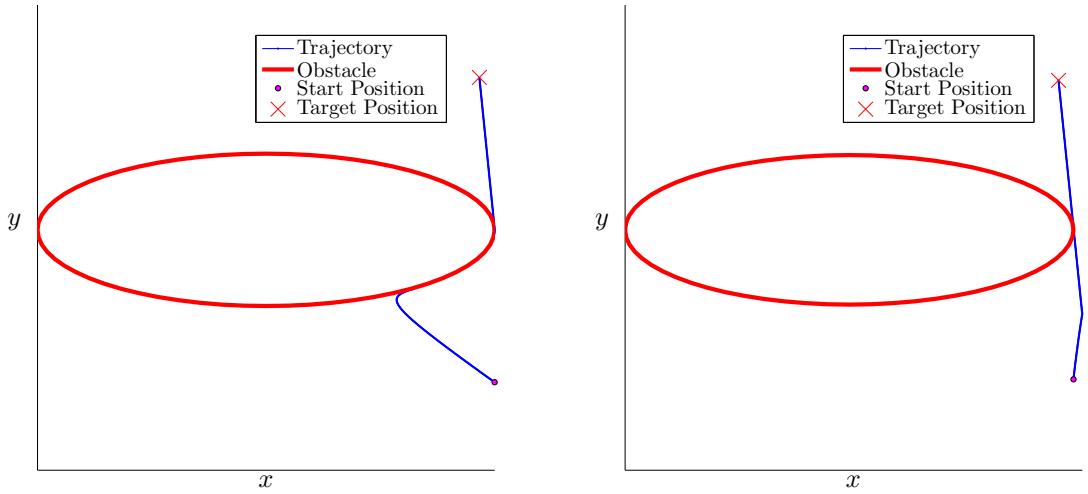
Some remarks on (3-11) are in order.

- By setting z and $\dot{z}$ to zero, (3-11) can also be used in 2D. An illustration of using (3-11) in 2D is shown in Figure 3-8b.
- The convergence terms of (3-2) and (3-11) are similar when the robot is far away from the obstacle, since the coordinates of point A are then small compared to the coordinates of the robot in the local coordinate frame ($R_{\text{pos}} - A \approx R_{\text{pos}}$). Close to the surface of the obstacle, the convergence term in (3-11) (Part II) is smaller than in (3-2), because $R_{\text{pos}} - A$ will approach zero. Part I of (3-2) and (3-11), however, is the same. This means that close to the ellipsoid, the rate of convergence relative to the elliptic curve part (Part I), is higher in (3-2) than in (3-11) when the same γ is used. Therefore, the convergence factor γ needs to be chosen higher in (3-11) than (3-2) to yield a similar convergence rate. When using (3-11) instead of (3-2), even when a higher γ is chosen, the robot is more likely to leave the limit cycle trajectory before being fully converged to the ellipsoid. This can lead to not smooth paths as discussed earlier in this section and shown in Figure 3-8b.
- A small time gain ($< 1\%$) can be obtained by basing the azimuth calculation on the current robot position on a scaled version of the ellipsoid. The other point on this scaled ellipsoid (which is necessary for the azimuth calculation) needs to be computed by solving (3-9) with a scaled Σ. Since these computations impose an extra computational load, and the gain is very small, this approach is not used in this thesis.

3-4 Case study

In this section, the algorithm described in Section 3-2 is used in a numerical example to illustrate avoiding two ellipsoidal obstacles, O_1 and O_2. The positions of the obstacles O_1 and O_2 are respectively

$$Q_1 = \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix} \quad \text{and} \quad Q_2 = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}.$$



(a) The convergence term in (2-6) introduces a detour in the trajectory of the robot.

(b) Using the 2D version of (3-11) results in a shorter path, but this path is more likely to be not smooth.

Figure 3-8: The detour caused by convergence occurs in the existing 2D method as well as in 3D, and can be removed using the differential equations described in (3-11).

The lengths of the semi-axes of the obstacles are the inverses of the entries in diagonals of the matrices Σ_1 and Σ_2, where

$$\Sigma_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \quad \text{and} \quad \Sigma_2 = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix}.$$

The semi-major axis of O_2 is aligned with the z-axis, the orientation of O_1 is arbitrarily picked. The orientations of obstacles O_1 and O_2 are, respectively,

$$R_1 = \begin{bmatrix} 0.35 & -0.57 & 0.74 \\ 0.93 & 0.11 & -0.35 \\ -0.12 & -0.81 & -0.57 \end{bmatrix} \quad \text{and} \quad R_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The robot's start and target position are respectively

$$R_{\text{pos}} = \begin{bmatrix} -2 \\ -4 \\ -3 \end{bmatrix} \quad \text{and} \quad T_{\text{pos}} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}.$$

The obstacles, the start position and the target position are illustrated in Figure 3-9. At the first time step, (3-9) and (3-8) are solved, which give the collision time and collision points respectively. The speed of the robot is 1, which gives a collision time of 1.91 seconds from R_{pos} to O_1 and 6.89 seconds to O_2. The first obstacle, O_1, is the closest disturbing obstacle with the first intersection point A as

$$A = \begin{bmatrix} -1.23 \\ -2.76 \\ -1.76 \end{bmatrix}.$$

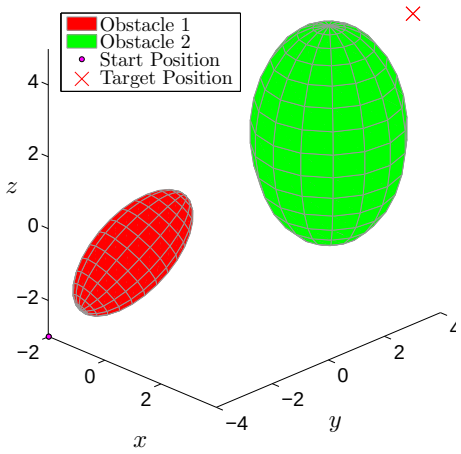


Figure 3-9: Obstacle avoidance scenario.

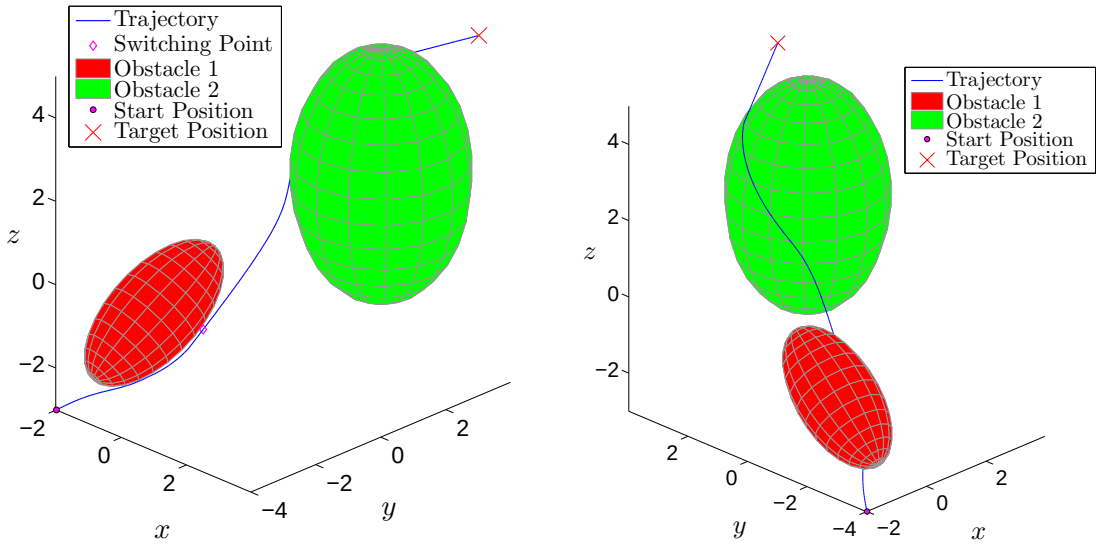
The rotation axis is then determined using the procedure described in Section 3-3-2 as

$$A_{\mathrm{r}} = \begin{bmatrix} -0.79 \\ 0.61 \\ -0.11 \end{bmatrix},$$

in the global coordinate frame. This rotation axis A_{r} together with the intersection point A are then plugged into (3-11) to compute the velocity components for the robot. These velocity components are then scaled to match the speed of the robot.

At each time step the process of computing the closest obstacle, the rotation axis and the velocity components is repeated. After 3.8 seconds, O_1 is no longer a disturbing obstacle and the robot leaves the attractor representing O_1. The line between the current robot position and the target position now intersects O_2, and the robot follows the trajectories generated from an attractor representing O_2 directly after leaving O_1. The switching point from O_1 to O_2 is shown in Figure 3-10a. When O_2 is no longer a disturbing obstacle, the robot moves in a straight line towards the target. The target is reached when the distance from the robot to the target is less than the threshold $T_{th} = R_{\text{speed}}\Delta T$, where ΔT is the time step. With a time step $\Delta T = 0.05$ seconds and a convergence rate $\gamma = 2$ the target is reached after 13.55 seconds. The complete trajectory of the robot from the start position to the target position is shown in Figure 3-10.

The method described above is now compared to the naive approach, i.e. using (3-2) and a rotation axis perpendicular to the plane with the origin and both intersection points (see Section 3-3). As mentioned before, the convergence rate γ in (3-2) and (3-11) should be chosen for each application depending on the size of the obstacles. In order to give a fair comparison between the two approaches, the convergence factor that leads to the shortest path in each of the cases is chosen. For the naive approach the convergence factor $\gamma = 0.4$ and the total traveling time is 13.8 seconds. The resulting trajectory is shown in Figure 3-11. The time savings of the smart approach versus the naive approach are relatively small in this scenario, namely 13.55 seconds versus 13.8 seconds (2%). One reason for this is that the



(a) View of the trajectory around O_1, with the switching point.

(b) View of the trajectory around O_2 and towards the target.

Figure 3-10: Trajectory around the two obstacles.

robot moves along a large portion of the ellipsoid. When the robot only touches the borders of the ellipsoid (as in the situation shown in Figure 3-7), the time savings will be larger due to the lack of a convergence based detour when using (3-11). In the example of Figure 3-7), the time savings is 10.51 seconds versus 11.49 seconds (9%).

3-5 Conclusions

The 2D method for obstacle avoidance using limit cycles has been extended to three dimensions. A simple set of differential equations steers the robot around any ellipsoidal obstacle. These equations contain a rotation axis that must be computed first. Several methods for computing a rotation axis are proposed in this chapter. One naive approach that requires very little computational effort uses as rotation axis the normal to the plane that lies on the origin of the obstacle and the intersection points between the line robot-target and the obstacle. Another approach is to compute the rotation axis using tools from geodesy that compute the optimal direction. On the surface of the ellipsoid, this method results in the resolution optimal path. The convergence term, however, introduces a detour in the trajectories of the robot, both in the existing 2D method as well as in the new 3D method. Consequently, a different set of differential equations is proposed that create a semi-stable attractor such that the convergence part of these differential equations is in the direction of the target position. This solves the detour problem in both the 2D and 3D case.

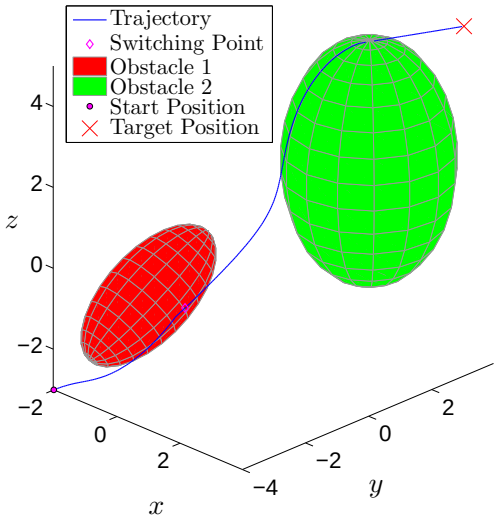


Figure 3-11: Obstacle avoidance in 3D using the naive approach.

Chapter 4

Avoiding Multiple Static and Multiple Moving Obstacles

In the previous chapters of this thesis, all obstacles were assumed to be static, i.e., not moving. In many real word applications, however, obstacles are not static. Moving obstacles can either be other robots, humans, animals, or vehicles. The Limit Cycle Method discussed in Section 2-5 and Chapter 3 is not able to include the velocity of the moving obstacles, which can lead to collision. In order to be able to avoid moving obstacles, we combine the Limit Cycle Method with the Velocity Obstacle Method. In this way, moving obstacles with (different) known velocities can be avoided. A description of what the Velocity Obstacle Method is, how it is used for 2D and 3D obstacle avoidance, and what its limitation are is given in Section 4-1. By introducing the Velocity Obstacle Method, the optimal rotation direction can no longer be computed the way it was done for 2D in Section 2-5-2 and for 3D in Section 3-3. Note that the method can also be used for static obstacles, making it more universal.

Another problem occurs when the robot needs to avoid multiple obstacles: since the robot only considers the closest obstacle at each point, the choice of rotation direction might not be optimal when avoiding multiple obstacles. To overcome both these problems, we propose to perform a tree search to find the optimal rotation direction in 2D and the resolution optimal rotation direction in 3D, as described in Section 4-2. The extended algorithm, with the Velocity Obstacle Method and the tree search included, is illustrated in Section 4-3 with a case study.

4-1 The Velocity Obstacle Method

The Velocity Obstacle Method was introduced by Fiorini in 1993 [14], and is used to avoid moving obstacles, the positions and velocities of which are known. Since the introduction of the Velocity Obstacle Method, several algorithms have been developed that are based on this method, sometimes in combination with other motion planners, e.g., the Non-Linear V-Obstacle method [33].

The Velocity Obstacle Method is described in Section 4-1-1. Its application in 2D and 3D is explained in Section 4-1-2 and Section 4-1-3, respectively.

4-1-1 Theory behind the Velocity Obstacle Method

The Velocity Obstacle Method uses the velocity obstacle (VO) concept, which maps the dynamic environment into the robot velocity space. The velocity obstacle is a first order approximation of the range of robot's velocities that would cause a collision with an obstacle somewhere in the future [14]. The Velocity Obstacle Method then chooses a velocity for the robot that lies outside the velocity obstacle in order to avoid collision. In the original method, a tree of avoidance maneuvers is searched to find the best maneuver according to some objectives. Here, we use the Limit Cycle Method incorporated in the Velocity Obstacle Method to compute a collision-free trajectory of a robot. This is illustrated with an example in the following section.

4-1-2 Application of the Velocity Obstacle Method in 2D obstacle avoidance

Consider a moving obstacle and a robot moving towards a target, as shown in Figure 4-1. The position of the obstacle is O_{pos} and its velocity is V_{O}. The velocity V_{O} of the obstacle, and the speed of the robot are assumed to be constant and known. Also, the speed of the obstacle is lower than the speed of the robot. The relative velocity $V_{\text{R,O}}$ of the robot with respect to the obstacle is computed by subtracting the velocity V_{O} of the obstacle from the velocity V_{R} of the robot, as shown in Figure 4-1. Based on the relative velocity $V_{\text{R,O}}$, it can be

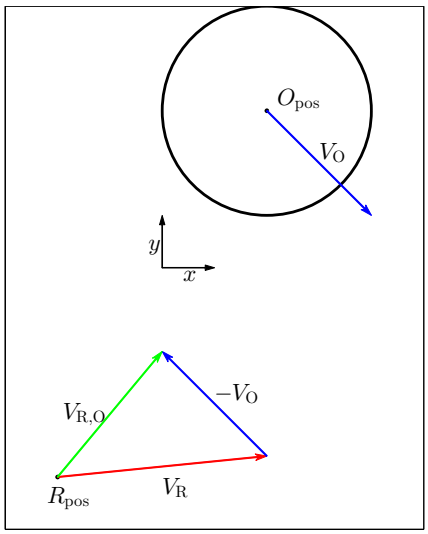


Figure 4-1: The relative velocity $V_{\text{R,O}}$ is computed by subtracting the velocity V_{O} of the obstacle from the velocity V_{R} of the robot, which gives the relative velocity $V_{\text{R,O}}$ of the robot with respect to the obstacle.

determined whether the obstacle is a disturbing obstacle, i.e., whether the robot will collide with the obstacle somewhere in the future. This is done using the velocity coordinate frame of

the obstacle, which is a coordinate frame that moves along with the obstacle, with the center of the obstacle at the origin and the axes aligned with the semi-axes of the obstacle. The transformation between the local coordinate frame $\hat{X}\hat{Y}\hat{Z}$ and the global coordinate frame XYZ is given by

$$\mathbf{x} = R\hat{\mathbf{x}} + O_{\text{pos}}. \tag{4-1}$$

Similarly to (3-9), the time(s) t of a potential collision can be determined by substituting (4-1) in (3-9), resulting in

$$(\hat{R}_{\text{pos}} + \hat{V}_{\text{R,O}} \cdot t)^T R^T R\ \Sigma^2\ RR^T(\hat{R}_{\text{pos}} + \hat{V}_{\text{R,O}} \cdot t) = 1, \tag{4-2}$$

which can be solved for t. Since this equation is applied in the local coordinate frame $\hat{X}\hat{Y}\hat{Z}$, the local rotation matrix $\hat{R} = R^T R$ is the identity matrix and $\hat{R}_{\text{pos}}$ is the position of the robot translated to the local frame. $\hat{V}_{\text{R,O}}$ is the relative velocity of the robot with respect to the obstacle in the local frame, as shown in Figure 4-2a. Note that, similarly to (3-9) in Section 3-2, the collision times can also be computed in the global coordinate frame by extending (3-9), with $\mathbf{p} = O_{\text{pos}} + V_{\text{O}} \cdot t$, to

$$(R_{\text{pos}} + V_{\text{R}} \cdot t - O_{\text{pos}} - V_{\text{O}} \cdot t)^T R\Sigma^2 R^T(R_{\text{pos}} + V_{\text{R}} \cdot t - O_{\text{pos}} - V_{\text{O}} \cdot t) = 1. \tag{4-3}$$

Just like in Section 3-2, (3-8) and (4-3) can have two complex solutions for t, one real solution, or two real solution. With two real solutions for t, namely t_1 and t_2, where $t_1 < t_2$, the obstacle is only considered a disturbing obstacle when $t_1 \geq 0$. The collision points can then be computed using the collision times t_1 and t_2. When using (4-3) to compute the intersection times, the intersection points (in the global frame) can be computed using (3-8); on the other hand, when (4-2) is used, the intersection points are computed as

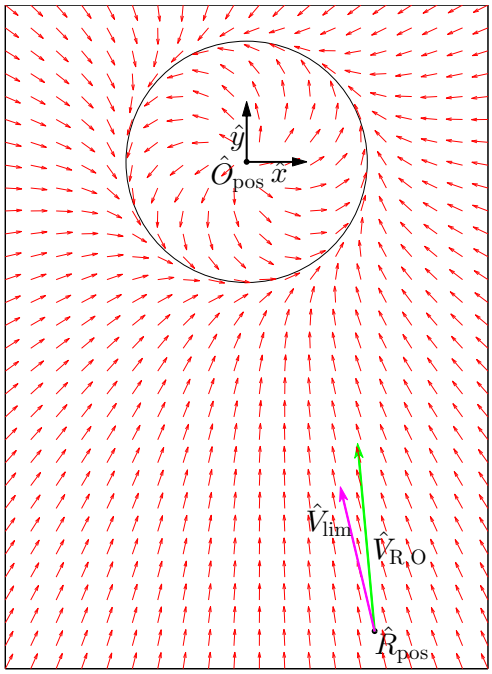
$$\mathbf{x}_{\text{int}} = R \cdot \hat{\mathbf{x}}_{\text{int}} + O_{\text{pos}} + V_{\text{O}} \cdot t, \tag{4-4}$$

where $\hat{\mathbf{x}}_{\text{int}}$ are the intersection points in the local coordinate frame.

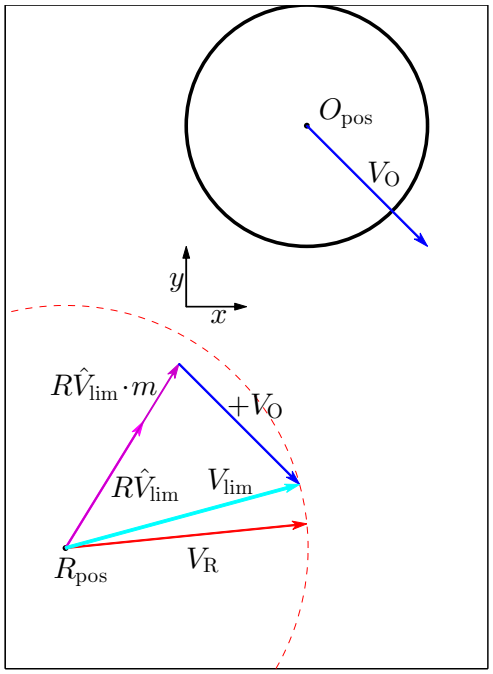
If from (4-2) or (4-3) it follows that the obstacle is a disturbing obstacle, then a limit cycle trajectory is calculated. This is done by placing a limit cycle on the obstacle, in the local coordinate frame, see Figure 4-2a. Based on $\hat{R}_{\text{pos}}$, the desired direction vector $\hat{V}_{\text{lim}}$ in the local frame is computed using the limit cycle equations (2-13) or the 2D version of (3-11). This direction vector then needs to be translated to the global coordinate frame. Since the speed of the robot is assumed to be constant, $\hat{V}_{\text{lim}}$ is scaled such that after translating it to the global frame, its magnitude equals the speed of the robot. Note that the differential equations describing the limit cycle can be scaled by any positive number without affecting the stability of the limit cycle. A positive scaling factor m is determined such that the velocity of the robot in the global frame $V_{\text{lim}} = R\hat{V}_{\text{lim}} \cdot m - V_{\text{O}}$ has magnitude R_{speed}, where m can be computed by solving

$$\|R\hat{V}_{\text{lim}} \cdot m - V_{\text{O}}\| = R_{\text{speed}}. \tag{4-5}$$

Two solutions for m can be obtained by solving (4-5): a positive solution and a negative solution. For the stability of the limit cycle, m must be positive, so the positive solution is chosen. The final desired velocity V_{lim} of the robot is then computed by adding the velocity of the obstacle V_{O} to the rotated, scaled limit cycle direction $R\hat{V}_{\text{lim}} \cdot m$, see Figure 4-2b. An example of using the Limit Cycle Method with moving obstacles is shown in Figure 4-3.



(a) The desired velocity $\hat{V}_{\text{lim}}$ in the local coordinate frame is computed using the coordinates of the robot $\hat{R}_{\text{pos}}$ and the limit cycle equations (2-13).



(b) The final velocity vector V_{lim} is obtained by first scaling and rotating vector $\hat{V}_{\text{lim}}$, yielding the desired velocity $R\hat{V}_{\text{lim}} \cdot m$ of the robot with respect to the obstacle. This vector is then translated to the desired velocity V_{lim} of the robot in the global frame by adding the velocity V_{O} of the obstacle. The red dashed circle represents the range of velocities of the robot (assuming R_{speed} is constant).

Figure 4-2: Computing the desired velocity of the robot with a moving obstacle.

4-1-3 Application of the Velocity Obstacle Method in 3D obstacle avoidance

The Velocity Obstacle Method in 3D is very similar to its application in 2D as discussed in the previous section. Equations (4-2), (4-3), (4-4), and (4-5) are augmented with an extra dimension, and the rest of the procedure of finding the robots velocity V_{lim} is analogous to the method described in Section 4-1-2. The optimal rotation axis A_{r} *in the local coordinate frame* can still be computed using the method described in Chapter 3 and in particular in Section 3-3-2. However, this is not necessarily the optimal rotation direction in the global coordinate frame. This renders the method of computing the resolution optimal rotation direction as described in Section 3-3-2 to be senseless when obstacles are moving with a non-negligible speed. For obstacles that move very slowly compared to the robot, the computed resolution optimal rotation direction (see Section 3-3-2) can still be used since the difference between the optimal direction in the local frame and the global frame is negligible (referring to Figure 4-2b, the difference between $R\hat{V}_{\text{lim}} \cdot m$ and V_{lim} is small when V_{O} is small). Since there are infinitely more rotation direction possibilities in 3D than in 2D, the chance that a rotation axis is optimal in both the local frame as well as in the global frame is very small. In order to find a rotation axis that is closer to the optimal rotation direction, more axes need to be

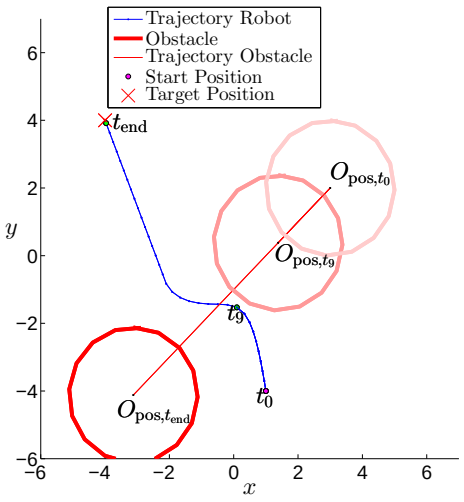


Figure 4-3: Example of using the limit cycle method with a moving obstacle; the obstacle is avoided by a clockwise rotation.

searched, as will be described in Section 4-2.

4-1-4 Limitations of using the Velocity Obstacle Method for obstacle avoidance

The inability of computing the optimal rotation direction in the global coordinate frame, as described in the previous section, is one of the major limitations of using the Velocity Obstacle Method with the Limit Cycle Method. There are several other limitations, such as

The assumption on the velocities of the obstacle. The position of the obstacle is assumed to be known, and its velocity is assumed to be known and constant. Also, the speed of the obstacle must be smaller than the speed of the robot, otherwise (4-5) might not have a solution, i.e., there exists no scaled $R\hat{V}_{\text{lim}}$ such that after adding the velocity of the obstacle V_O the resulting velocity in the global frame V_{lim} has magnitude R_{speed}. When obstacles are moving faster than the robot, the avoidance method described in Section 2-5-2 can be used, which makes the robot always pass behind the obstacle. By choosing a smaller convergence rate γ, the robot will start encircling the obstacle earlier, which can also help to avoid fast moving obstacles more safely. The downside of choosing a smaller convergence rate γ is, however, that the robot might leave the limit cycle trajectory before being fully converged to the limit cycle, which leads to non-smooth, discontinuous paths, as discussed in Section 3-3-2 and shown in Figure 4-4b. Also, when the obstacle is moving faster than the robot and it suddenly changes course, the resulting path of the robot might not be collision-free.

The loss of a minimum turning radius in the global frame. As discussed in Section 2-5-2, the dynamic and kinematic constraints for most robots can be met by using a minimum radius of a circular limit cycle, which will result in a minimum turning radius of the robot. The minimum turning radius of a vehicle corresponds to the smallest circle that this vehicle can drive. In the local velocity frame of the obstacle, the minimum

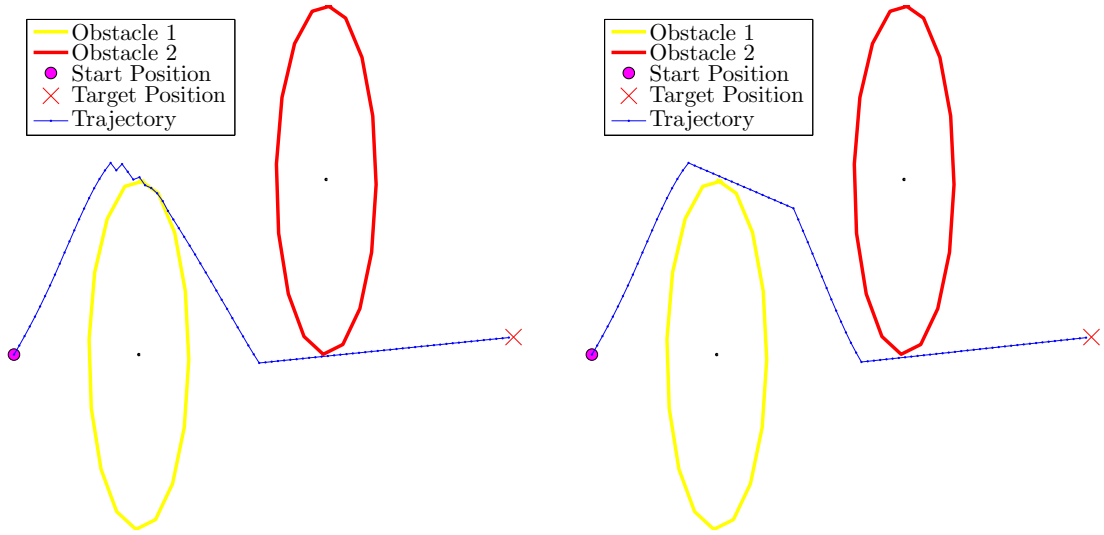
turning radius of the robot is still the same as the radius of the limit cycle, however, when translating the velocity components of the robot to the global coordinate frame, this property is lost. This can be seen in Figure 4-3: the obstacle has a radius of 2, but, the turning radius of the robot at t_9 is less than 2.

The inability to compute the optimal rotation direction in the global frame. The optimal rotation direction in the local frame can be computed. This is similar to how the rotation direction is chosen in Section 2-5-2 and Section 3-3. However, since we now use a local frame that has a velocity V_O, the rotation axis that was computed in the local frame might not lead to the shortest path in the global frame, hence, it is not the optimal rotation direction in the global frame. This problem occurs in 2D as well as in 3D (as discussed for 3D in the previous section).

The risk of switching behavior between obstacles. When the robot needs to avoid multiple obstacles in opposite rotation directions, the robot can start to switch between two obstacles. This is illustrated in Figure 4-4a. When the robot leaves the limit cycle trajectory around Obstacle 1 and starts to follow the limit cycle trajectory around Obstacle 2, Obstacle 1 can become a disturbing obstacle again. The robot will then again follow the limit cycle trajectory around Obstacle 1, and so on. Although the robot will still be able to avoid the obstacles in a collision-free way, this behavior is unwanted because it results in a jagged path which can place high acceleration forces on the robot, and it also complicates the tree search (as will be described in Section 4-2) by adding many more nodes to the tree. The switching behavior can occur in 2D and 3D, for both static and moving obstacles. Although this problem is especially prevalent in situations with multiple moving obstacles, it can also occur with multiple static obstacles (as in the case of Figure 4-4a). This problem can sometimes be solved by choosing a higher convergence factor γ (as in Figure 4-5, where γ is larger than in Figure 4-4a). However, we propose as a solution to the switching problem to adapt the algorithm in Section 2-5-2 and Section 3-2 such that the robot follows the line l towards the target until the closest obstacle is a disturbing obstacle. The difference with the existing algorithms in the previous chapters is, that once the robot leaves the limit cycle trajectory of one obstacle, it does not immediately start to follow the limit cycle trajectory of the next disturbing obstacle, but instead follows l until the next disturbing obstacles is closer by than the previous obstacle. This solution to the switching problem is illustrated in Figure 4-4b.

4-2 Tree search for finding the shortest path

Some of the problems with the Velocity Obstacle Method can be solved by trying several rotation directions, and choosing the one that leads to the shortest path. This will not only solve the problem of computing the globally (resolution) optimal rotation direction for a single moving obstacle, but it will also yield the globally (resolution) optimal rotation directions in the case of multiple obstacles. In the case of 3D obstacles, the globally *resolution optimal* rotation direction is the rotation direction, that is used in the tree search, which leads to the shortest path. The resolution can be increased by using more rotation axes in the tree search.



(a) When multiple obstacles are avoided with different rotation directions, the robot can start to switch between the two obstacles. When Obstacle 1 is no longer a disturbing obstacle, Obstacle 2 is the closest disturbing obstacle and the robot starts to move downwards to avoid Obstacle 2. However, now Obstacle 1 becomes a disturbing obstacle again, and so on.

(b) The switching problem can be solved by letting the robot follow the line towards the target as long as the closest disturbing obstacle is not the closest obstacle in general.

Figure 4-4: Multiple static or moving robots can cause a switching behavior in the robot when avoiding the obstacles, especially when the convergence rate γ is small. In this example the convergence rate γ is chosen very low, to better illustrate the problem.

In 2D, since all rotation directions (clockwise and counterclockwise) around each obstacle can be searched, we obtain a globally *optimal* rotation direction by using the tree search.

Without a tree search, the choice of rotation axis with multiple obstacles might not be optimal, since the algorithm only considers the closest obstacle, as shown in Figure 4-5. In Figure 4-5 the robot decides to go clockwise around Obstacle 1 because the line l towards the target lies above the center of the ellipse. However, by clockwise rotation, Obstacle 2 becomes a disturbing obstacle at a later stage, which would not happen if counterclockwise rotation was chosen. Therefore, the total path turns out longer than when Obstacle 1 would be avoided counterclockwise.

We propose to use a tree search in order to find the (resolution) optimal rotation direction. By using a tree structure, the algorithm does not have to compute the same partial path twice when computing all possible paths with multiple obstacles. Each node in the tree represents the length of the partial path from the moment an obstacle becomes the closest disturbing obstacle to the moment the robot leaves the obstacle. Each node is associated with a certain rotation direction, the number of children of each node depends on the number of rotation directions that is searched. The root node represents the start position, and has an associated length of zero. The end node represents the partial path from the last obstacle to the target position. We first discuss the tree search for the 2D case in Section 4-2-1. Then, we explain

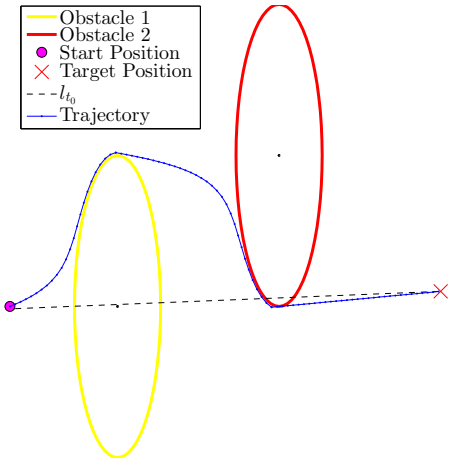


Figure 4-5: Since the line l_{t_0} from the start position to the target position at t_0 lies above the center of Obstacle 1, Obstacle 1 is avoided clockwise, which is not the best choice since Obstacle 2 then becomes a disturbing obstacle and the total path is longer than when Obstacle 1 would be avoided counterclockwise. The path counterclockwise around Obstacle 1 is shown in Figure 4-6a as the path from the start position, via node 2 and node 5, to the target position.

the principles of the tree search in 3D in Section 4-2-2.

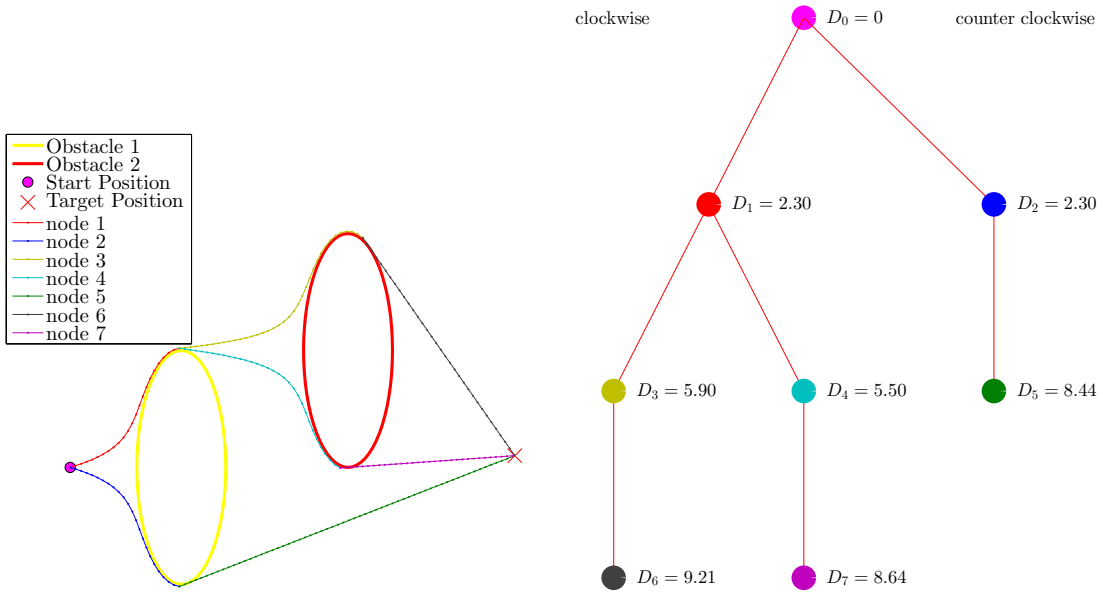
4-2-1 Tree search in 2D

In 2D, there are two possible rotation directions around each obstacle, clockwise and counterclockwise. The maximum number of paths around the obstacles is denoted as $N_{\max}$, and depends only on the number of obstacles. With only two possible rotation directions around each obstacle, it is straightforward to show that

$$N_{\max} = 2^{n_O}, \tag{4-6}$$

where n_O is the number of obstacles. In most cases, the number of paths around the obstacles is fewer than $N_{\max}$, since not every obstacle is a disturbing obstacle in all paths, e.g., Obstacle 2 in Figure 4-6a is not a disturbing obstacle in the path that goes counterclockwise around Obstacle 1.

As shown in Figure 4-6, each partial path around an obstacle is represented by a node in the tree. This tree is created level by level, i.e., in Figure 4-6, first node 1 and node 2 are computed, then node 3 to node 5 are computed, and finally node 6 and node 7 are computed. At each level, the total costs of each node are determined by combining the individual costs of all parent nodes. The tree search is complete when all nodes are end nodes, or when the associated cost of one end node in a certain level is lower than the costs of all other nodes. In the example of Figure 4-6, there four levels: node 0 is in level 0, node 1 and node 2 are in level 1, and so on. At level 2, the cost of end node 5 is still higher than the costs of node 3 and node 4, and therefore, the next level is explored as well. Then, the cost of end node 5 is less than the cost of end node 6 and end node 7, which is why the associated path, counterclockwise around Obstacle 1, is chosen in this example.



(a) All rotation directions are tried in order to find the one that leads to the shortest path.

(b) Building a tree with all possible rotation directions, using as nodes the partial paths. The total accumulated distance at each node i is denoted by D_i.

Figure 4-6: Graph search for rotation direction. Two obstacles must be avoided by the robot to reach the target. The paths around each obstacle, using the two rotation directions, is computed and represented in a graph.

In the example of Figure 4-6, the obstacles are static. In the case of multiple static obstacles, the advantage of the tree search is that the globally optimal rotation direction can be determined by searching all clockwise and counterclockwise paths around every obstacle. When there is one or multiple moving obstacles, the benefit of using the tree search is even greater, since it takes care of the problem of finding the optimal rotation direction with multiple obstacles as well as the problem of finding the optimal rotation direction in the global frame instead of each of the local frames of the obstacles, as discussed in Section 4-1-4.

When all obstacles are either static or moving with a constant, known velocity, the tree search has to be done only once, at the start position of the robot. However, when the positions and velocities of the obstacles are not exactly known, or even changing, the tree search has to be done more often, namely every time one of the obstacles changes course. This requires extra computational power, and the necessary computational power increases exponentially with the number of obstacles, as stated in (4-6). If a tree search is done at each time step, the complexity of the total algorithm can increase by an order of magnitude with only a few obstacles. We show an example of the increase in computational complexity in Section 4-3.

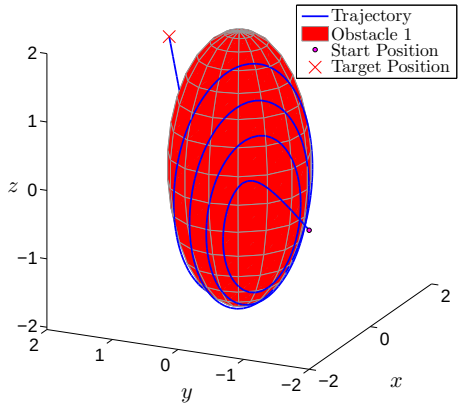


Figure 4-7: An example of a poor choice of a rotation axis leading to a spiraled curve on the surface of the obstacle. Here the rotation axis $A_{\mathrm{r}} = [-1, -1, -0.5]^T$.

4-2-2 Tree search in 3D

A tree search in 3D is different from the tree search in 2D. The main difference comes from building the tree: since there are infinitely many rotation directions, only a subset of all rotation directions can be searched. Therefore, the optimal rotation direction can generally not be computed; instead, the resolution optimal rotation axis can be found, where the resolution can be increased by exploring more rotation axes. Not all rotation directions will steer the robot around the obstacle: some rotation directions will result in a spiraled or closed elliptic curve on the surface of the obstacle, as shown in Figure 4-7. When the rotation axis A_{r} is chosen perpendicular to the surface and in the direction of the target, the robot will only converge to the surface of the ellipsoid and stay there, similarly to the trajectory shown in Figure 3-1b.

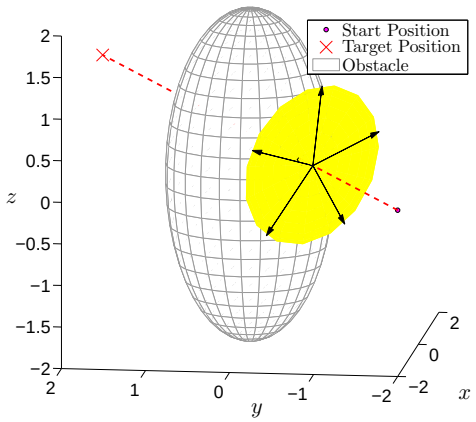
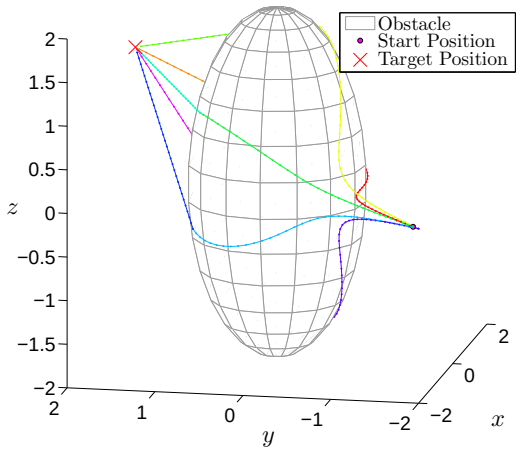


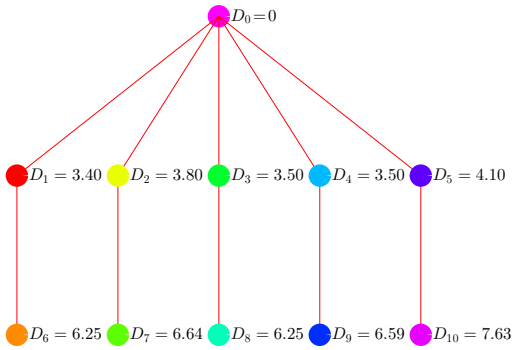
Figure 4-8: Axes perpendicular to the direction of the target are chosen for the tree search, the yellow plane represents a plane perpendicular to the direction of the robot V_{r}. In this case five axes, represented by the arrows on the yellow plane, are chosen for the tree search.

Consequently, when building a tree with different rotation axes, these problematic rotation axes should not be included. Instead, a subset of rotation axes is determined using a procedure similar to the computation of the rotation axis in Section 3-3-2. Recall that in Section 3-3-2 the rotation axis A_r is computed as the cross product of the direction to the target V_r and a rotated vector V_{AB} in the tangential plane at the intersection point. One method of building a tree with different rotation axes would be to rotate V_{AB} using several different angles (azimuths) and computing their corresponding rotation axes A_r. The resulting rotation axes all lie on a plane perpendicular to the desired direction of the robot V_r. Thus, instead of using several different vectors V_{AB}, the rotation axes can be directly computed as vectors lying in a plane perpendicular to the direction vector V_r. This is illustrated with five different, equally spread rotation axes in Figure 4-8.

These five rotation axes are then used to compute their corresponding trajectories using the method described in Section 3-3-2. Similarly to the 2D case, the partial trajectories correspond to nodes in the tree, as shown in Figure 4-9. Note how quickly the three expands



(a) Five rotation directions are tried in order to find the one that leads to the shortest path.



(b) Building a tree with five rotation axes, using as nodes the partial paths. The total accumulated distance at each node i is denoted by D_i.

Figure 4-9: Graph search for rotation axis. One obstacle must be avoided by the robot to reach the target. The paths around the obstacle, using the five rotation axes, are computed and represented in a tree.

compared to the tree in the 2D case: the maximum number of paths N_{max} in 3D is

$$N_{max} = N_{rot}^{n_O}, \tag{4-7}$$

where N_{rot} is the number of rotation axes that is searched.

The obstacle in the example in Figure 4-9 is static; when there are multiple moving obstacles, the procedure of the tree search is exactly the same. In Figure 4-8, the rotation axes are equally spread, i.e., the angle between any two neighboring vectors is $360°/5 = 72°$. The tree search will yield the resolution optimal rotation axis, where the resolution can be increased by searching more rotation axes. For fast moving obstacles, the rotation axes can be chosen such that more axes are searched that lead to trajectories passing behind the obstacle, and

fewer axes that lead to trajectories passing in front of the obstacle. This method is not used in this thesis due to time constraints, but can be considered as a recommendation.

4-3 Case study

The methods described in the previous sections are used in a case study, where a robot must avoid two moving obstacles. The same obstacles as in the example in Section 3-4 are used, but now with different positions and with constant velocities. The start situation is shown in Figure 4-10.

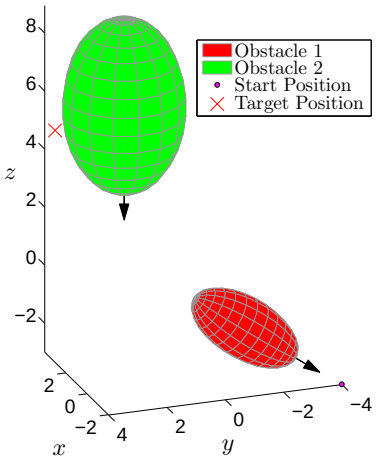


Figure 4-10: Obstacle avoidance scenario, the arrows represent the velocities of the obstacles.

The start positions of the obstacles O_1 and O_2 are, respectively,

$$Q_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad \text{and} \quad Q_2 = \begin{bmatrix} 2 \\ 2 \\ 6 \end{bmatrix}.$$

The lengths of the semi-axes of the obstacles are the inverses of the entries in diagonals of the matrices Σ_1 and Σ_2, where

$$\Sigma_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \quad \text{and} \quad \Sigma_2 = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix}.$$

The semi-major axis of O_2 is aligned with the z-axis, the orientation of O_1 is arbitrarily picked. The orientations of obstacles O_1 and O_2 are respectively

$$R_1 = \begin{bmatrix} 0.35 & -0.57 & 0.74 \\ 0.93 & 0.11 & -0.35 \\ -0.12 & -0.81 & -0.57 \end{bmatrix} \quad \text{and} \quad R_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The robot's start and target position are respectively

$$R_{\text{pos}} = \begin{bmatrix} -2 \\ -4 \\ -3 \end{bmatrix} \quad \text{and} \quad T_{\text{pos}} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}.$$

The velocities of obstacles O_1 and O_2 are in the directions of their semi-major axes. Since the inverses of the last entries in Σ_1 and Σ_2 have the greatest value, they correspond to the semi-major axes of the obstacles which are represented by the last columns in R_1 and R_2 respectively. Each obstacle has a speed of 0.5, corresponding to the velocities

$$V_{O_1} = \begin{bmatrix} -0.06 \\ -0.41 \\ -0.29 \end{bmatrix} \quad \text{and} \quad V_{O_1} = \begin{bmatrix} 0 \\ 0 \\ -0.50 \end{bmatrix}.$$

The velocities of the obstacles do not necessarily have to be in the direction of the semi-major axis of the obstacle, they can be in any direction. In this case, however, the velocities of the obstacles are chosen in the directions of their semi-major axes since most vehicles with oblong shapes move in the direction of their semi-major axis, e.g., submarines and airplanes.

With all the positions and velocities known, at the start position, (4-3) and (3-8) are used to compute the collision time and intersection points with respect to each obstacle. Then, since O_1 is the closest disturbing obstacle, a tree is built to search for the best rotation axis in the tree. First, three different rotation axes are computed, and used to compute their corresponding trajectories around O_1, using (3-11) with $\gamma = 2$. Subsequently, O_2 becomes a disturbing obstacle, and the trajectories around O_2, again using three different rotation axes, are computed. The three trajectories around O_2 are computed from all the end points of the three trajectories around O_1, leading to a total of $3^2 = 9$ trajectories, as shown in Figure 4-11. Note that the robot does not immediately start encircling O_2 once it leaves O_1; this is

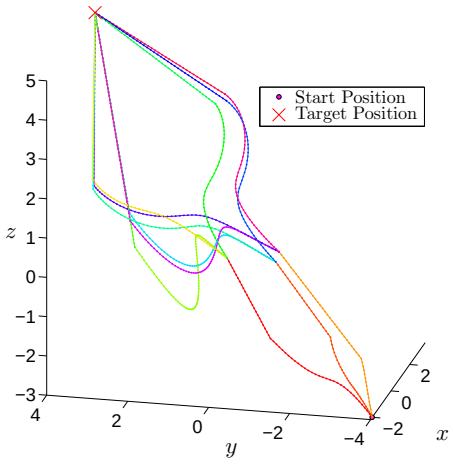


Figure 4-11: All trajectories that are searched are shown as the dotted lines. Three rotation axes are searched for each obstacle; these rotation axes equally spread.

a result of the policy described in Section 4-1-4 to eliminate the switching effect. This policy

can be seen in Figure 4-11 and Figure 4-13, where, once the robot leaves O_1, it first moves in a straight line towards the target before starting to avoid O_2.

The resulting tree with the associated total distances is shown in Figure 4-12. With the speed of the robot $R_{\text{speed}} = 1$, the total distance has the same numeric value as the amount of time it took to cover that distance. The path with the smallest total distance is chosen, which in

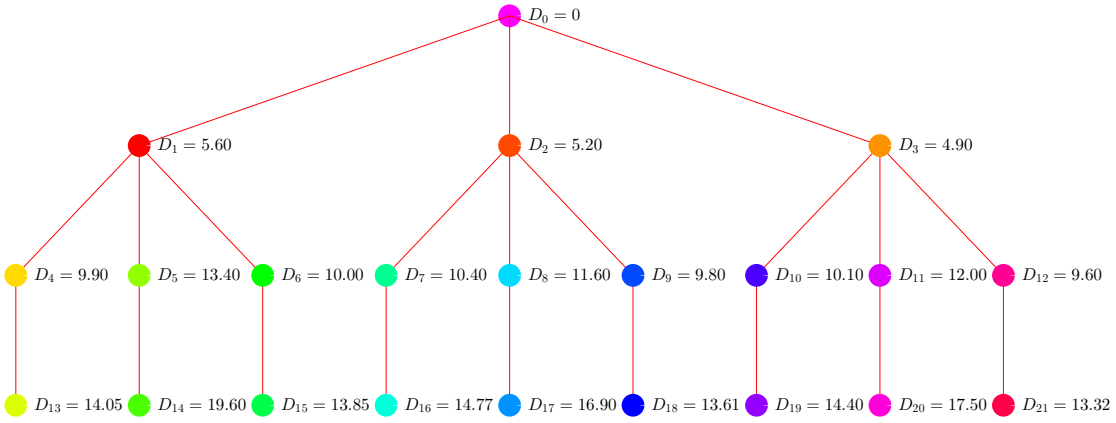


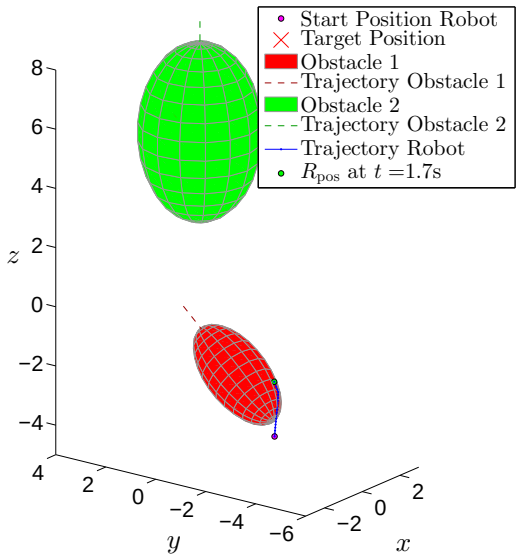
Figure 4-12: The tree of possible paths is built using three rotation axes for each obstacle. The total accumulated distance at each node i is denoted by D_i.

this case is the path via node 0–node 3–node 12–node 21. The corresponding rotation axes for O_1 and O_2 are, respectively,

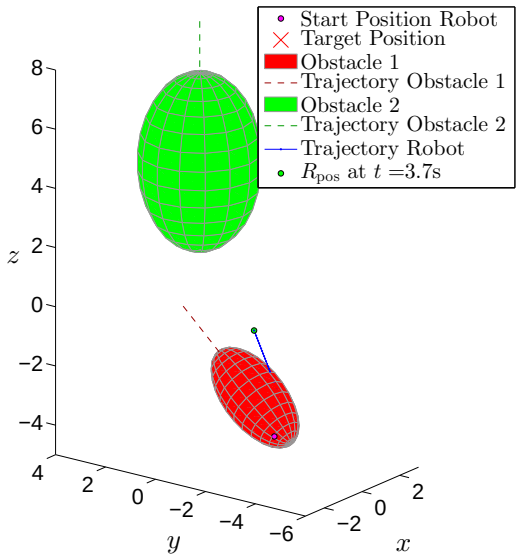
$$A_{r_1} = \begin{bmatrix} 0.91 \\ -0.18 \\ -0.38 \end{bmatrix} \quad \text{and} \quad A_{r_2} = \begin{bmatrix} 0.91 \\ -0.15 \\ -0.39 \end{bmatrix}. \tag{4-8}$$

Now that the rotation axes are known for both obstacles, (3-11) are used to compute the trajectory of the robot. The resulting trajectories are shown in Figure 4-13.

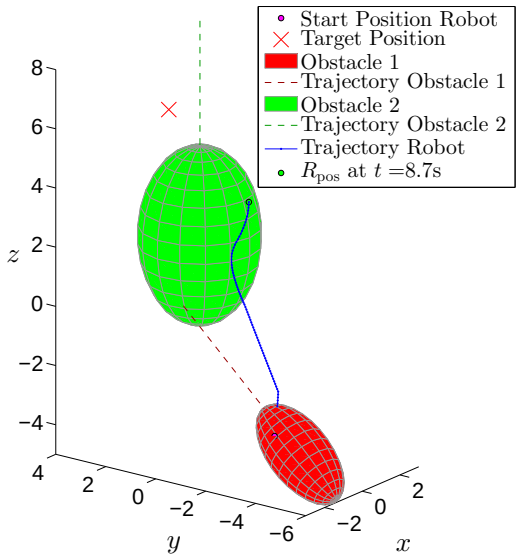
Note that the tree search can be performed either at each time step, or, since the obstacles have constant velocities, only at the start position. If the velocities of the obstacles do not change, both approaches result in exactly the same trajectory. When doing a tree search, the robot computes the proximity and desired velocity for each position on the partial paths in the tree, which imposes quite a high computational load. For the tree search at the start position, the total number of points that is computed is 1084, with a time step of $\Delta T = 0.1s$. Therefore, if a tree search is performed at each time step, it will approximately increase the computational load by three orders of magnitude; however, the computational load will decrease closer the the target, since fewer nodes have to be computed. For the implemented algorithm in *MATLAB* the tree search takes approximately 1 second. However, the algorithm is not optimized for speed, and the computational load can most likely be significantly reduced. In this case study the tree search is only done once, that is, at the start position.



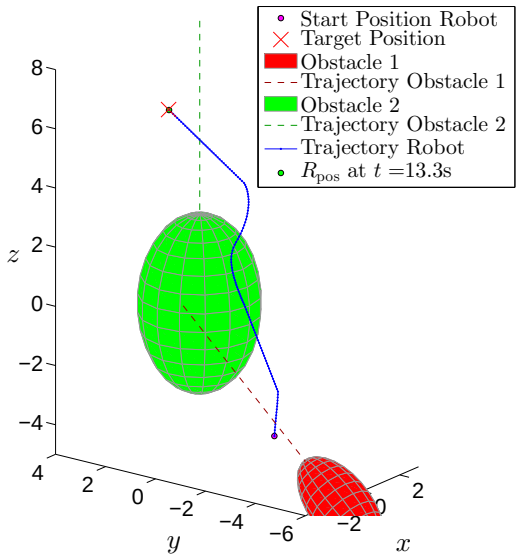
(a) Avoiding the first obstacle using the rotation axis A_{r_1} in (4-8), $t = 1.7$s.



(b) After avoiding O_1, the robot moves in a straight line to the target position, until O_2 becomes the closest obstacle, $t = 3.7$s.



(c) Avoiding O_2 using the rotation axis A_{r_2} in (4-8), $t = 8.7$s.



(d) The robot reached the target position, and the trajectory is complete, $t = 13.3$s.

Figure 4-13: Several stages in the trajectories of the robot and the obstacles.

4-4 Conclusions

We combined the Limit Cycle Method with the Velocity Obstacle Method for avoiding moving obstacles in 2D and 3D. The problem is transformed to a problem where the obstacle is static by introducing a local coordinate frame whose origin moves along with the center of the obstacle, and whose axes are aligned with the semi-axes of the obstacle. In the local coordinate frame, the desired direction can be computed using the methods described in the previous chapters. This direction is then translated to the global coordinate frame to yield the velocity components of the robot. The optimal rotation direction around the obstacles is chosen in the local obstacle frame, which generally is not the optimal rotation direction in the global frame since the resulting path in the global frame might be longer than when a different rotation direction was used. Also, since the robot bases the rotation direction only on the closest obstacle, the rotation direction might not be optimal with multiple (static or moving) obstacles. Both these problems are addressed by performing a tree search, where the paths resulting from several different rotation direction are computed. Using this tree search, the globally optimal rotation directions can be found with multiple moving obstacles in 2D, and the resolution optimal rotation axes with multiple moving obstacles can be found in 3D. This three search, however, imposes an extra computational load. By combining the Velocity Obstacle Method with the Limit Cycle Method, and with the use of a tree search for finding the optimal rotation direction, multiple ellipsoidal obstacles with constant and bounded velocities can be avoided in 2D and 3D.

Chapter 5

Conclusions and Recommendations

With the increase of the number of mobile robot applications in the past decades, robot motion planning has become more and more important. However, in most cases, the problem of finding a collision-free path towards a target position is too computationally complex, and the optimal solution cannot be found. Many algorithms have been developed that find an approximate solution. Some of these algorithms, such as the approximate cell decomposition methods, can find a so-called resolution optimal path (see Section 2-2), i.e., a path closer to the optimal path can be found by decreasing the size of the cells, but these algorithms are often computationally complex. Other algorithms, e.g., most of the the potential field methods, are computationally inexpensive, but in general may not find the optimal solution, and can get stuck in a local minimum before reaching the target. Of all the existing algorithms, there is no perfect algorithm that is suitable in all situations; yet, an appropriate motion planning method must be chosen in each motion planning problem.

The obstacle avoidance problem, which is a subproblem of the motion planning problem, can be solved using one of the techniques discussed in Chapter 2. The recent Limit Cycle Method is one of these obstacle avoidance algorithms that has been used for avoiding obstacles in the context of robot soccer. This method places a limit cycle around the obstacle, and uses the corresponding trajectories of the limit cycle to avoid the obstacle. Once it has passed the obstacle, the robot leaves the limit cycle and moves on to the next obstacle, or, if there are no more obstacles in its path, it moves in a straight line to the target position. The main advantage of the Limit Cycle Method is its simplicity and that it requires very little computational power. However, the existing Limit Cycle Method concerns only the 2D case with static obstacles, limiting the practical applicability of the method.

In this thesis, a new algorithm has been developed that uses the Limit Cycle Method to avoid static obstacles in three dimensional space. Moreover, attractors (which are limit cycles in 3D) with ellipsoidal shapes are used instead of spherical ones. The attractor does not necessarily represent the exact shape of the obstacle; instead, it is used to represent a safety bound around an obstacle. Using ellipsoidal shapes allows for different obstacle shapes to be represented better, and it can also be used to assign larger safety bounds to certain parts of the obstacle, e.g., a larger safety bound can be used in front of the obstacle. A set of

differential equations is proposed that generates a stable ellipsoidal attractor; this attractor is then placed on the obstacle, and the robot follows its trajectory until it has passed the obstacle. The ellipsoid can be avoided by either choosing a fixed rotation direction, or by computing the resolution optimal path around the surface of the ellipsoid. The resolution optimal path around the surface of the ellipsoid is computed using an algorithm that finds an approximate solution to the so-called second geodetic problem, in which the task is to compute the shortest path between two points on the globe. This same method is used to compute the rotation direction on the surface of the ellipsoid.

The existing 2D Limit Cycle Method, as well as the new 3D version, contain a convergence term in the differential equations that are used to compute the robot trajectories. This convergence term, however, introduces a detour in the path of the robot by making the robot's trajectory converge in the direction of the center of the obstacle. Indeed, convergence in the direction of the center of the ellipsoid is unwanted when the robot does not move across the center of the ellipsoid, but instead only passes the ellipsoid on one of its flanks. Thus, a new set of differential equations is proposed that creates a semi-stable attractor, yielding convergence in the direction of the target position. This semi-stable attractor is stable for all trajectories starting outside the ellipsoid, and, since the robot will never end up inside the ellipsoid, this is sufficient to use the attractor for obstacle avoidance. By using this new attractor, the robot no longer converges towards the center of the ellipsoid, which removes the detour in the path of the robot whenever the robot is passing the obstacle on one of its flanks. This approach solves the detour problem in both the 2D and the new 3D Limit Cycle Method.

By incorporating the Limit Cycle Method with the Velocity Obstacle Method, moving obstacles with a constant velocity can be avoided. The Velocity Obstacle Method transforms the problem of avoiding a moving obstacle to a problem where the obstacle is static, by using a local coordinate frame that moves along with the obstacle. However, the globally optimal rotation direction can no longer be calculated directly when using the Velocity Obstacle Method for moving obstacles. This is a result of the fact that the optimal rotation direction is computed in the local coordinate frame, but, when transforming it to the global coordinate frame, its optimality is lost. Also, in the case of multiple static obstacles (both in 2D and 3D), the computed rotation direction is generally not the globally optimal rotation direction. Since the computation of the optimal rotation direction is based solely on the closest obstacle, the computed rotation direction might not lead to the shortest total path when multiple obstacles are present.

For multiple static or moving obstacles, the optimal rotation direction in 2D can be found by searching the two possible rotation directions (clockwise and counterclockwise) around each obstacle in a tree search. A tree is then built where the nodes of the tree represent the partial trajectories around an obstacle, with their corresponding rotation direction. The globally optimal rotation direction is consequently chosen as the one that leads to the shortest path. In 3D, there are infinitely many possible rotation directions, and only a *resolution* optimal rotation direction can be found by using the tree search. The disadvantage of the tree search is that it increases the computational load of the Limit Cycle Method. On the other hand, the lengths of the resulting robot trajectories are reduced by employing the tree search.

These contributions to the Limit Cycle Method result in an algorithm that is able to avoid static and moving obstacles in 2D and 3D.

Recommendations and future work

The Limit Cycle Method introduced in this thesis can be used to avoid moving obstacles. However, the speed of the robot and the velocities of the obstacles are assumed to be constant, and the robot is assumed to be without constraints, i.e., the robot can move in all directions at any given time instant. In future studies, the effects of a variable robot speed can be considered. A variable robot speed can result in not only decreasing the travel time, but it can also be used in such a way as to meet the vehicle constraints, e.g., moving slower when turning.

Another improvement can be made in the tree search for 3D obstacle avoidance. In this thesis, a simple tree search is used to find the rotation axis by searching different, equally spread rotation axes. This tree search can be improved by not limiting the tree search to an equally spread set of rotation axes, but, e.g., searching more rotation axes that lead to trajectories where the robot passes behind the obstacle, and searching fewer rotation axes that lead to trajectories where the robot passes in front of the obstacle. Also, a different search strategy can be investigated instead of the breadth-first strategy that is used in this thesis, for instance, the A* algorithm [54]. These strategies could potentially decrease the computational costs of the tree search by searching the tree more efficiently.

The original Limit Cycle Method has been tested on real robots, i.e., soccer robots. For moving obstacles in 2D and 3D, the method has not yet been tested in experiments. The Limit Cycle Method with moving obstacles in 2D could be tested using wheeled robots, such as the soccer robots. For testing the Limit Cycle Method in 3D, other robots are needed, such as small submarines or aerial vehicles. These robots do not necessarily need to be ellipsoidal; as described in Section 2-5-1, many different shapes can be used as a limit cycle. These other shapes can then be used for obstacle avoidance, but also for manipulator control or for surveillance vehicles. Another area where the Limit Cycle Method might be useful is in computer games; with its low computational costs (in the case of few obstacles) and smooth trajectories, the Limit Cycle Method can be used to quickly compute trajectories of virtual vehicles and thus reduce the computational load of the game. By testing the Limit Cycle Method in one of these applications, its performance in real-life situations with uncertain measurements can be tested, and compared to other motion planning methods.

Appendix A

On 2D limit cycles

A-1 Van der Pol limit cycles

One of the best known limit cycles stems from the Van der Pol oscillator. Van der Pol found stable oscillation in electrical circuits. The 2nd-order system can be described by

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0, \tag{A-1}$$

where x is the position and t is the time and $\dot{x} = \frac{dx}{dt}$. This equation can be seen as a mass-spring-damper model where the damping force depends on the position. This equation can be rewritten with $x_1 = x$ as

$$\begin{aligned} \dot{x}_1 &= \alpha x_2 \\ \dot{x}_2 &= -\alpha x_1 + \mu(1 - x_1^2)x_2, \end{aligned} \tag{A-2}$$

where μ is a nonnegative constant. All trajectories starting from any initial state (x_1, x_2)

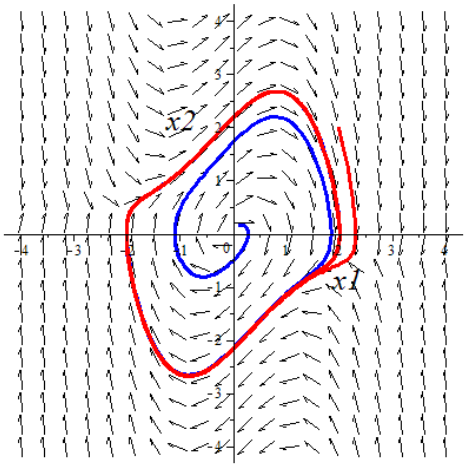


Figure A-1: Phase portrait of a Van der Pol oscillator with $\mu = 1$

converge towards a closed curve, as illustrated in Figure A-1. The trajectories starting form initial conditions

$$(x_{1_{i1}}, x_{2_{i1}}) = (0.1; 0.15) \quad \text{and} \quad (x_{1_{i2}}, x_{2_{i2}}) = (2; 2) \tag{A-3}$$

are shown as the blue and red curve respectively. It is possible to proof mathematically that the Van der Pol equation has a unique limit cycle using the Poincare-Bendixson theorem [5], but this is beyond the scope of this thesis.

A-2 Lyapunov stability analysis on limit cycles used for obstacle avoidance

The stability of the limit cycles described in Equations (2-6) can be shown using the direct Lyapunov stability analysis method. The direct Lyapunov method begins with the intuitive notion that one measure of the state of a physical system is the total energy stored in the system [16]. For a stable system, the energy decreases over time and obtains a minimum at a stable equilibrium point x_e (e.g. a damped oscillator). Here x is a vector containing x_1 and x_2. A scalar function $V(x)$ of the states, called a Lyapunov function candidate, is defined having the following properties:

- $V(x_e) = 0$
- $V(x) > 0$, $\forall x \neq x_e$
- V is continuous and has continuous derivatives with respect to all components of x

The Lyapunov function candidate is considered a Lyapunov function if the time derivative along the trajectories of the system is less than or equal to zero for all states,

- $\dot{V}(x) = \frac{\partial V}{\partial x}\dot{x} \leq 0$

If a Lyapunov function can be found for a system, then the equilibrium is stable. If $V(x) < 0$ and $V(\mathbf{x})$ is radially unbounded ($||x|| \to \infty \Rightarrow V(x) \to \infty$) then the equilibrium is globally asymptotically stable [16], meaning that all trajectories will converge towards the equilibrium point as $t \to \infty$

Consider for the system (2-6) the Lyapunov function candidate

$$V(x) = x_1^2 + x_2^2 \tag{A-4}$$

The time derivative of the Lyapunov function candidate along the trajectories of the system (defined in (2-6)), is given by

$$\begin{aligned} \dot{V}(x) &= 2x_1\dot{x}_1 + 2x_2\dot{x}_2 \\ &= 2x_1x_2 + 2x_1^2(1 - x_1^2 - x_2^2) - 2x_1x_2 + 2x_2^2(1 - x_1^2 - x_2^2) \\ &= 2V(x)(1 - V(x)). \end{aligned} \tag{A-5}$$

The derivative of $V(x)$ is positive for $V(x) < 1$ and negative for $V(x) > 1$. Hence, on the level surface $V(x) = c_1$ with $0 < c_1 < 1$ all trajectories will be moving outward, while on the

level surface $V(x) = c_2$ with $c_2 > 1$ all trajectories will be moving inward. This shows that the annular region

$$M = \{x \in \mathbb{R}^2 | c_1 \leq V(x) \leq c_2\} \tag{A-6}$$

is positively invariant; that is, every solution of (A-6) that starts in M remains in M for all $t \geq 0$. It is also closed, bounded and free of equilibrium points since the origin $x = 0$ is the only equilibrium point. From the Poincare-Bendixson theorem it can be concluded that there is a periodic orbit in M. Since the above argument is valid for any $c_1 < 1$ and any $c_2 > 1$, we can let c_1 and c_2 approach 1 so that the set M shrinks toward the unit circle. This, in effect, shows that the unit circle is a periodic orbit [27].

A-3 Example of the limit cycle method for obstacle avoidance

An example of the limit cycle method described in Section 2-5-2 is shown in Figure A-2. The following steps are used:

1. Draw a line from the robot to the target in a global coordinate frame $\sum OXY$; the line l is represented by
$$l : ax + by + c = 0. \tag{A-7}$$
2. Treat any obstacle as a disturbing obstacles O_{d} if the line l crosses them, else, threat it as a non-disturbing obstacle O_{n}.
3. Move towards the target if there is no disturbing obstacle O_{d}.
4. Otherwise, follow the limit cycle trajectory that is generated around the closest disturbing obstacle. Repeat steps (1)-(4) until the destination is reached.

At the start position a line l_{start} is drawn from start to goal position. Obstacle A is a disturbing obstacle at this point, obstacle B is a non-disturbing obstacle. Step 4 is followed until point 1, at this point B becomes a disturbing obstacle and A a non-disturbing obstacle. The robot follows step 4 for obstacle B until point 2, at this point there are no more disturbing obstacles, and the robot moves toward the goal position (step 3).

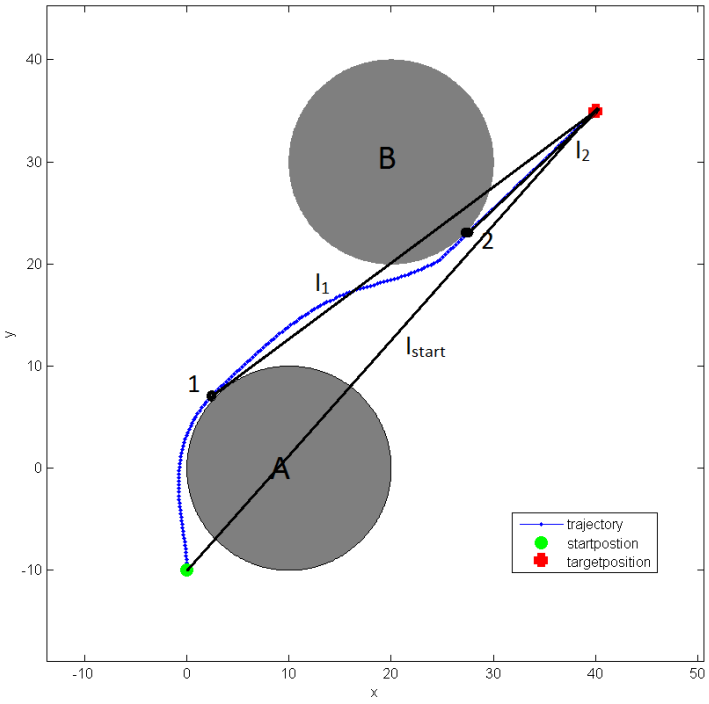


Figure A-2: Navigation example

Appendix B

On 3D limit cycles

B-1 Finding minimum of function

As discussed in Section 3-3-2, the attractor described by the differential equations (3-11) is stable for trajectories starting outside the attractor, provided that

$$f(\mathbf{x}) = \frac{x}{a^2}(x - A_1) + \frac{y}{b^2}(y - A_2) + \frac{z}{c^2}(z - A_3) \tag{B-1}$$

is larger than zero for all $\mathbf{x} \neq \mathbf{x}_e$. The minimum of this function (which should be larger than zero) can be found by solving the constrained optimization problem

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t.} \quad g(\mathbf{x}) = 1 - \left(\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} \right) \leq 0. \end{aligned} \tag{B-2}$$

The minimum $\mathbf{x}^*$ is obtained using Karush-Kuhn-Tucker multipliers [6]. The KKT conditions state that there exists a μ such that

$$\begin{aligned} \nabla f(\mathbf{x}^*) + \mu \nabla g(\mathbf{x}^*) &= 0 \\ \mu \cdot g(\mathbf{x}^*) &= 0 \\ \mu &\geq 0 \\ g(\mathbf{x}^*) &\leq 0. \end{aligned} \tag{B-3}$$

First, the Lagrangian is computed as

$$\mathcal{L}(\mathbf{x}, \lambda) = \frac{x}{a^2}(x - A_1) + \frac{y}{b^2}(y - A_2) + \frac{z}{c^2}(z - A_3) + \mu \left(1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} \right), \tag{B-4}$$

then, the optimality conditions are given by

$$\begin{aligned} 2\frac{x}{a^2} - \frac{A_1}{a^2} - \mu\frac{2x}{a^2} &= 0 \\ 2\frac{y}{b^2} - \frac{A_2}{b^2} - \mu\frac{2y}{b^2} &= 0 \\ 2\frac{z}{c^2} - \frac{A_3}{c^2} - \mu\frac{2z}{c^2} &= 0 \\ \mu\left(1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2}\right) &= 0 \\ \mu &\geq 0 \\ 1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} &\leq 0. \end{aligned} \tag{B-5}$$

From the complementary slackness condition, it follows that either

$$\mu = 0 \quad \text{or} \quad \left(1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2}\right) = 0.$$

When $\mu = 0$, then, from the first three conditions it follows that $\mathbf{x} = \frac{A}{2}$. However, since A is a point on the surface of the attractor it follows that

$$\left(\frac{A_1}{a}\right)^2 + \left(\frac{A_2}{b}\right)^2 + \left(\frac{A_3}{c}\right)^2 = 1,$$

and, consequently, that

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = \frac{1}{4}\left(\left(\frac{A_1}{a}\right)^2 + \left(\frac{A_2}{b}\right)^2 + \left(\frac{A_3}{c}\right)^2\right) = \frac{1}{4}.$$

This does not satisfy the inequality constraint, so μ must be greater than zero, and $\left(1 - \frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2}\right) = 0$. All points on the surface of the attractor satisfy the KKT conditions. The smallest μ for which the inequality constraint is met, is $\mu = \frac{1}{2}$, which results in $\mathbf{x}^* = A$.

Bibliography

[1] C. Ahrikencheikh and A. A. Seireg. *Optimized-Motion Planning: Theory and Implementation*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion,grand challenges of robotics. *IEEE Robotics and Automation Magazine*, 14(1):61–70, 2007.

[3] A. Benzerrouk, L. Adouane, and P. Martinet. Dynamic obstacle avoidance strategies using limit cycle for the navigation of multi-robot system. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[4] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

[5] W. Boyce. *Elementary Differential Equations and Boundary Value Problems: International Student Version*. International student version. John Wiley & Sons, 2008.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] S. Campbell, W. Naeem, and G. W. Irwin. A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres. *Annual Reviews in Control*, 36(2):267–283, 2012.

[8] Y. Chen, H. He, and X. An. Motion planning of intelligent vehicles: A survey. In *2006 IEEE International Conference on Vehicular Electronics and Safety, ICVES*, pages 333–336, 2006.

[9] F. Cheng and Y. Ma. Application of ellipse limit-cycle navigation to fast mobile robots. In *2010 Chinese Control and Decision Conference, CCDC 2010*, pages 1434–1437, 2010.

[10] P. Cheng and P. Chen. Navigation of mobile robot by using d++ algorithm. *Intelligent Service Robotics*, 5(4):229–243, 2012.

[11] A. C. C. Coolen and T. W. Ruijgrok. Image evolution in hopfield networks. *Physical Review A*, 38(8):4253–4255, 1988.

[12] L. Ellekilde and J. Perram. Tool center trajectory planning for industrial robot manipulators using dynamical systems. *Int. J. Rob. Res.*, 24(5):385–396, 2005.

[13] A. Elwakil and M. Kennedy. A semi-systematic procedure for producing chaos from sinusoidal oscillators using diode-inductor and fet-capacitor composites. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 47(4):582–590, 2000.

[14] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17(7):760–772, 1998.

[15] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[16] G. Franklin, J. Powell, D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems: International Edition*. Pearson Prentice Hall, 2009.

[17] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[18] X. Gao, M. Wan, Q. ju Huang, and C. xiang Liu. Flocking of multi-agents with obstacle avoidance based on limit-cycle. In *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, volume 1, pages 549 –553, may 2010.

[19] V. Gavrilets, B. Mettler, and E. Feron. Human-inspired control logic for automated maneuvering of miniature helicopter. *Journal of Guidance, Control, and Dynamics*, 27(5):752–759, 2004.

[20] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 57(1-4):65–100, 2010.

[21] R. Grech and S. G. Fabri. Trajectory tracking in the presence of obstacles using the limit cycle navigation method. In *Proceedings of the 20th IEEE International Symposium on Intelligent Control, ISIC '05 and the 13th Mediterranean Conference on Control and Automation, MED '05*, volume 2005, pages 101–106, 2005.

[22] N. Hara, H. Kokame, and K. Konishi. Circular periodic motion generation for mobile robots using limit cycle systems. In *American Control Conference (ACC), 2010*, pages 4271 –4276, 30 2010-july 2 2010.

[23] N. B. Hui and D. K. Pratihar. A comparative study on some navigation schemes of a real robot tackling moving obstacles. *Robotics and Computer-Integrated Manufacturing*, 25(4-5):810–828, 2009.

[24] Y. K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[25] M. S. Jie, J. H. Baek, Y. S. Hong, and K. W. Lee. *Real time obstacle avoidance for mobile robot using limit-cycle and vector field method*, volume 4251 LNAI - I of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2006.

[26] C. Karney. Algorithms for geodesics. *Journal of Geodesy*, 87(1):43–55, 2013.

[27] H. Khalil. *Nonlinear Systems*. Prentice Hall, 1996.

[28] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[29] D. Kim and J. Kim. A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer. *Robotics and Autonomous Systems*, 42(1):17–30, 2003.

[30] Y. Kim, J. Ki, and D. Kwon. Evolutionary programming-based univector field navigation method for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 31(3):450–458, 2001.

[31] J. J. Kuffner Jr. and S. M. La Valle. Rrt-connect: an efficient approach to single-query path planning. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001, 2000.

[32] R. Kulic and Z. Vukic. Methodology of concept control synthesis to avoid unmoving and moving obstacles (ii). *Journal of Intelligent and Robotic Systems: Theory and Applications*, 45(3):267–294, 2006.

[33] F. Large, C. Laugier, and Z. Shiller. Navigation among moving obstacles using the nlvo: Principles and applications to intelligent vehicles. *Autonomous Robots*, 19(2):159–171, 2005.

[34] S. M. Lavalle. Robot motion planning: A game-theoretic foundation. *Algorithmica (New York)*, 26(3-4):430–465, 2000.

[35] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[36] S. M. LaValle. Motion planning: Part ii: Wild frontiers. *IEEE Robotics and Automation Magazine*, 18(2):108–118, 2011.

[37] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[38] Y. Lee and Y. Kim. Distributed unmanned aircraft collision avoidance using limit cycle. In *Control, Automation and Systems (ICCAS), 2011 11th International Conference on*, pages 121 –125, oct. 2011.

[39] S. Leyendecker, S. Ober-Blöbaum, J. E. Marsden, and M. Ortiz. Discrete mechanics and optimal control for constrained systems. *Optimal Control Applications and Methods*, 31(6):505–528, 2010.

[40] Z. Li and J. Canny. *Nonholonomic Motion Planning*. Kluwer international series in engineering and computer science: Robotics. Kluwer Academic, 1993.

[41] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[42] J. Ng and T. Bräunl. Performance comparison of bug navigation algorithms. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 50(1):73–84, 2007.

[43] D. K. Pratihar, K. Deb, and A. Ghosh. A genetic-fuzzy approach for mobile robot navigation among moving obstacles. *International Journal of Approximate Reasoning*, 20(2):145–172, 1999.

[44] Z. Qu, J. Wang, and C. E. Plaisted. A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. *IEEE Transactions on Robotics*, 20(6):978–993, 2004.

[45] J. H. Reif. Complexity of the mover's problem and generalizations. *Annual Symposium on Foundations of Computer Science - Proceedings*, pages 421–427, 1979.

[46] A. Richards, T. Schouwenaars, J. P. How, and E. Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[47] V. Sezer and M. Gokasan. A novel obstacle avoidance algorithm: "follow the gap method". *Robotics and Autonomous Systems*, 60(9):1123–1134, 2012.

[48] C. Shi, Y. Wang, and J. Yang. A local obstacle avoidance method for mobile robots in partially known environment. *Robotics and Autonomous Systems*, 58(5):425–434, 2010.

[49] L. E. Sjöberg and M. Shirazian. Solving the direct and inverse geodetic problems on the ellipsoid by numerical integration. *Journal of Surveying Engineering*, 138(1):9–16, 2012.

[50] R. A. Soltan, H. Ashrafiuon, and K. R. Muske. Ode-based obstacle avoidance and trajectory planning for unmanned surface vessels. *Robotica*, pages 1–13, 2010.

[51] M. Spong and S. Hutchinson. *Robot Modeling and Control*. Wiley, 2005.

[52] G. Tan, H. He, and A. Sloman. Ant colony system algorithm for real-time globally optimal path planning of mobile robots. *Zidonghua Xuebao/Acta Automatica Sinica*, 33(3):279–285, 2007.

[53] S. H. Tang, W. Khaksar, N. B. Ismail, and M. K. A. Ariffin. A review on robot motion planning approaches. *Pertanika Journal of Science and Technology*, 20(1):15–29, 2012.

[54] D. S. Yershov and S. M. Lavalle. Simplicial dijkstra and a * algorithms: From graphs to continuous spaces. *Advanced Robotics*, 26(17):2065–2085, 2012.

[55] N. K. Yilmaz, C. Evangelinos, P. F. J. Lermusiaux, and N. M. Patrikalakis. Path planning of autonomous underwater vehicles for adaptive sampling using mixed integer linear programming. *IEEE Journal of Oceanic Engineering*, 33(4):522–537, 2008.

[56] J. Yoo, J. Park, and H. J. Kim. Distributed control for multi-target surveillance using limit cycle. In *Control Automation and Systems (ICCAS), 2010 International Conference on*, pages 2361 –2364, oct. 2010.

[57] L. Zeng and G. M. Bone. Mobile robot navigation for moving obstacles with unpredictable direction changes, including humans. *Advanced Robotics*, 26(16):1841–1862, 2012.

[58] A. Zou, Z. Hou, S. Fu, and M. Tan. *Neural networks for mobile robot navigation: A survey*, volume 3972 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2006.