Project report fall 2007

Collision Avoidance Concepts for Marine Surface Craft

Øivind Loe

Trondheim, December 19, 2007

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology, Mathematics and Electrical Engineering
DEPARTMENT OF ENGINEERING CYBERNETICS

Preface

Robots and vehicles operating without the need of human intervention have always amazed me. Being a fundamental requirement for autonomous navigation, *collision avoidance* has thus been the subject of my project. The project work has been very time consuming, but at the same time very rewarding. I hope others can benefit from this work as well.

I would like to express my gratitude to Morten Breivik, my supervisor on this project. He has been a constant motivator, and has provided invaluable help during all parts of the project.

Øivind Loe
Trondheim, December 2007

Abstract

Effective collision avoidance is a prerequisite for autonomous vehicles. This report is a review of several methods for collision avoidance, including the Potential Field method, the Vector Field Histogram method, the Dynamic Window method, a method based on the A* algorithm, the Rapidly-Exploring Random Tree method and finally a method based on Constrained Nonlinear Optimization. In addition to theoretical discussions of the methods, qualitative and quantitative comparisons are made between the methods, as well as hybrid methods based on them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

As technology is progressing, we are steadily moving from a world of manual control, to a world where more and more processes are automatically controlled using computer technology. Process plants, aircraft autopilots, and even washing machines are all examples of this tendency. Automation is all about relieving humans of the three D's: Dull, Dirty and Dangerous.

Already, remotely controlled robots are performing complex tasks for us, such as bomb destruction and exploration of hazard zones. Direct or remote control is however not always possible or even economical, and as the presence of human intervention decreases, the need for autonomy increases.

Autonomy can be defined as a robot's ability to make its own decisions in order to reach a goal. This is a very exciting prospect in robotics.

For autonomy to be feasible, a robot must be able to navigate its environment, avoiding any obstacles. There has been a lot of research activity in this field, and one example of how far we actually have come is the DARPA Grand Challenge [1, 57].

The Grand Challenge is a competition sponsored by the *Defense Advanced Research Projects Agency* (DARPA), the central research organization of the United States Department of Defense. The competition was first held in the Mojave Desert region of the United States in 2004, where the competing vehicles had to travel through 241 kilometres of rough terrain, completely without human intervention. None of the vehicles finished however: The most successful vehicle got stuck after only 11.8 kilometres.

DARPA repeated the challenge in 2005 with a similar course, and this time five vehicles completed the entire course. Having bested the *Grand Challenge*, a new challenge was put forth for 2007; the *DARPA Urban Challenge*. The new challenge had the vehicles navigating a 96 kilometre urban course while following traffic rules and negotiating traffic. Of the 11 teams participating, 6 teams finished the Urban Challenge. This was a major achievement.

Figure 1.1: This is *Boss*, the Chevy Tahoe used by *Tartan Racing*, the winners of the Urban Challenge. (Image courtesy of Tartan Racing)

The main focus of this report is on collision avoidance systems for *Unmanned Surface Vehicles* (USVs), but the theory and algorithms are applicable to most dynamical systems. For a USV, a good collision avoidance system can serve one of two purposes: Assist an operator in avoiding collisions, or help provide complete autonomous control of the vehicle.

The holy grail of motion control is being able to give your vehicle a mission, and then trust that it will complete the mission satisfactorily. Another prospect is giving an operator responsibility of hundreds of USVs performing some task, only requiring supervisory control. Autonomous control is approaching, and collision avoidance is an important ingredient.

1.2 Contribution

The contribution of this report is a review of major algorithms for collision avoidance. Both qualitative and quantitative comparisons have been made, indicating the strengths and weaknesses of the different approaches.

To be able to make the quantitative comparison, 6 different collision avoidance algorithms have been implemented, mainly in the Matlab programming language. Details on the implementation of these algorithms are elaborated on.

The algorithms have been combined to produce a total of 15 controllers, both local, global and hybrids. All of these controllers have been run through 5 different scenarios, and the results have been represented.

1.3 Outline

This report is made up of several distinct chapters. The second chapter is a purely theoretical review of collision avoidance algorithms. In the third chapter, focus is on implementation details, both of the developed simulator and the algorithms. The fourth chapter shows the results of the simulations, and provides short discussions of these. Practical considerations are made in Chapter five, including *the rules of the road* and sensor equipment. A final discussion is made in Chapter six, and a conclusion can be found in Chapter seven.

1.4 Notation

$\mathbf{x}$	State vector
$\mathbf{u}$	Input vector; vector of manipulated variables
ψ	The heading of a vehicle
ψ_{sp}	Set-point for heading
u	Surge velocity of a vehicle
u_{sp}	Set-point for surge velocity
$\mathcal{W}$	Workspace
$\mathcal{C}$	Configuration space
$\mathcal{C}_{free}$	Free part of $\mathcal{C}$
$\mathcal{C}_{obs}$	Obstructed part of $\mathcal{C}$

1.5 Abbreviations

AIS	Automatic Identification System
ARPA	Automatic Radar Plotting Aid
CA	Collision Avoidance
COLREGS	International Regulations for Avoiding Collisions at Sea
CPA	Closest Point of Approach
DOF	Degree Of Freedom
DW	Dynamic Window
GPS	Global Positioning System
IMU	Inertial Measurement Unit
MPC	Model Predictive Control
VFF	Virtual Force Field
VFH	Vector Field Histogram
DW	Dynamic Window
RRT	Rapidly-Exploring Random Tree
TCPA	Time of Closest Point of Approach
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle

Chapter 2

Theory

2.1 Preliminaries

Over the years, a wealth of different collision avoidance methods have been developed. The methods differ greatly in their levels of sophistication, ranging from the obvious methods such as *the potential field method* [24, 25, 28] to more involved methods as constrained numerical optimization [34].

2.1.1 Global vs local methods

Before getting intimate with some of the different methods, a couple of distinctions have to be made. The most important may be the difference between global and local collision avoidance methods.

Global methods are often referred to as *path planning* or *motion planning* methods. They store information about the surrounding environment in a map, and use this map to find a route to a given *goal*. The goal can either be a set of coordinates, or a complete description of the vehicle state, including velocity, orientation, etc.

A global method is thus a method that uses global information about the environment. This includes information which is not *sensable* from the current state of the vehicle. The major downside to the global methods is their long computation time and memory usage. The amount of time needed results in updates taking everything from a couple of seconds to several minutes. This makes the global methods inappropriate for rapidly changing dynamic environments, as their reaction time is too long.

Local methods on the other hand, require orders of magnitude less computing power than the global methods. This makes them excellent for dynamic environments, as they are able to see and react to changes very quickly. The local methods are fast because they only consider the immediate environment of a vehicle. Local methods can make good progress towards the goal, but they can never guarantee that the goal is reached. In other words: global convergence cannot be proved for local methods.

Global methods may also fail in finding a valid path to the goal, but they are more

likely to succeed than the local methods.

Global methods are often referred to as *deliberate* methods, while local methods are named *reactive* or *reflexive*.

2.1.2 The hybrid approach

As explained in Section 2.1.1, both global and local methods have their weaknesses. The hybrid methods try to solve this problem by combining a global method with a local method. This is often done as a multi-tier system as in Figure 2.1, where the global method generates a nominal path, and the local method tries to follow it.

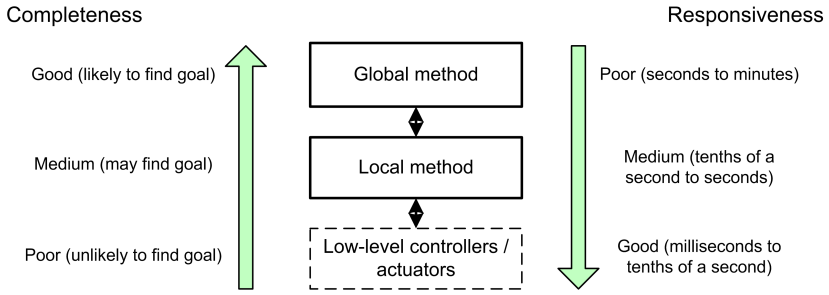


Figure 2.1: The hybrid architecture matches a global method with a local method to create a controller with good responsiveness, and a good chance of efficiently finding the goal. Completeness is used as a measure of a methods ability to find a solution to a problem if such a solution exists. Low-level controllers are included for reference.

As the local method runs at a higher rate than the global method, it is able to see and react to events not planned for by the global method. This makes the hybrid approach more robust than the global method alone, while it still maintains the tractable properties of the global methods.

For examples using the hybrid approach, see [3, 10, 30].

2.1.3 Spaces

The work space

Another thing worth mentioning is the use of different spaces in the collision avoidance algorithms. The physical two- or three-dimensional environment of the vehicle is called the *workspace* $\mathcal{W}$ of the vehicle. We thus have $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$ depending on the number of world dimensions. Regions in $\mathcal{W}$ occupied by obstacles are denoted $\mathcal{O} \subset \mathcal{W}$, and are forbidden for our vehicle.

The configuration space

To make collision avoidance simpler to manage, especially for vehicles with many *degrees of freedom* (DOF), a new space is defined, namely *the configuration space* $\mathcal{C}$ [32]. Also known as the *C-space*, the configuration space of a vehicle contains all possible configurations of the vehicle. A 6 DOF vehicle then results in a 6-dimensional configuration space.

$\mathcal{C}$ also contains regions forbidden because of physical constraints such as obstructions. If our vehicle is defined by $\mathcal{V}(q) \subset \mathcal{W}$ at a configuration/state q, the forbidden configurations are defined as:

$$\mathcal{C}_{obs} = \{q \,\epsilon\, \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}. \tag{2.1}$$

$\mathcal{C}_{obs}$ thus contains all the configurations of our vehicle where it would have been intersecting any obstacles. By avoiding this set, our vehicle avoids collisions. The set containing the allowed configurations is called the *free set* and is given by:

$$\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}. \tag{2.2}$$

Collision avoidance is now reduced to keeping our vehicle inside $\mathcal{C}_{free}$, and path planning is a matter of finding a path from q_{init} to q_{goal} inside $\mathcal{C}_{free}$

Other spaces

The *Ego-Dynamic* space [41], and the more advanced *Ego-KinoDynamic* space [40] are examples of other spaces that can make collision avoidance even simpler. They are both derived from the configuration space $\mathcal{C}$. They add features such as removing parts from $\mathcal{C}_{free}$ that our vehicle cannot access without entering $\mathcal{C}_{obs}$ at a later time.

2.2 Local methods

In this section, three different local collision avoidance algorithms are presented. They range from the simple *potential field* method, to the more advanced *dynamic window* approach.

2.2.1 Potential Field

The potential field method is one of the most intuitive schemes for collision avoidance. The algorithm is very simple, and with it, one can achieve pretty decent results with very little development time. It was proposed by Khatib in 1985 [24, 25].

The main idea behind the potential field method is that every obstacle exerts a repulsive force on our vehicle, while the destination exerts an attractive force. If our vehicle then applies this force to guide itself, it will hopefully end up at the target. We will later see that this method is far from perfect.

Virtual force

The potential field generated by the obstacles is defined by:

$$U_O(\mathbf{p}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & if\ \rho \leq \rho_0 \\ 0 & if \rho > \rho_0 \end{cases}, \tag{2.3}$$

where $\mathbf{p}$ is the vector from a point on an obstacle to our vehicle, $\rho = \|\mathbf{p}\|_2$, and ρ_0 represents the limit distance for how far the potential field from an obstacle reaches. In the original potential field method, one point is used for each obstacle. This point is selected as the one closest to the vehicle.

The force generated by this potential field is:

$$F_O(\mathbf{p}) = \begin{cases} \eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{\mathbf{p}}{\rho^3} & if\ \rho \leq \rho_0 \\ 0 & if \rho \geq \rho_0 \end{cases}. \tag{2.4}$$

The attractive potential field generated by the destination can be expressed as in (2.5), where $\mathbf{x}$ is the coordinate vector of the vehicle and $\mathbf{x}_d$ is the destination:

$$U_{\mathbf{x}_d}(\mathbf{x}) = k_p\mathbf{e}^2, \qquad \mathbf{e} = \mathbf{x} - \mathbf{x}_d. \tag{2.5}$$

The force generated by this field works as a general P-controller. Two modifications have to be made before the force can be used. First, to improve stability of the method, a dissipative force, proportional to the velocity of our vehicle $\dot{\mathbf{x}}$ is added. Second, the maximal force generated by the proportional part of the force has to be given an upper limit. This is to make sure the behavior of the method is consistent and not depending on the distance to the goal. The final force generated by the destination is given as:

$$F_{\mathbf{x}_d}(\mathbf{x}) = -k_v\dot{\mathbf{x}} - k_v \begin{cases} \frac{k_p}{k_v}\mathbf{e} & if\ \ \frac{k_p}{k_v}\|\mathbf{e}\| \leq V_{max} \\ V_{max}\frac{\mathbf{e}}{\|\mathbf{e}\|} & else \end{cases}. \tag{2.6}$$

The maximum force generated by the proportional part of this force is $k_v V_{max}$.

The sum of the forces determines the desired force on our vehicle. It is now up to the individual control allocation algorithm of the vehicle to make the actuators apply the force to the vehicle. For under-actuated vehicles this can pose a problem.

Evaluation

The potential field method is an elegant and simple method for collision avoidance, but it has four major drawbacks, which makes it difficult to use in practice [28].

- *Trap situations due to local minima*
  As with all local methods, the potential field method suffers from the possibility of local minima. If the vehicle gets to a place where the attractive force cancels the repulsive forces, it can get stuck. This is known as a local minimum. The

method finds a solution, but the solution is not the correct one. The presence of local minima is more likely in densely cluttered areas.

- *No passage between closely spaced obstacles*
  The method will not guide the vehicle on a path between closely spaced obstacles, e.g. a gate. As the vehicle approaches the passage, the combined repulsive force from the obstacles will become greater than the attractive force, and the vehicle will end up in a local minimum or avoid the passage, taking another route.

- *Oscillations in the presence of obstacles*
  In the original method, the repulsive force from the obstacles decreases very rapidly with the distance to the obstacles. This means that our vehicle has to move relatively close to the obstacles before it experiences any force. In the presence of multiple obstacles, this can cause oscillatory motion.

- *Oscillations in narrow passages*
  The oscillations become more evident when the vehicle travels through a narrow passage. A proof of this can be found in [28].

These drawbacks become most apparent when navigating through densely cluttered areas, and the method can be quite efficient when the configuration space $\mathcal{C}$ or the vehicle is uncluttered [5]. There are also ways of identifying and escaping local minima which may be used to improve the method; see *Randomized Path Planning* on page 10.

Similar methods

Virtual Force Field The potential field method depends on knowledge about the obstacles in the area to be navigated. A slight variation of the method, called the *virtual force field* (VFF) [27], gathers data through (possibly inaccurate) sensor readings to create a world map. The world map is represented by a two-dimensional Cartesian histogram grid, which contains the probabilities of obstacles being located at different locations in the area. Repulsive forces are then calculated from each of the cells on this map as in (2.4), but also multiplied with the probability value of the cell.

Harmonic Potential Function Fields A way of dealing with the local minima that appear in the potential field method is by the use of harmonic potential functions fields [20, 26]. These fields are called navigation functions, as they have no other local minima than the goal.

There is however a downside to the harmonic potential function fields. The fields are only free of local minima for *point-based* vehicles. For non-point vehicles, they cannot guarantee a global solution. The method also requires complete knowledge about the shape, location and motion of all obstacles, and is thus not very suitable for dynamic collision avoidance.

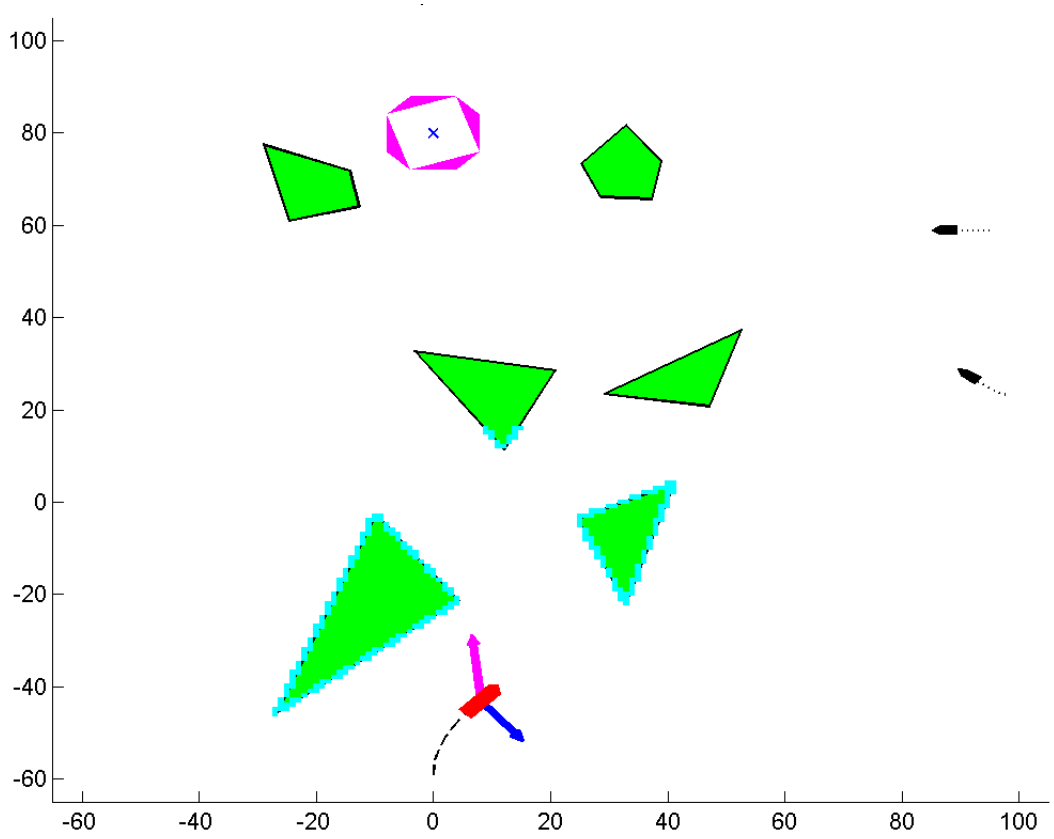


Figure 2.2: This figure shows a collision avoidance scenario. The red vessel in the bottom of the image is supposed to get to the magenta shape in the top of the map without colliding with any of the green obstacles. It also has to watch out for the other vessels coming in from the right of the map. The virtual forces applied by the virtual potential field are visualized as arrows. The magenta arrow pointing towards the goal represents the attractive force. The repulsive force is represented by the blue arrow, and one can clearly see that it tries to keep the vessel away from the first obstacle. The values on the axes are given in metres.

NF1 and NF2 The NF1 and NF2 algorithms are other methods for generating navigation functions [6]. The NF1 method labels all nodes in the map with the L^1 distance to the goal. This field tends to make vehicles hug the contours of obstacles.

The NF2 method creates a skeleton representing passages in the world, reminiscent of a Voronoi diagram, and uses this skeleton when generating the navigation function. NF2 uses more computation time than NF1, but in turn keeps vehicles as far from the obstacles as possible.

Randomized Path Planning This path planner is closely related to the potential field method. It uses a potential field method to navigate the map, and when a local

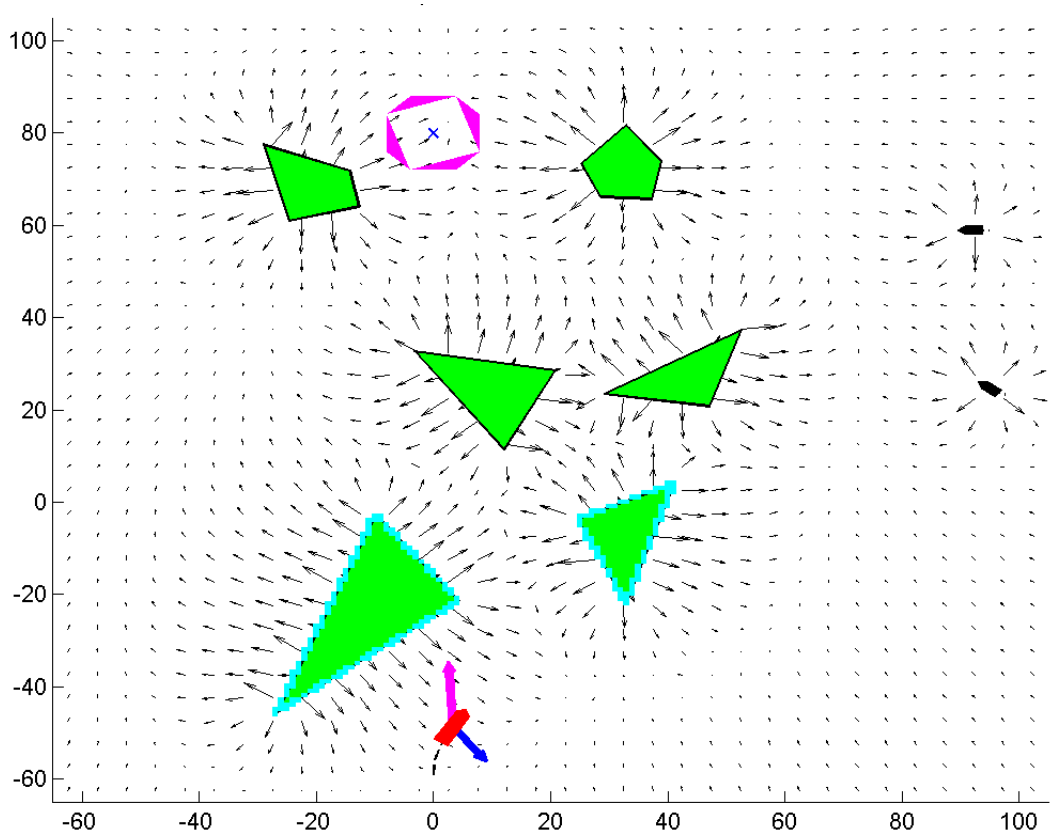


Figure 2.3: This is a visualization of the virtual forces that would be applied to our vessel at different locations of the map. Near obstacles, the repulsive forces dominate, pointing away from the obstacles. In open areas, the attractive forces dominate. Because of the enormous range of magnitudes, all forces $F = [F_x \ F_y]^T$ are displayed as $F' = F\frac{ln(F^T F)}{F^T F}$. Some anomalies can be spotted. These are due to quantization errors.

minimum is encountered, it executes approximations of Brownian motions, i.e. discrete random walks, to escape the minima. This is claimed to be effective for systems with many DOFs [6]. Another way of escaping local minima is simply filling the minimum until a saddle is created, and the vehicle keeps moving.

2.2.2 Vector Field Histogram

The Vector Field Histogram method (VFH) was introduced by Koren and Borenstein in 1990 [8, 27]. The goal was to overcome the inherent limitations of the VFF method, while still maintaining VFFs low computational complexity.

As the VFF method, VFH gathers data on the obstructions in the world in a two-dimensional *Cartesian histogram grid* $\boldsymbol{C}$. Each cell (i, j) in the histogram grid contains

the probability $c_{i,j}$ of an obstruction being present at that location.

Where the VFF method generates control commands directly from this map, the VFH method introduces an intermediate world-representation, to help making good control choices. To reduce calculation load, all cells further away than a given distance d_{max} are disregarded, giving us the *active histogram grid* $\boldsymbol{C}^*$.

The polar histogram

The intermediate world-representation is a one-dimensional polar histogram $\boldsymbol{H}$ constructed around our vehicles current location. $\boldsymbol{H}$ is calculated from $\boldsymbol{C}^*$, and contains n angular sectors h_k, each representing the *polar obstacle density* in that sector.

The angle of cell (i, j) with respect to our vehicle is given by:

$$\beta_{i,j} = \tan^{-1}\left(\frac{y_j - y_0}{x_i - x_0}\right), \tag{2.7}$$

where (x_0, y_0) are the coordinates of our vehicle, and (x_i, y_j) are the coordinates of the cell in question. The polar obstacle density for each of the sectors is then:

$$h_k = \sum_{i,j} \begin{cases} (c^*_{i,j})^2(a - bd_{i,j}) & if \quad \lfloor(\beta_{i,j}/\alpha)\rfloor = k \\ 0 & else \end{cases}, \tag{2.8}$$

where α is the sector-span in radians, and $d_{i,j}$ the Euclidian distance between cell (i, j) and our vehicle. The positive constants a and b are selected such that $a - bd_{max} = 0$.

Before proceeding with the rest of the algorithm, the quality of the polar histogram calculated with (2.8) has to be evaluated. A low resolution Cartesian histogram grid will result in a severely jagged polar histogram, which may later give us problems when trying to determine the next steering direction. A solution to this problem is applying a filter to $\boldsymbol{H}$:

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} + \cdots + lh_k + \cdots + 2h_{k+l-1} + h_{k+l}}{2l + 1}. \tag{2.9}$$

The filtered polar histogram can be seen in Figure 2.4.

Selecting the next steering command

As can be seen in Figure 2.4, the polar histogram has *peaks* and *valleys*. The valleys where the values of h'_k are lower than a given threshold are possible candidates for the next steering command. The sector in a valley that is closest to the goal, while still being a certain distance from the valley edge, is selected. Specifics on the means of finding this sector can be found in [8].

Velocity control

The velocity command from the VFH algorithm depends on the obstacle density in the current direction of travel, $h_c^{'}$:

$$V^{'} = V_{max}\left(1 - \frac{\min(h_c^{'}, h_m)}{h_m}\right), \tag{2.10}$$

where V_{max} is an upper limit for the velocity, and h_m is an empirically determined constant. The VFH algorithm extends this command by also reducing the vehicle velocity when performing turns:

$$V = V^{'}\left(1 - \frac{\Omega}{\Omega_{max}}\right) + V_{min}. \tag{2.11}$$

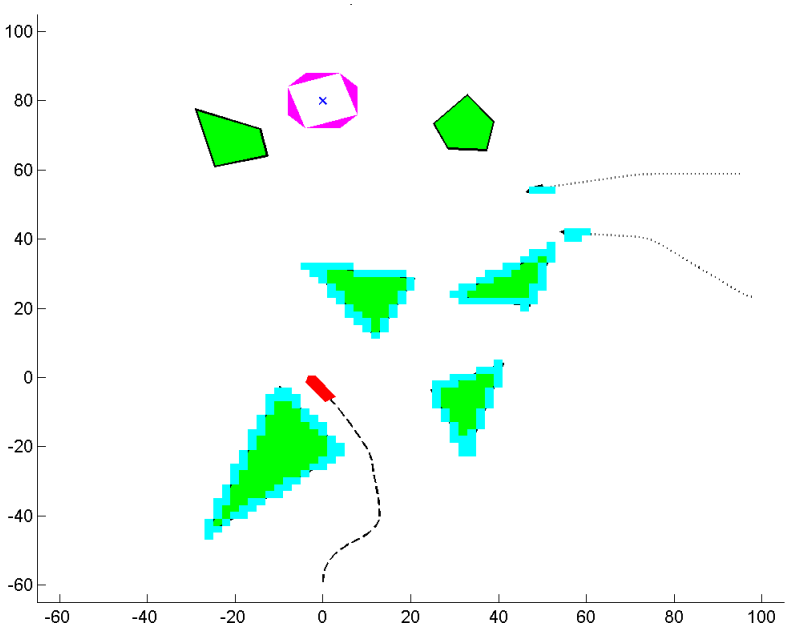
Ω is here defined as the steering rate of the vehicle, and Ω_{max} is the maximum allowable steering rate. V_{min} is added to the equation to make sure the vehicle does not come to a stop while performing its maximum turn.

Evaluation

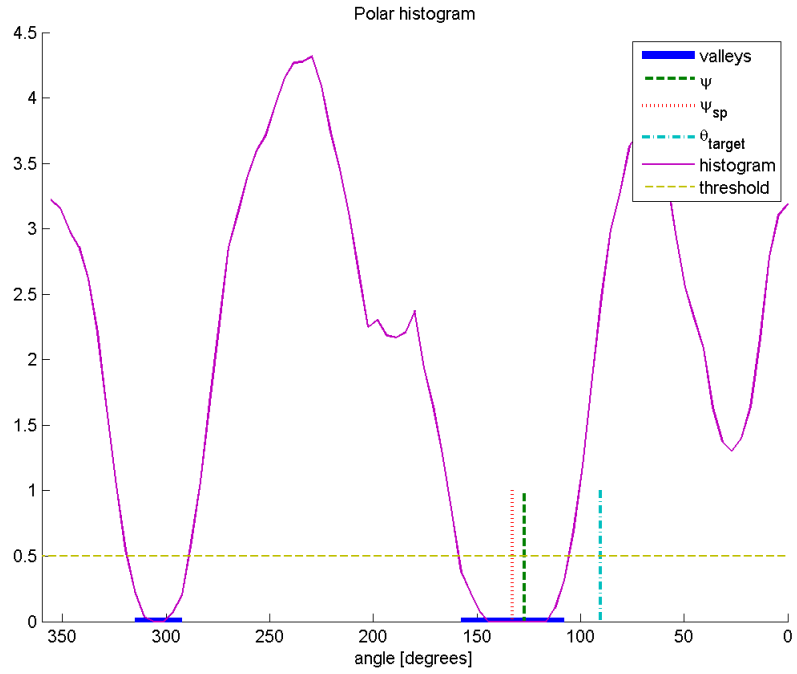
The VFH method was developed to take on the problems experienced with the VFF method. It does not experience the oscillations of the VFF method, and does not have any problems navigating through closely spaced obstacles. The method does not mend the problem with local minima though, as it is also a local, memory-less method.
The method also has an additional downside: As it does not take vehicle dynamics into account, it is better suited to holonomic vehicles or vehicles with fast turning dynamics, and may have problems controlling slower vehicles.

Similar methods

VFH+ The VFH+ method is an enhanced version of the VFH method [51]. It improves VFH by taking the width of the robot into account, and acknowledging that most robots cannot change their direction of travel instantly. The trajectories of the robot are assumed to lie on circular arcs and straight lines, and steering decisions that would lead the robot to collide are masked away from the polar histogram.



(a) Boat and obstacles



(b) Resulting polar histogram and steering command

Figure 2.4: The above figures shows the VFH controller in action. (a) shows a boat and some obstacles, while (b) shows the resulting histogram. θ_{target} is the direction of the target and Ψ is the current heading of the boat. The thick lines at the bottom of the graph are the identified valleys, and Ψ_{sp} is the heading command for the boat calculated by the VFH algorithm. The histogram starts at 0 degrees and revolves counterclockwise around the vessel.

2.2.3 Dynamic Window

The Dynamic Window approach was first published in the article *The Dynamic Window Approach to Collision Avoidance* by Fox, Burgard and Thrun in 1997 [14]. The method was designed to take the limited velocities and accelerations of vehicles into account, giving a collision avoidance method that does not ask for impossible control actions.

Traveling along arcs

The method assumes that the velocities of our vehicle, both angular and translational, are constant within a given time-interval. The trajectory of our vehicle during a time-interval can then be estimated as a straight line, or a constant-radius arc. If we let r denote the angular velocity of our vehicle $r = \dot{\psi}$, the arc center (M_x^i, M_y^i) can be calculated as:

$$M_x^i = -\frac{u_i}{r_i} \sin\theta(t_i) \tag{2.12}$$

$$M_y^i = \frac{u_i}{r_i} \cos\theta(t_i), \tag{2.13}$$

where u_i and r_i are the translational and angular velocities of the vehicle, during interval i. The radius of the arc is given by:

$$M_r^i = \frac{u_i}{r_i}. \tag{2.14}$$

Admissible velocities

The pair (u_i, r_i) now represents the velocities of our vehicle during interval i, and it is up to the Dynamic Window algorithm to select the best values for these. Not all of the pairs represent valid choices.

One of the restrictions put on the choice of (u_i, r_i) is that it must not put the vehicle in danger of colliding in the next interval. In other words: the vehicle must be able to come to a complete stop by braking in the next interval, for the choice to be valid. The set of these velocities is called *admissible velocities*, and is defined as follows:

$$V_a = \left\{ u, r \,\middle|\, u \leq \sqrt{2 \cdot dist(u,r) \cdot \dot{u}_b}, \quad r \leq \sqrt{2 \cdot dist(u,r) \cdot \dot{r}_b} \right\}, \tag{2.15}$$

where $\dot{u}_b$ and $\dot{r}_b$ are the maximum braking accelerations of the vehicle, and $dist(u, r)$ is the distance the vehicle can travel along the given arc before hitting an obstacle.

The notion of admissible velocities in the Dynamic Window method can be related to the *Ego-Dynamic* and *Ego-KinoDynamic* spaces. See Section 2.1.3.

The Dynamic Window

The second restriction to be put on the choice of velocities is that they have to reside in a *dynamic window*. The dynamic window consists of the velocities that can be reached during the next time interval. This window is centered around the current velocities of the vehicle (u_a, r_a), and Δt is the time interval in which the accelerations $(\dot{a}, \dot{r})$ works on it. These accelerations must of course lie within the capabilities of the vehicle. The dynamic window V_d is defined as:

$$V_d = \{u, r \,|\, u \,\epsilon\, [u_a - \dot{v}\Delta t, u_a + \dot{v}\Delta t]\,, \quad r \,\epsilon\, [r_a - \dot{r}\Delta t, r_a + \dot{r}\Delta t]\,\}\,. \tag{2.16}$$

Let V_s denote the set of velocities attainable by our vehicle. The set of valid velocities V_r for the next time interval is then given by (2.17). This set is our search space.

$$V_r = V_s \cap V_a \cap V_d. \tag{2.17}$$

Selecting the optimal velocities

When the search space has been determined, it is time to select the optimal velocities in the space. This is done by maximizing the following objective function over the search space:

$$G(u, r) = \sigma\left(\alpha \cdot \text{heading}(u, r) + \beta \cdot \text{dist}(u, r) + \gamma \cdot \text{velocity}(u, r)\right). \tag{2.18}$$

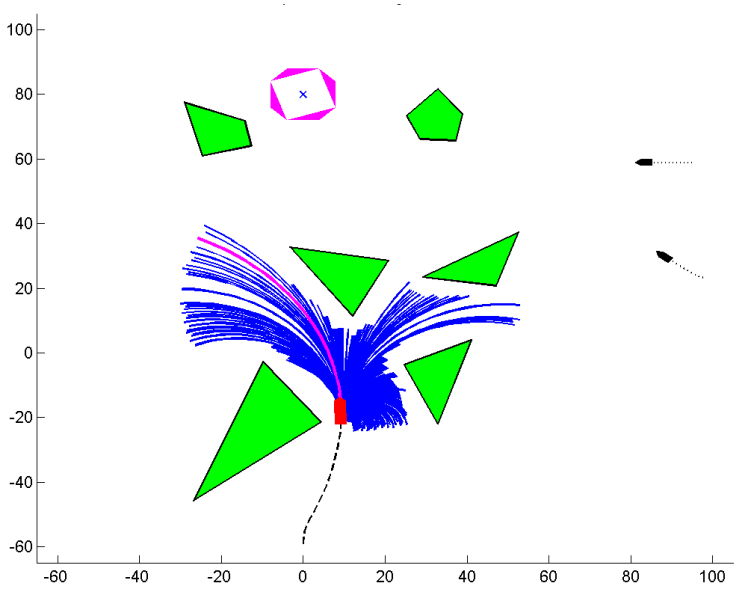
The objective function is a linear combination of the three functions: *heading*, *distance* and *velocity*. Using these functions, maximizing the objective function should make sure our vehicle travels in the right direction while avoiding obstacles and keeping its velocity high.

The distance and velocity functions should be self-explanatory. The heading function is also pretty simple. It is given as $180 - \theta$, where θ is the angle between the vehicle direction and the direction of the target. The vehicle direction is calculated as the direction obtained after a full angular deceleration after the next interval.

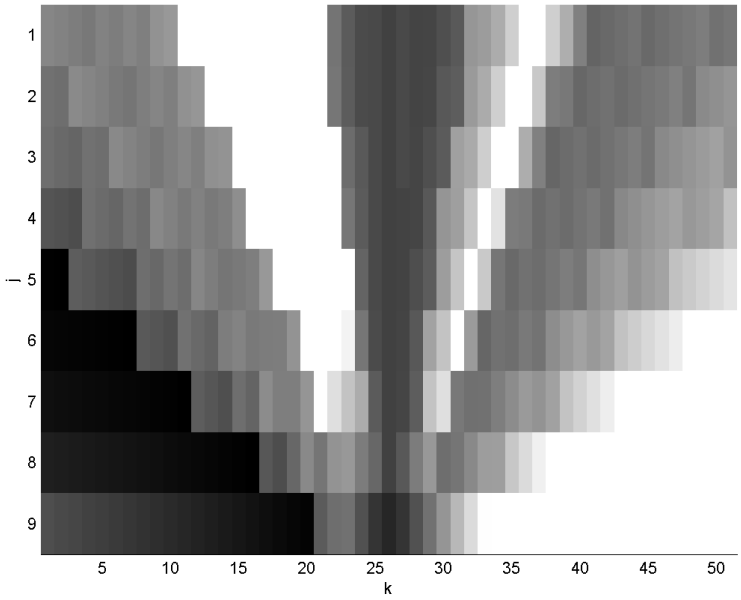
To make the search simple, the search area is discretized, giving us a set of M possible translational velocities u_j, and N possible angular velocities r_k. The number of different choices is now $M \cdot N$, and the value of the objective function can be calculated for each choice. Selecting the optimal velocities is then reduced to a matter of picking the choice with the largest objective function value. The objective function is low-pass filtered using $\sigma(\ldots)$ to help reducing fluctuations in the decision making.

Evaluation

An obvious advantage of this method with respect to the VFH method is that it takes the vehicle dynamics into consideration. A downside to the method is that it requires a good deal more computational power, as the distance function has to be calculated for all the trajectories in the search space.

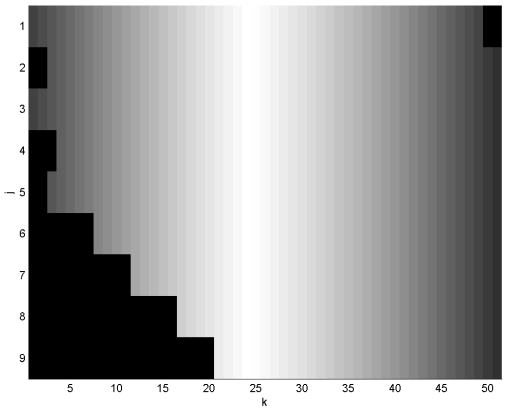


(a) Boat and obstacles

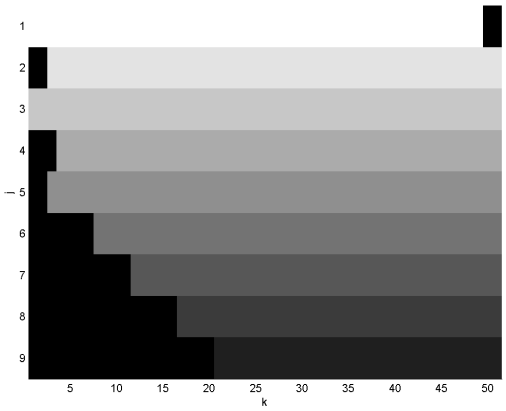


(b) Distance map

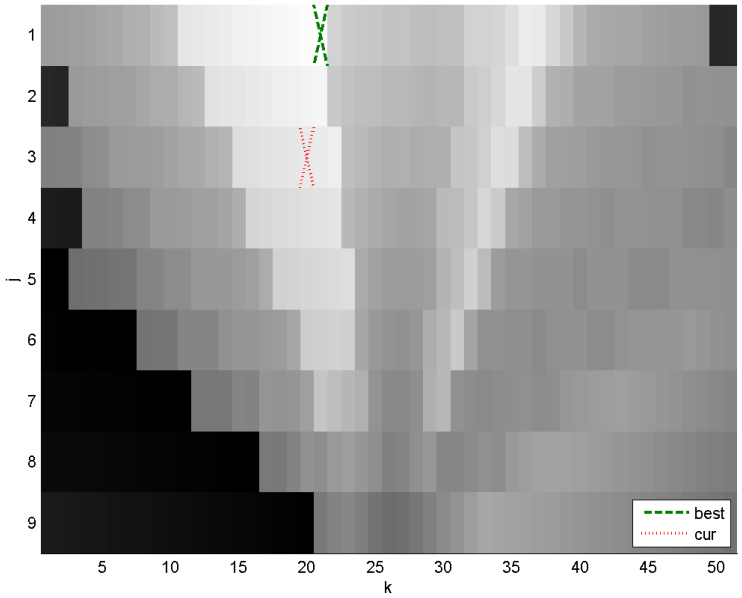
Figure 2.5: In (a) our vessel is navigating the map using the Dynamic Window method. The distance map is shown in (b). The choices of angular velocities lie along the x-axis, and the different surge velocities lie along the y-axis. The brighter a cell in the distance map is, the further our vehicle can travel using that velocity command. Brighter is better.



(c) Heading map



(d) Velocity map



(e) Final dynamic window map

Figure 2.5: The heading-map is displayed in (c). It requests more clockwise angular velocity. The black areas in the map depict velocities that are not admissible. They are not a problem as long as we have other choices. The velocity map in (d) is very simple. Its only job is to bias the selection of velocities towards higher velocities, providing progress through the map. The resulting dynamic window map is shown in (e). Features of the individual maps can still be seen in it. The current velocities of our vehicle is marked with a red cross, and the best choice of velocities, the one that will be carried out, is marked with the green cross at the top of the figure.

Similar methods

Global Dynamic Window Approach The *Global Dynamic Window Approach* [9] is in fact a hybrid method. It combines the dynamic window approach with the *NF1* algorithm (see page 10) in an attempt to provide a method that is not susceptible to local minima.

Examples can be constructed however, where a vehicle using this approach enters a limit cycle, and never reaches the goal [19]. A *provably convergent* version is presented in [19].

Curvature Velocity Method The *Curvature Velocity Method* [46] is closely related to the dynamic window approach. One of the main differences is the way the desired velocity is selected from the velocity space of the vehicle. Instead of discretizing the velocity space and evaluating the utility of resulting set of possible commands, this method relies on constrained optimization to make the selection.

2.3 Global methods

Three global methods will be presented in this section. The first method is a path-planner, while the two others are motion planners. The methods differ in complexity, starting with the simple A* method, continuing with the RRT method, and ending up with a method based on constrained nonlinear optimization.

2.3.1 A*

The first method that comes to mind when thinking about path planning is often Dijkstra's algorithm. It can calculate the shortest path between two points in a map consisting of nodes/cells. In this section, a more general method will be presented, namely the *A-star* method, or simply A*. A* is a *best-first* search algorithm, which uses heuristics to achieve computational optimality [50, 54].

Cell decomposition

To be able to use the A* method, the environment of the vehicle must first be decomposed into cells, connected to make up a graph. There are many ways to do this, including the use of quad-trees, but a straight-forward way is to decompose it into a set of equally sized cells, creating a grid structure covering the map. A bitmap can then be used to represent the map, where a value 1 at location (i, j) in the bitmap means that the area covered by cell (i, j) contains an obstacle. A 0 indicates that the area is clear of obstacles.

This representation can easily be extended to three or more dimensions, to include height information, time, etc. Note however, that the memory requirement grows exponentially with the number of dimensions included.

The A* algorithm

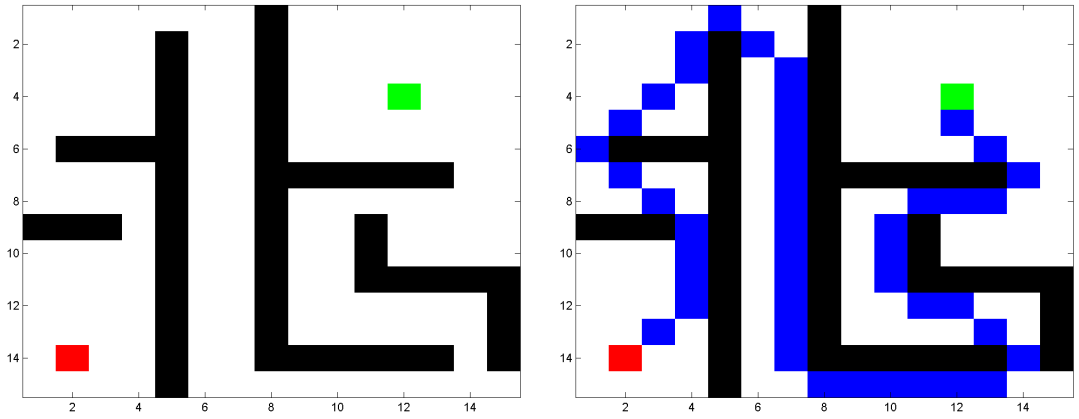
```
1: closed ← ∅
2: cost(start) ← 0
3: open ← MIN_HEAP(start)                          ▷ Sorted by h(node) + cost(node)
4: while open ≠ ∅ do
5:     node ← EXTRACT_MIN(open)
6:     if node = goal then
7:         return GET_PATH(node)                    ▷ Minimal path to goal found
8:     end if
9:     for nb ← GET_NEIGHBOURS(node) do
10:        cost ← cost(node)+ MOVECOST(node, suc)   ▷ Total cost to nb
11:        if nb ϵ open ∧ cost ≤ cost(open(nb)) then
12:            continue                             ▷ Goto next successor
13:        end if
14:        if nb ϵ closed ∧ cost ≤ cost(closed(nb)) then
15:            continue                             ▷ Goto next successor
16:        end if
17:        parent(nb) ← node
18:        cost(nb) ← cost
19:        h(nb) ← heuristic estimate of minimal distance from suc to goal
20:        open ← nb
21:    end for
22:    closed ← node
23: end while
```

The algorithm starts exploring the map at the start coordinates. It uses a *closed* and an *open* set. The closed set contains all the nodes we know a path to, while the open set contains all the nodes we have encountered but not yet explored. At every iteration, the algorithm extracts a node from the open set. If this node is the goal, then we are done. Else, all the immediate neighbors of the node that are not obstructed get added to the open set.

Completeness, admissibility and computional optimality

A* is a complete algorithm, meaning that it will always find a solution if one exists. Whether a solution actually exists in our case, depends on type and resolution of the world map.

To decide which node to examine next, the algorithm sorts all the nodes in the open set by the *estimated total cost* of a path to the goal through the different nodes. The cost from the start to a node in the open set, $g(x)$ is already known. The algorithm needs a minimal estimate $h(x)$ of the cost from the node to the goal. The total estimated cost is then given by $g(x) + h(x)$. If the heuristic function h is admissible, then A* will always



(a) Problem **(b)** Solution

Figure 2.6: (a) represents the problem: The black cells are walls, and the A* algorithm has to find the shortest route from the red cell in the lower-left to the green cell in the upper-right. The solution is shown in (b).

find the optimal path from the start to the goal with respect to the cost of the path. h is admissible if it never over-estimates the cost to the goal.

Evaluation

The A* method is able to calculate the shortest path from our vehicle to the goal through the environment graph, using limited computational power, but possibly large amounts of memory. Whether this path is actually the shortest path through the environment depends on the type and resolution of the graph.

Using the generated path directly may prove difficult, at least in cluttered environments, as the path does not take vehicle dynamics into account. The algorithm may also propose paths that are not possible to execute, and it is thus not complete for our problem.

The A* algorithm has been used extensively as the deliberate, path-planning part of larger navigation and collision avoidance systems. An example can be found in [30].

Similar methods

Dijkstra Dijkstra's algorithm is actually a special case of the A* algorithm. Setting $h(x) = 0 \, \forall \, x$ reduces the A* algorithm to Dijkstra's algorithm. Dijkstra's algorithm will most of the time explore more nodes than A*, since it has no notion of the cost from unexplored nodes to the target. *Breadth-first* search and *Depth-first* search are also special cases of A* [49].

Bellman-Ford The Bellman-Ford algorithm is also an algorithm for finding the shortest path between two nodes in a *graph*, but unlike Dijkstra and A*, it supports

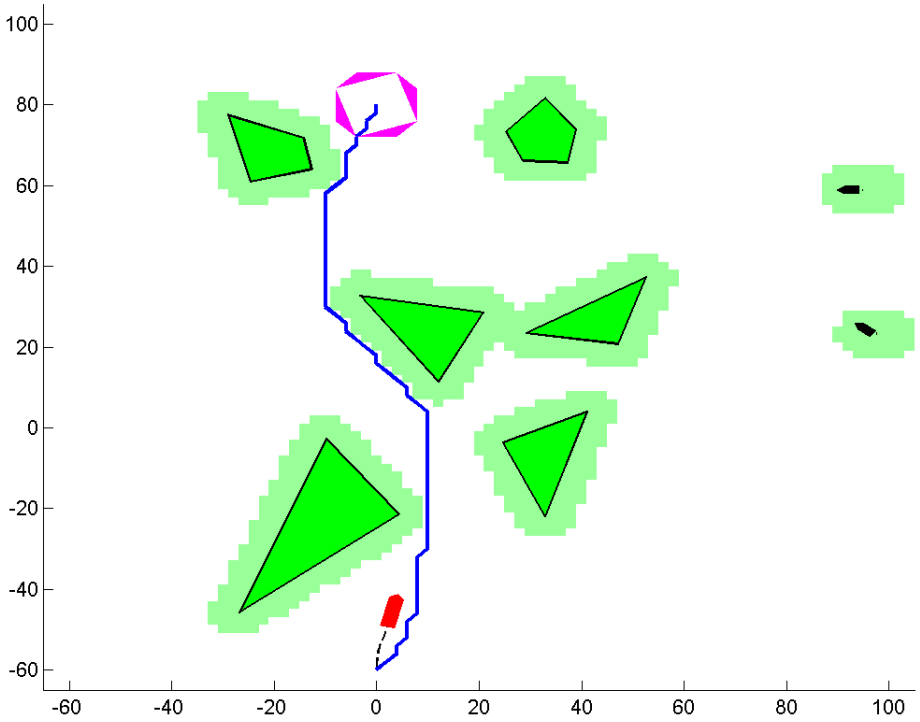


Figure 2.7: The figure shows a path through the map generated by the A* algorithm. The path is generated in the *configuration space* $\mathcal{C}$ of the vessel. The colored regions around the obstacles are part of $\mathcal{C}_{obs}$, and can thus not be traversed by the A* algorithm. The limited resolution of the environment map can clearly be seen in the jagged path generated by the algorithm.

negative costs between nodes [49].

D* The D* algorithm, also known as *Dynamic A**, is a variation of the A* algorithm. It is a better choice than A* when dealing with a partially unknown environment because it is faster at regenerating a path when new data arrives [47].

Visibility Graph Where the depicted A* algorithm finds the shortest way through a cell-representation of the environment, the Visibility Graph algorithm finds a way through using the vertices of the polygonal obstacles [35].

The first step in the algorithm consists of connecting the vertices of the obstacles with each other and the start and goal locations. Each vertex is connected to all other vertices in its line of sight. This forms a graph including the start and goal. The second step consists of finding the shortest path through this graph using a search algorithm such as Dijkstra.

A configuration space representation of the environment must be used for the generated path to be feasible for non-point robots.

2.3.2 Rapidly-Exploring Random Tree

The *Rapidly-Exploring Random Tree* method (RRT), was introduced by LaValle in 1998 [31]. It is a motion-planning method that can plan a path while taking the dynamics of the vehicle into account. As a randomized method, the RRT can explore the space of possible solutions several orders of magnitude faster than a complete method. The solution provided by the RRT is sub-optimal, but will in most cases prove sufficient. Tan discusses the method with application to AUVs in [48], while Frazzoli uses it for spacecraft motion planning in [15].

A tree

The RRT method will, as its name indicates, generate a tree structure during its execution. This tree starts at the state of our vehicle, and iteration after iteration explores larger portions of the configuration space $\mathcal{C}$ of the vehicle. An important property of the RRT method is that it not only plans the coordinates of our vehicle, but all its states: coordinates, heading and velocities. If a path has been generated by the RRT method, we know it is a feasible path.

The algorithm

The RRT algorithm is relatively simple to implement. Given the initial state of our vehicle $\mathbf{x}_{init}$, and the requested goal configuration $\mathbf{x}_{goal}$, the algorithm looks like this:

```
1: T ← TREE({x_init, 0})
2: if CAN_CONNECT(x_init, x_goal) then
3:     T ← {CONNECT_STATE(), CONNECT_INPUT()} as child of x_init
4:     goal found
5: else
6:     for k ← 1...N do                                   ▷ The main loop
7:         x_rand = RANDOM_STATE()
8:         if x_rand ϵ C_free then
9:             x_near = NEAREST_NEIGHBOUR(x_rand, T)
10:            if CAN_CONNECT(x_near, x_rand) then
11:                u ← CONNECT_INPUT()
12:                x_next ← CONNECT_STATE()
13:                T ← {x_next, u} as child of x_near
14:                if CAN_CONNECT(x_next, x_goal) then
15:                    T ← {CONNECT_STATE(), CONNECT_INPUT()} child of x_next
16:                    goal found
17:                end if
18:            end if
19:        end if
20:    end for
```

21: **end if**
22: **if** *goal found* **then**
23: **return** *shortest path from* $\mathbf{x}_{init}$ *to* $\mathbf{x}_{goal}$ *in* $\mathcal{T}$ *with respect to some given metric*
24: **else**
25: **return** $\emptyset$
26: **end if**

To be able to use this algorithm, the functions it uses have to be defined. CAN_CONNECT($\mathbf{x}_1$, $\mathbf{x}_2$) is the function that does most of the work. It uses a local motion planner to try to connect the two given vehicle states $\mathbf{x}_1$ and $\mathbf{x}_2$. For the local planner, a compromise has to be made between its accuracy and computional complexity. An accurate planner will more often be able to connect $\mathbf{x}_1$ and $\mathbf{x}_2$, while a fast planner will be able to make more attempts with different state vectors.

The sets of states are connected by integrating the vehicle from $\mathbf{x}_1$ while applying a series of inputs $\mathbf{u}$. If the vehicle comes within a predefined distance of $\mathbf{x}_2$ within a given time and without exiting its configuration space, then the connection was successful, and the function returns true. CONNECT_INPUT() can later be called to get the series of inputs $\mathbf{u}$ that made this possible, and CONNECT_STATE() can be called to get the final state, which by definition must be in the vicinity of $\mathbf{x}_2$.

RANDOM_STATE() returns a random state vector within the state space of our vehicle. There are many ways of generating these random states, and the way they are generated can have a serious impact on the performance of the algorithm. Using a uniform distribution makes sure all parts of $\mathcal{C}_{free}$ are explored, while selecting a distribution biased towards a smaller area may speed up the algorithm significantly.

NEAREST_NEIGHBOUR($\mathbf{x}$, $\mathcal{T}$) returns the node in $\mathcal{T}$ whose state vector is closest to $\mathbf{x}$ by some metric. Using the Euclidean distance is common as it ensures that the algorithm tries to connect nodes that are geometrically close, creating a short path through $\mathcal{C}_{free}$.

N is the maximum number of times to run the algorithm. Logic can be implemented to end the algorithm prematurely if a satisfying path to $\mathbf{x}_{goal}$ has been located.

When the algorithm fails

The RRT method is not complete. Even though a solution to the planning problem exists, it cannot be guaranteed that the algorithm will find one in a limited time interval. This is however not as catastrophic as it might sound. The most obvious solution to this problem is to re-use the path generated the last time the algorithm was run. This path will likely still be valid.

Another way to solve the problem is to use one of the partial solutions existing in $\mathcal{T}$. All of these branches are feasible, even though they don't end up at the goal. By selecting a branch that makes progress towards the goal, the vehicle has something to do until the completion of the next iteration of the algorithm.

Optimizations and modifications

One of the strengths of the RRT method is the number of ways in which it can be modified and optimized. An optimization for speed is to reuse the tree generated during the previous iteration. This is possible if the local planner is able to reconnect with a node in the old tree for the next iteration. The nodes descending from this node can then be re-used. All other nodes have to be discarded.

Another optimization, which can improve both speed and path-quality, is adding an intermediate node on the edges between all nodes. While this may not seem like a good idea at first, it helps reduce overshoots in the planned path.

As already mentioned, the choice of the local planner and randomization function has great influence on the performance of the algorithm and the characteristics of the final path. The different metrics used in the algorithm can also be used as tuning parameters.

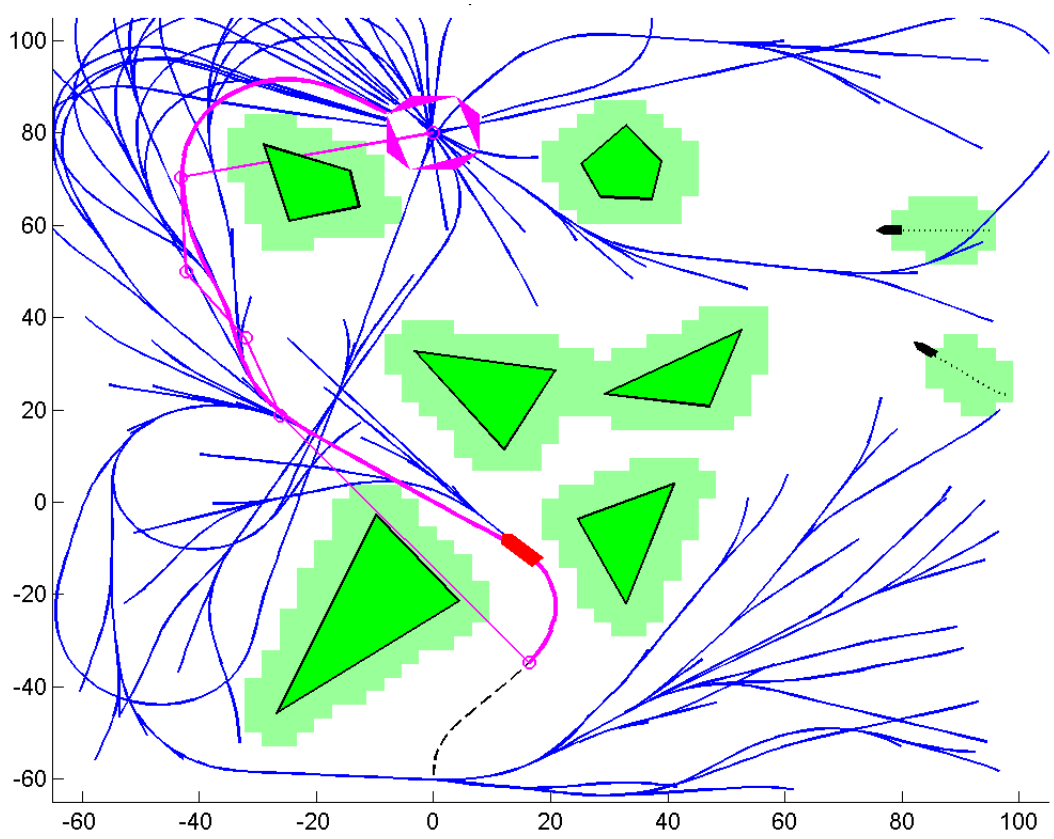


Figure 2.8: The figure shows the RRT algorithm working its magic. The blue tree structure originating at the vessel shows the paths explored by the RRT algorithm. The magenta line was selected as the most optimal of the explored paths ending at the goal. It is not optimal, but not that bad either. The small circles along the optimal path, connected with a thin line, are the coordinates of the states which when connected make up the path.

Evaluation

The RRT method seems promising. It takes vehicle dynamics into account, so we can be sure the path it generates is feasible. It is pretty simple to extend the method to handle collision avoidance with dynamic objects, and by modifying the distance metrics, we can define areas in the state space that are more or less desirable than others.

The cost function employed in the RRT algorithm can be manipulated to favor time optimality, fuel optimality, shortest path, and so on. This gives the method a lot of flexibility. The method is also reasonably fast.

Figure 2.8 shows the RRT method in action. The paths generated by the RRT tend to be a bit *bendy*, but if this ever becomes a problem, smoothing techniques can be employed to enhance them.

There has been a lot of activity around the RRT algorithm, and attempts are still made to improve it. A promising method showing off good results can be found in [44]. The method is well suited for parallel processing systems, which is a property that will become more important in the years to come.

Similar methods

Probabilistic Roadmaps The probabilistic roadmap method [5, 23, 31] does collision avoidance in two different steps: The first step generates the probabilistic roadmap of the configuration space $\mathcal{C}$. This is a time-consuming operation, but for static maps, it only needs to be done once. The second step consists of querying the map, to find a path through $\mathcal{C}$.

The roadmap is made by generating a set of nodes, randomly distributed inside $\mathcal{C}_{free}$. The algorithm then attempts to connect as many of these as possible by edges. Two nodes can be connected if there exists a path between them in $\mathcal{C}_{free}$. As in RRT, a local planner is used for this task. What is left after the first step is a unidirectional graph covering $\mathcal{C}_{free}$. Information about the paths connecting the nodes is discarded.

When querying the roadmap for a path between two states $\mathbf{x}_{init}$ and $\mathbf{x}_{goal}$, the algorithm tries to connect these two to a connected part of the roadmap using the local planner. If it is successful, a search can be performed on the graph to find the shortest route through the graph using some given metric. Finally, the local planner is used to generate a forward path through the nodes.

The method works well for systems with many degrees of freedom, preferably in static environments. For dynamic environments, the RRT method is simpler, and possibly more efficient.

Evolutionary methods Evolutionary methods work by randomly selecting possible solutions to the problem at hand. For the path/motion planning problem, that would mean creating different paths between the two states $\mathbf{x}_{init}$ and $\mathbf{x}_{goal}$. These paths need not be feasible, meaning they can move outside of the configuration space of our vehicle. They may also be unfeasible from a dynamical viewpoint.

The solutions are then given a score based on their properties. Time needed and fuel consumption are among things that may be represented in the score. Infeasible paths take a hit in the scores depending on how infeasible they are.

The last step of the algorithm is to select one or more of the top-performing paths, and then generate new paths by combining these and adding some noise. The new set of paths is now given scores, and the algorithm repeats itself until a path of required quality is found.

The ECoPS system for use on the SEAFOX [10], and a system developed by Zheng and Ito [38, 39] are examples of implementations of evolutionary methods.

2.3.3 Constrained Nonlinear Optimization

The idea of using *numerical optimization* algorithms for controlling dynamic systems is not a new one. Often called *model predictive control* (MPC), the method has attained a widespread usage in recent years, especially in the field of *process control* [36].

Standard form

A constrained nonlinear problem can be stated as:

$$\begin{array}{lll} \min_{\mathfrak{X}\epsilon\mathbb{R}^n} & J(\mathfrak{X}) & \\ \text{s.t.} & & \\ & c(\mathfrak{X}) \quad \leq & \underline{0} \\ & ceq(\mathfrak{X}) \quad = & \underline{0}, \end{array} \tag{2.19}$$

where $f : \mathbb{R}^r \to \mathbb{R}$ is a continuous, smooth function with a well-defined *gradient* and *Hessian*, while $c : \mathbb{R}^r \to \mathbb{R}^p$ and $ceq : \mathbb{R}^r \to \mathbb{R}^q$ must have defined gradients.

An example of an algorithm able to solve this problem is the *SQP* algorithm. If successful, it returns a vector $\mathfrak{X}$ which minimizes f without violating the inequality and equality constraints c and ceq.

Motion planning

Using the framework defined in (2.19), it is possible to define a constrained nonlinear problem whose goal is to navigate a vehicle through a given terrain. There are multiple ways of doing this and the presented problem formulation may well not be the best.

Given a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, define the solution of the optimization problem to be the command sequence to apply to reach the goal $\mathbf{x}_f = (x_f, y_f)$ without hitting any obstacles.

$$\mathfrak{X} = [\mathbf{u}_1 \ \delta t_1 \ \mathbf{u}_2 \ \delta t_2 \ \cdots \ \mathbf{u}_N \ \delta t_N] \,. \tag{2.20}$$

The problem can then be formulated as in [34]:

$$
\begin{array}{lll}
\min_{\mathfrak{X}} & J(\mathfrak{X}, \mathbf{x}(\cdot)) & \\
\text{s.t.} & & \\
 & \begin{bmatrix} x(t_f) - x_f \\ y(t_f) - y_f \end{bmatrix} & = \underline{0} \\
 & h(\mathbf{x}(\cdot)) & \leq 0.
\end{array} \tag{2.21}
$$

$\mathbf{x}(t)$ is calculated by integrating $f(\mathbf{x}, \mathbf{u}_i)$ from the initial condition $\mathbf{x}(0) = \mathbf{x}_0$, where $\mathbf{u}_i$ is taken in sequence from $\mathfrak{X}$ and applied to the system for a time δt_i. The time at the end of the input sequence is given by $t_f = \sum_{i=1}^{N} \delta t_i$. The equality constraint makes sure the vehicle ends up at the goal by demanding that the vehicle coordinates are located at the goal at time t_f. Obstacle avoidance is guaranteed by the inequality using the continuous and differentiable function h, which is positive when $\mathbf{x}(t) \, \epsilon \, \mathcal{C}_{obs}$, and zero or negative when $\mathbf{x}(t) \, \epsilon \, \mathcal{C}_{free}$.

By selecting an appropriate cost function J, important qualities of the resulting path can be controlled. By making J monotonically increasing in for instance t_f, a time-optimal path can be found. Other factors such as fuel-consumption and covered distance are also easy to implement.

Optimality

By now, it may seem like this method is just what doctor ordered, generating an optimal path for us while taking vehicle dynamics into account. The RRT algorithm presented in Section 2.3.2 only managed to generate sub-optimal trajectories. There are, however some significant challenges.

First, the problem defined in (2.21) is generally a non-convex problem. Current methods for solving optimization problems cannot guarantee global solutions when faced with non-convex optimization problems. This means that even though a solution is found to the problem, it is not necessarily the best path.

Feasibility

The second problem is actually finding a feasible path. Cluttered environments may result in the algorithm being unable to find a feasible solution to the problem even though one exists. This makes the method less robust than desired.

There are multiple solutions trying to mend this problem. One is to generate a sub-optimal trajectory using another method, and then using this solution as the initial solution of the optimization problem. The solver may then be able to improve the quality of the sub-optimal path.

Evaluation

Solving the motion planning problem using constrained nonlinear optimization is a good idea. There are several challenges in making this work, and how successful the imple-

mentation will be is heavily dependant on both the problem formulation and the type of solver used.

Currently, a problem with the method is its need for computational power. This will become less noticeable over time when computers get more powerful and the solver algorithms are parallelized to take advantage of the expanding set of computer processors. Another problem is convergence. Since we are dealing with a highly nonlinear problem, convergence towards a global optimum is difficult to achieve with standard nonlinear solvers. The use of *pseudospectral methods* may help solve this problem [7].

A path generated by the optimization algorithm can be seen in Figure 2.9.

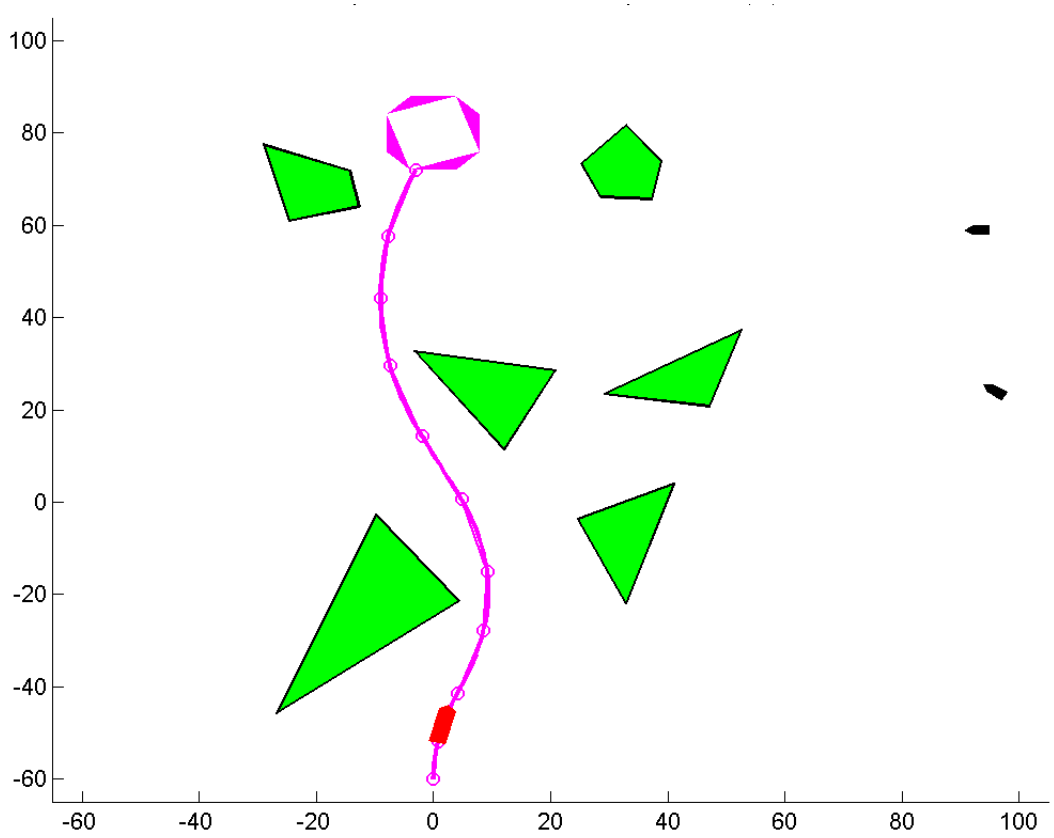


Figure 2.9: This figure shows a path generated by the MPC. One might wonder why the generated path does not end in the center of the goal, given the constraints in (2.21). The reason for this is given in Section 3.2.6. The path is nonetheless of high quality, and the controller brings the vessel to the goal.

2.4 Summary of characteristics

A summary of the properties of the different collision avoidance methods can be found in Table 2.1. As we have just seen, the different methods differ greatly not only in complexity, but also in way-of-thinking. While the VFH method looks for valleys close to the desired direction, the dynamic window approach does a complete search through its possible choices for the next interval, attempting to select the best choice.

The A* algorithm performs a search in the coordinate space of the vehicle without considering vehicle dynamics. It provides the shortest path through the environment, but since it fails to consider dynamics of the vehicle and the motion of other dynamic objects, it is difficult to use it alone. The generated path might not even be feasible.

The RRT method and the constrained numerical optimization method both try to solve the same problem: Given a dynamic model of our vehicle, find a feasible and optimal path to the goal. Both methods fail, however in providing completeness and optimality.

	VFF	VFH	DW	A*	RRT	MPC
Scope	local	local	local	global	global	global
Procedure	Apply virtual force field	Find open valleys	Find optimal velocities	Find L^2 shortest trajectory	Randomized optimization	Nonlinear optimization
Prediction	no	no	yes	no	yes	yes
Optimal	no	no	no	no	no	no
Vehicle dynamics	no	no	yes	no	yes	yes
CPU requirement	very low	very low	low	low	medium to high	high to very high

Table 2.1: A summary of the properties of the different collision avoidance methods.

Chapter 3

Simulator

A simulator has been developed as part of this project to be able to make a quantitative evaluation of the different methods for collision avoidance and motion planning. The simulator was developed in Matlab, as Matlab allows for rapid prototyping, has a good plotting facility, and last but not least has good support for vector and matrix math.

Downsides to Matlab are among others the slow code execution and the lack of modularity, proper object-orientation and references. Some of the most heavily used routines of the simulator were ported to the *C* programming language to make the execution of some of the collision avoidance methods less time consuming.

The simulator was designed to be as open-ended as possible. It has support for an unlimited number of dynamic objects in a simulation at the same time, and it is fairly easy to implement new objects. See Appendix B for more information on how to use and manipulate the simulator.

Integration in the simulator is carried through using the Forward Euler method. The choice fell on this integrator mainly because of its speed. Forward Euler can be quite inaccurate, but the only significant effect it had on the simulations was increasing the minimum turning circle of the vessel. As the control algorithms used the same model, this did not represent a major problem.

3.1 Vessel model

One of the objects in the simulator is *the vessel*. Its task is to simulate a real-life vessel, and is driven by a 3 DOF mathematical model [13] to replicate its dynamics.

The equations of motion

The states of the vessel consists of the vessels generalized coordinates $\boldsymbol{\eta} = [x\ y\ \psi]^T$, and their derivatives $\boldsymbol{\nu} = [u\ v\ r]^T$. (x, y) are the coordinates of the vessel in the world frame, while ψ is the heading of the vessel, also in the world frame. The velocities of the

vessel are on the contrary specified in the frame of the vessel. u is the surge, or forward velocity, while v is the sway, or sideways, velocity. r is the rate of turn, specified in radians per second.

Figure 3.1: The vessel model parameters have been adjusted to roughly approximate the dynamics of the *Viknes 830* vessel. [4]. (Image courtesy of Viknes)

The equations of motion are given as:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{3.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau}. \tag{3.2}$$

$\boldsymbol{R}(\psi)$ is a rotation matrix used to transform the vessel-fixed velocities to the world frame. $\boldsymbol{M}$ is the mass and inertia matrix, $\boldsymbol{C}(\boldsymbol{\nu})$ is a coriolis and centripetal matrix, while $\boldsymbol{D}(\boldsymbol{\nu})$ is a damping matrix. $\boldsymbol{\tau}$ is the generalized force vector acting on the vessel.

This vessel model does not take environmental disturbances such as wind, waves and ocean currents into account. These can be handled efficiently using conventional methods such as wave filtering and integral action [13].

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & v \\ 0 & 0 & -u \\ -v & u & 0 \end{bmatrix} \tag{3.5}$$

$$\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \begin{bmatrix} d_{2u}u|u| + d_{1u}u \\ d_{2v}v|v| + d_{1r}v \\ d_{2r}r|r| + d_{1r}r \end{bmatrix}. \tag{3.6}$$

The vessel has only two actuators, the propeller and the rudder, and is thus *under-actuated* because it has less actuators than degrees of freedom. The actuators are simplified to the point where they are forces that can be controlled directly. This can be justified by assuming that the vessel has low-level control loops for the actuators, adjusting them to match the requested force. The generalized force vector $\boldsymbol{\tau}$ is defined as follows:

$$\boldsymbol{\tau} = \begin{bmatrix} F_x \\ F_y \\ l_r F_y \end{bmatrix}, \tag{3.7}$$

where F_x is the forward force generated by the propeller, F_y is the lateral force generated by the rudder, and l_r is the distance along the x axis of the *point of attack* of the rudder force. The constants used in these equations are summarized in Table 3.1.

Low-level controllers

In most cases, it is not desirable to have higher level collision avoidance algorithms controlling the actuators of the vessels directly. It is much more convenient to let those algorithms control values such as vessel speed and turn-rate directly. This is accomplished by having a set of low-level controllers drive the actuators trying to fulfill the commands from the higher level algorithms. The resulting control structure is visualized in Figure 3.2.

Two options for controlling the vessel are made available to the algorithms. They can either control the velocity and turn-rate of the vessel, using the set-points (u_{sp}, r_{sp}), or they can control the velocity and heading of the vessel through (u_{sp}, ψ_{sp}).

The low-level controllers implemented in the simulator are very simple. First of all, they control the forces of the actuators directly. In an actual application, they would

Parameter	Value	
m	3300	$[kg]$
I_z	1320	$[\frac{kg}{m^2}]$
d_{1u}	16.6	
d_{2u}	8.25	
d_{1v}	9900	
d_{2v}	330	
d_{1r}	330	
d_{2r}	0	
$F_{x,max}$	2310	$[N]$
$F_{y,max}$	28.8	$[N]$
l_r	4	$[m]$

Table 3.1: Parameters for the vessel's equation of motion. The parameters are a rough estimate of the parameters of the *Viknes 830* [4].

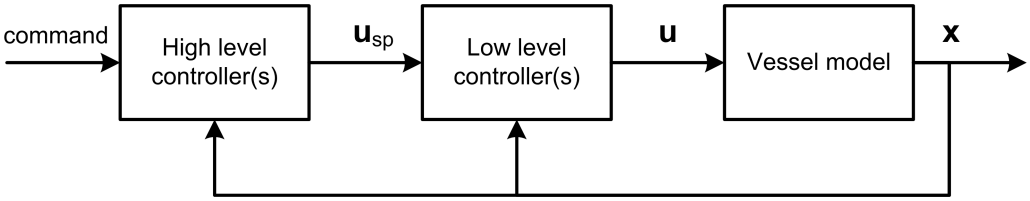


Figure 3.2: This figure shows the control structure employed for the vessel. Starting from the left, high-level controllers receive commands from an external source. Combining this with a state measurement or estimate $\mathbf{x}$, the high-level controllers generate set-points $\mathbf{u}_{sp}$ for the low-level controllers. The low-level controllers then provide the vessel with actuator commands $\mathbf{u}$.

rather control the angle of the rudder and the RPM of the propeller. Second the controllers cancel the dampening in the equations of motion. This is done to avoid any steady-state error without using integral action, and to simplify the response of the vessel.

The controllers employed when using the set-points (u_{sp}, r_{sp}) are:

$$F_x = (d_{1u} + d_{2u}|u|)u + m(rv + K_{p,p}(u_{sp} - u)) \tag{3.8}$$

$$F_y = \frac{1}{l_r}((d_{1r} + d_{2r}|r|)r + I_z \cdot K_{p,r}(r_{sp} - r)). \tag{3.9}$$

When using the set-points (u_{sp}, ψ_{sp}), the same controller is used for F_x, but the controller for F_y is different:

$$F_y = \frac{K_{p,\psi} I_z}{l_r}((\psi_{sp} - \psi) - K_{d,\psi} r). \tag{3.10}$$

The parameters for the controllers are given in Table 3.2.

Parameter	Value
$K_{p,p}$	0.1
$K_{p,r}$	5
$K_{p,\psi}$	5
$K_{d,\psi}$	1

Table 3.2: Parameters for low-level controllers. The parameters were selected using trial-and-error to shape the step-responses of the vessel model.

3.2 Controller implementation

A significant amount of time went into the work of implementing the different collision avoidance methods. In some cases, modifications or additions had to be made to the original methods to make them work efficiently. Relevant implementation details are summarized in the following sections.

3.2.1 Potential Field

The implementation of the potential field method is based on the *virtual force field* variant (see page 9). In the two-dimensional histogram grid, the edges of all obstacles within a given radius are marked with a probability 1. This can be done because our knowledge of the environment is error-free. The total virtual force is then given by:

$$F = F_{\mathbf{x}_d}(\mathbf{x}) + \sum_i F_O(\mathbf{p}_i). \tag{3.11}$$

As the vehicle is underactuated, the virtual force generated by the field cannot be applied directly. Instead, the forces are used to generate commands for the low-level controllers of the vehicle in an attempt to make it move according to the virtual forces: The set-point for heading is set to the direction of the total force, and the set-point for velocity is set to the magnitude of the force vector projected down on the direction vector of the vehicle. The velocity is chosen in that way to make the vehicle brake when the force vector points away from the direction of travel, possibly indicating a threat of collision:

$$\psi_{sp} = \tan^{-1}\left(\frac{F_y}{F_x}\right) \tag{3.12}$$

$$u_{sp} = \left(F^T d\right) d, \qquad d = [\cos\psi \ \ \sin\psi]^T. \tag{3.13}$$

3.2.2 Vector Field Histogram

The VFH method was implemented as described in Section 2.2.2. No significant modifications were required. The commands given by the method are velocities and directions, and can be sent directly to the low-level controllers of the vehicle.

3.2.3 Dynamic Window

The search space V_r (2.17) was implemented as a two-dimensional map, containing samples uniformly scattered across the space $V_s \cap V_d$. Non-admissible velocities $\overline{V_a}$ were implemented by marking cells in the map as inaccessible.

Scaling of the objective function

The performance of the algorithm is very dependent on the scaling of the different components in the objective function. With the original objective function (2.18), the distance component had to be given a high priority to prevent the vehicle from colliding. This gave problems when choices heading away from the goal had a high $\text{dist}(u, \omega)$ value, as the algorithm often selected paths leading away from the goal even if it had an available path leading towards the goal.

To mend this problem, the functions *heading*, *velocity* and *dist* were all scaled to give a value $\in [0, 1]$. For the first two functions, *heading* and *velocity*, a linear scaling was applied with success. The *dist* function required a bit more elaborate approach though.

Curved paths and paths with low velocities cover less distance than straight paths and paths with high velocities. The distance of a path therefore had to be scaled according to some maximum distance. The result is a distance function where paths are not rated according to their distance, but rather their degree of lack of obstacles.

To achieve this, a circle was defined going through the vehicle with center in its forward direction (see Figure 3.3). Arcs are only tested for obstacles inside this circle, and the obstacle-free distance is normalized by dividing it by the total arc length, i.e. the distance along the arc from the vehicle to the circle edge.

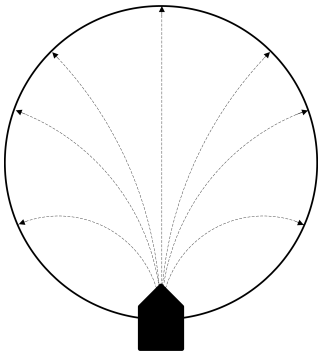


Figure 3.3: Maximum distance planned along an arc is limited by the black circle.

Filtering

The objective function value was not passed through a filter as in (2.18), as measurement noise and inaccurate sensors have not been included in the boat model and information flow. In a live implementation of the dynamic window approach, this should be included.

World representation

The dynamic window approach uses a configuration space representation of the world. Our vessel has three states, resulting in a C-space $\mathcal{C} \epsilon \mathbb{R}^3$. To simplify our algorithms, and reduce memory requirements, which will be important when introducing prediction, an approximation to this C-space is made.

The approximation consists of collapsing the orientation-dimension ψ in $\mathcal{C}$, giving a two-dimensional C-space $\mathcal{C} \epsilon \mathbb{R}^2$. This is the same as pretending the vessel is a circle with center in the origin of the vessel, and a radius such that it covers the entire vessel. Configurations where this circle intersects obstacles are not allowed. The downside to doing this is that we disallow certain configurations that were allowed in the original three-dimensional C-space, possibly closing some of the passages through the original C-space. For slim vehicles, this approximation may be coarse, but normally it will not be a problem.

Figure 3.4 shows the collapsed C-space generated for a set of polygon obstacles.

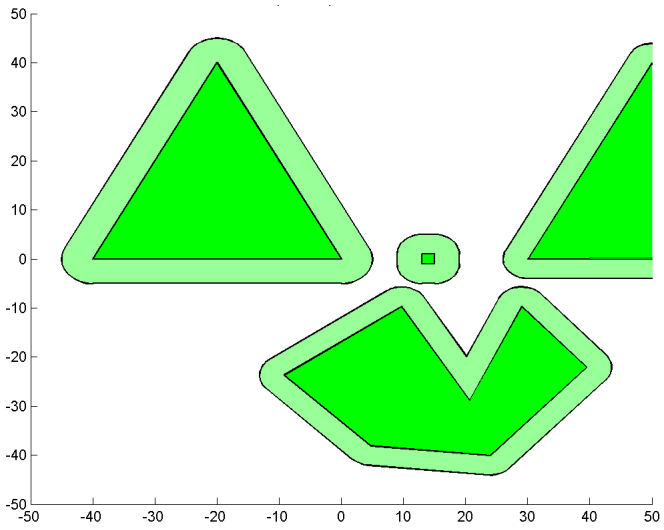


Figure 3.4: This figure visualizes the configuration space representation used in the algorithms. A zone is generated around the polygon obstacles, and if a point representation of our vehicle keeps out of this zone, the vehicle will not collide with the obstacles. The extent of the zones will vary with the size of the controlled vehicle.

Prediction

To make the collision avoidance method more effective, prediction of dynamic obstacles (other vehicles) was introduced. It is assumed that our vehicle knows the coordinates, heading and velocity of all other vehicle in its vicinity. Our controller can then integrate the states of the other vehicles forward in time to get a prediction of their future states.

Our vehicle can normally not predict the future steering commands of the other vehicles. An exception is with cooperating vehicles (see Section 5.5). One could imagine analyzing where the other vehicles had to move with respect to obstructions, etc. and combine it with a measure of uncertainty, but even doing this cannot safeguard us from false predictions. *What if the vehicle suddenly decides to speed up, or suddenly takes a sharp turn?*

A considerably simpler approach was used in this implementation. For all other vehicles, it is assumed that $\psi_{sp} = 0$, and $u_{sp} = u$. Then the vehicle states are integrated using the vessel model of Section 3.1 (in case of a vessel). Our two-dimensional C-space is then expanded to include a *time dimension*, using the predictions of the other vehicles for future times $t > t_0$.

As approximate time is known along the arcs in the dynamic window, these can now be collision tested against the new three-dimensional $\mathcal{C}$, providing us with simple, yet effective prediction in the Dynamic Window algorithm.

3.2.4 A*

Path tracking

The implementation of the A* algorithm was in itself pretty straight-forward, see Section 2.3.1. The way-points of the path generated by the A* algorithm were followed in order, switching to the next way-point when the current is within a radius of 30 metres of the vehicle.

No prediction

The algorithm was implemented in the two-dimensional C-space, without prediction. The reason for this is that the velocity of the vehicle along the path is not known, and thus neither the time along the path, which is necessary for prediction to be meaningful. One could have made a rough estimate, by for instance assuming the velocity of the vehicle to be constant and dividing it by the Euclidean distance along the path. The problem with this, in addition to the constant velocity requirement, is that the A* path consists of sharp 45 ° and 90 ° turns, and is impossible to track accurately by our vehicle. The time along the path will therefore end up being too large, as the path predicts our vehicle will travel longer than in reality.

3.2.5 Rapidly-Exploring Random Tree

The RRT algorithm was, like the Dynamic Window algorithm, implemented in the three-dimensional C-space with prediction (see Section 3.2.3). This can be done in the RRT algorithm, as opposed to A*, because the RRT algorithm integrates the vehicle model and thus has a good estimate of time along any explored path.

Constant velocity

When selecting a new state with $\mathbf{x}_{rand} = \text{RANDOM\_STATE}()$, only the two first states of the vehicle state vector are used, namely x and y, the position of the vehicle. By limiting the choice to these coordinates, we avoid creating target states with difficult headings ψ. We thus have:

$$\mathbf{x}_{rand} = [x_{rand}\ y_{rand}]^T . \tag{3.14}$$

A downside to not including velocity in the new state is that the RRT algorithm loses control of the velocity of the vehicle. It cannot adjust the velocity to avoid danger, only steer clear. The only way for the algorithm to lower its speed of approach is therefore to create a path with superfluous turns and hurls. In a live implementation, the RRT algorithm should be allowed to control velocity.

The Halton sequence

For generating the random coordinates (x, y), the *Halton* sequence was employed. The Halton sequence is a quasi-random sequence, which is nearly uniformly distributed. Even though it appears random, it is really a deterministic sequence, based on prime numbers [59]. It has been claimed that this distribution gives better results than ordinary randomized and biased distributions [48]. The reason for this is probably that the near-uniform distribution allows the algorithm to quickly explore the search space before creating major branches impeding path optimality.

Distance metric

The metric used by the NEAREST_NEIGHBOUR($\mathbf{x}$, $\mathcal{T}$) function is worth mentioning. The closest vehicle state $\mathbf{x} = [x\ y\ \psi\ u\ v\ r]^T$ to $\mathbf{x}_{rand}$, is not just the one where the geometrical distance is the smallest: The heading of the vehicle must also be considered.

The chosen metric was a combination of the Euclidean distance metric and a penalty for how far the heading of the vehicle departed from the direction to $\mathbf{x}_{rand}$:

$$\text{DIST}(\mathbf{x}_{rand}, \mathbf{x}) = (x_{rand} - x)^2 + (y_{rand} - y)^2 + \xi \left(\psi - \tan^{-1} \left(\frac{y_{rand} - y}{x_{rand} - x} \right) \right)^2 . \tag{3.15}$$

The value of ξ in the above equation determines how *bendy* the resulting path will be, but too high values makes it harder for the algorithm to find a feasible path to the goal.

The local controller

The local controller used by CAN_CONNECT($\mathbf{x}_1$, $\mathbf{x}_2$) was designed to be as simple as possible. When trying to connect a state $\mathbf{x}_1$ with $\mathbf{x}_2$, the controller simply steers the heading of the vehicle towards $\mathbf{x}_2$:

$$\psi_{sp} = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right). \tag{3.16}$$

3.2.6 Constrained Nonlinear Optimization

This controller proved to be the most difficult to get working. With the problem formulation defined in Section 2.3.3, the Matlab SQP solver had major problems even finding a feasible solution to the problem. Not finding any feasible solutions, the Matlab solver simply gave up. An attempt was also made with a couple of solvers from the TOMLAB, but the results were the same.

Revised Optimization Problem

After numerous iterations, the revised optimization problem was defined as in (3.17). The constraint demanding that the vehicle end up at the goal was removed. In its place, a weight on the distance from the last vehicle coordinates to the goal was added to the cost function J. The same was done with the constraint function. This means that a path generated by the algorithm may not reach the goal, and it may pass through obstacles, but the solver tries to avoid it, since the costs of doing so are high.

To limit the search space, and ease the task of the optimization routine, constraints were set on the command sequence $\mathfrak{X}$:

$$\begin{aligned} &\min_{\mathfrak{X}} \quad J(\mathfrak{X}, \mathbf{x}(\cdot)) \\ &\text{s.t.} \\ &\qquad \mathfrak{X}_{low} \leq \mathfrak{X} \leq \mathfrak{X}_{high}. \end{aligned} \tag{3.17}$$

These constraints ensure that the solver only looks for $\mathfrak{X}$ in a set of feasible command sequences, and improves both the performance and robustness of the optimization. The cost function was defined as follows:

$$J(\mathfrak{X}, \mathbf{x}(\cdot)) = \alpha \cdot \|\mathbf{e}_f\|_2^2 + \beta \cdot h(\mathbf{x}(\cdot)) + \gamma \cdot t_f + \zeta \cdot \max_{i,j}(\delta t_i - \delta t_j)^2 + \xi \cdot \sum_i r_{sp,i}^2. \tag{3.18}$$

The first element of the cost function puts weight on minimizing $\|\mathbf{e}_f\|_2$, the distance between the goal and the final position of the vehicle; $\mathbf{e}_f = [x(t_f) - x_f,\ y(t_f) - y_f]^T$. This should drive the generated path towards the goal. The second element puts weight on the path being free of obstacles. The obstacle function h will be defined later.

To promote time-optimal paths, the time used by the input sequence is weighted. This weight must be significantly smaller than the weight on goal achievement as not to

compromise this more important objective. To get an even distribution of commands over time, the largest difference in δt_i is weighed. Finally, a weight is put on the commands for angular velocity to prevent excessive turning.

Obstacle penalty function

The first attempt at creating an obstacle penalty function h followed the approach presented in [34]. It relies on creating a continuous function $h_l(x, y)$ defined over the entire map, where the value of the function is low in open areas, and increases exponentially as it enters an obstacle and moves towards its center. The function could unfortunately only be used with convex obstacles, and that was too limiting.

After a series of attempts of making a penalty function, one was found that provided adequate performance. It is not based on sampling the map at discrete coordinates, but rather the intersection between a predicted path and an obstacle. As can be seen in Figure 3.5, the obstacle is cut in half by the predicted path. For the penalty function to guide the path out of the obstacle, its value has to decrease monotonically as the path is pushed out of the obstacle, but increase monotonically when the path is pushed further into the obstacle.

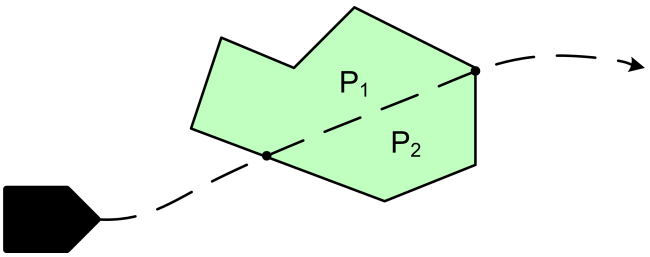


Figure 3.5: The figure shows how a predicted path for the vessel splits the polygonal obstacle, creating two smaller polygons, P_1 and P_2.

Looking at Figure 3.5, two possible choices become apparent. The predicted path splits the obstacle in two. As the path moves out of the polygon, the area of the smallest part decreases monotonically. The part of the obstacle circumference remaining on the smallest part also decreases monotonically.

Going with a function of the circumference of the smallest part has two advantages over using its area. First, the gain of the area function depends on the thickness of the obstacle, and escaping from a thin obstacle becomes difficult. The circumference is less plagued by the change in gain. Second, using the circumference normally provides a larger decrease in penalty when exiting the obstacle than the area function, encouraging the solver to get the predicted path entirely free of the obstacles.

Initial guesses

To improve the performance of a numerical optimization, the solver can be provided with an initial guess $\mathfrak{X}_0$ at the solution. For nonlinear problems this is especially important, as

it greatly increases the possibility of finding the globally optimal solution. Appreciating this fact, the MPC controller carries out optimization in two passes:

1. Find an input sequence taking the vessel to the goal, ignoring collisions. This is done by excluding obstacles from the optimization. The resulting input sequence will provide the initial guess for the pass.

2. Use the initial guess from the previous pass, and optimize. This time, obstacles are included in the optimization.

The two-step optimization improves the chances of finding a good path to the goal, thus increasing robustness. The weights used for the two steps are given in Table 3.3.

Parameter	**Pass 1**	**Pass 2**
Reach goal: α	1	1
Avoid obstacles: β	0	1
Time optimality: γ	0.05	0.0001
Even distribution: ζ	0.01	0.001
Smooth turns: ξ	0	0

Table 3.3: This table summarizes the parameters used in the two optimization passes. The first pass uses no weight on obstacle avoidance, and thus has no regard of these. Note that none of the passes use the *smooth turns* weight ξ. This is because the solver already provided smooth turning curves, and the solver already had enough of a problem finding a solution to the problem given by these parameters.

World representation

The world representation used for this controller is the same as the one for the RRT and Dynamic Window controller. It is based on the three-dimensional configuration space with time as one of the dimensions, providing prediction of the movements of other dynamic objects in the environment.

3.2.7 Hybrids

It was desired to match up all the global and local methods to see which hybrid method would provide the best results. With a total of three global and three local methods, this resulted in 9 unique hybrid methods:

- *A** with *Potential Field*

- *A** with *Vector Field Histogram*

- *A** with *Dynamic Window*

- *RRT* with *Potential Field*

- *RRT* with *Vector Field Histogram*

- *RRT* with *Dynamic Window*

- *MPC* with *Potential Field*

- *MPC* with *Vector Field Histogram*

- *MPC* with *Dynamic Window*

The connection between the global and the local methods was made by telling the local controllers to use the path generated by the global methods as guidance. A path-tracker was developed to provide the local methods with goal coordinates progressing along the path. Velocity information is not passed between the global and local methods. This should be implemented in a live implementation, but it is not a significant problem here.

Chapter 4

Evaluation

In this chapter, an attempt is made to evaluate the different methods for collision avoidance and motion planning. The most widely used way of evaluating collision avoidance systems is numerical simulation. In some cases, properties such as convergence can be proven, as shown in [19], but simulation must still be employed to assess the different transient qualities of a given method.

The local controllers are run once every $T_{local} = 0.5$ seconds, and the global controllers are run once every $T_{global} = 10$ seconds. This is also the case with the hybrid controllers, consisting of one local and one global controller.

4.1 Measures of performance

The collision avoidance methods are evaluated according to how well they perform the task of guiding the vehicle towards the goal. The following measures are used to evaluate the performance of each method.

- *Success*
  The rate of success is determined by whether or not the method managed to guide our vehicle to the goal, avoiding obstacles.
- *Reasonability*
  Would a human have selected this path through the environment? How good is the path?
- *Distance* (S)
  The distance traveled by the vehicle in metres. The shorter distance the better.
- *Average velocity* (V)
  Average velocity of the vehicle in metres per second. Higher velocity makes the vehicle able to complete the scenario in less time, which is a good thing.
- *Energy use* (E)
  Total energy applied to the vehicle model over the course of the scenario; given

in joules. None of the algorithms are optimized for energy use, but this is still a good measure of whether or not the controller makes the vehicle do unnecessary maneuvers.

- *Time to complete* (T_w)
  World-time needed to complete the scenario; given in seconds.

- *CPU time* (T_c)
  Calculation time used by the control algorithm; given in seconds. Less is better, as the algorithms can then be implemented in systems with less resources.

As the RRT method is randomized, and will deliver varying solutions to the collision avoidance problems, all values reported for the RRT method and hybrids based on it are given as a mean of $N = 100$ runs. The mean is an estimate of the expected value of the results. For a set of N values y_i, the mean is given as:

$$\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{4.1}$$

The estimate of the standard deviation is also given for these methods. It is given by s_y, and while there are multiple ways of calculating it, the following way gives an unbiased estimate [53]:

$$s_y^2 = \frac{1}{N-1} \sum_{i=1}^{N} (y_i - \bar{y})^2. \tag{4.2}$$

To make the calculation time T_c comparable, all simulations were run on the same computer. Its specifications are given in Table 4.1.

Processor	Intel Core 2 Duo E6700 2.67GHz @ 3.35 GHz
Memory	2 x Corsair TWIN2X 6400C4 DDR2, 2048MB CL4
Motherboard	Asus P5W DH Deluxe
Operating System	Windows Vista Business
Matlab Version	R2007a

Table 4.1: Specifications of the computer the simulations were run on.

For most of the controllers, the *rate of success* is given as either *success* or *failure*. The exceptions are the controllers using the RRT algorithm, which their success rate given as their percentage of success over the N runs.

4.2 Scenarios

Five different scenarios were created to illustrate the different strengths and weaknesses of the collision avoidance methods. The scenarios range from simple ones without any dynamic objects, to complex scenarios with interaction between the dynamic objects in the scene.

4.2.1 Scenario 1: Walk in the park

As the title suggests, this is the easy scenario. The environment is free of local minima where the local methods can get stuck, and there are no dynamic obstacles present. The goal with this scenario is to show that all the presented collision avoidance methods can be effective given a simple environment.

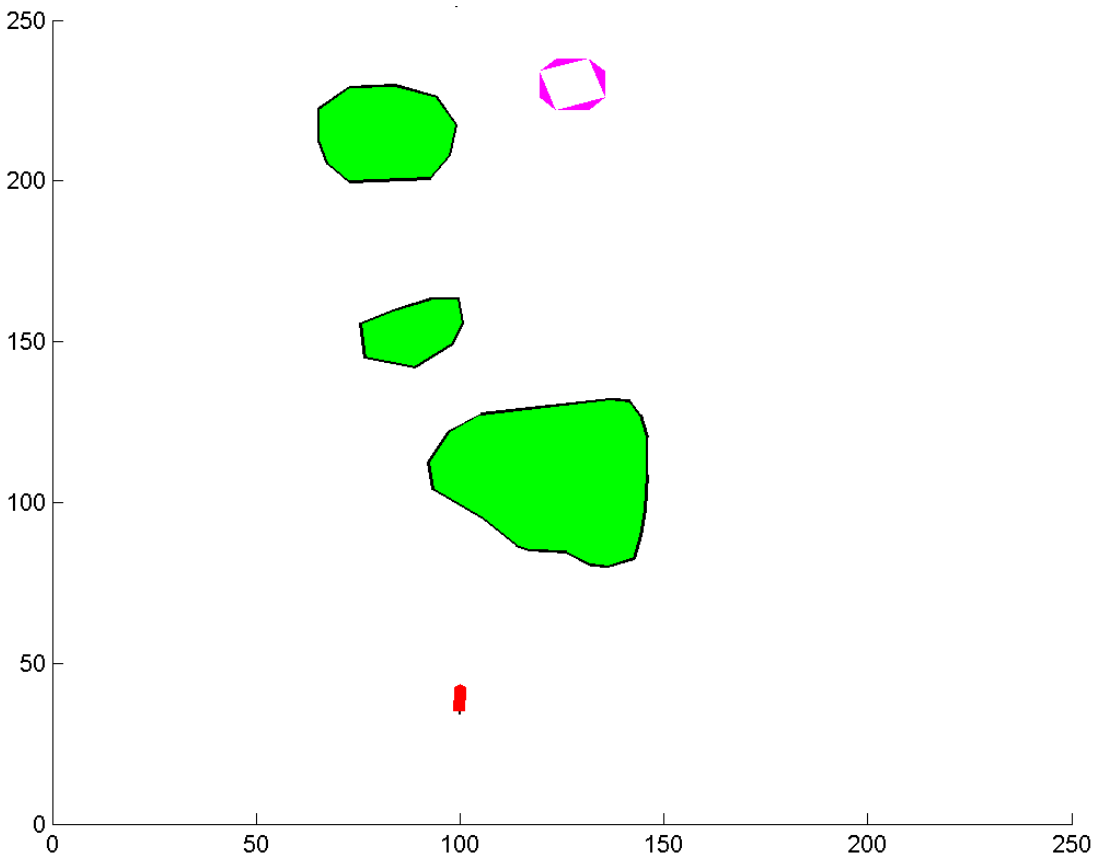
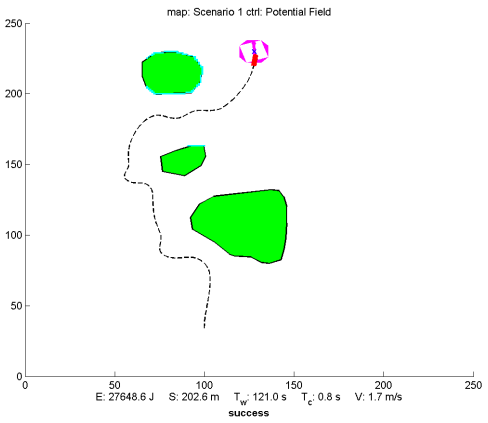
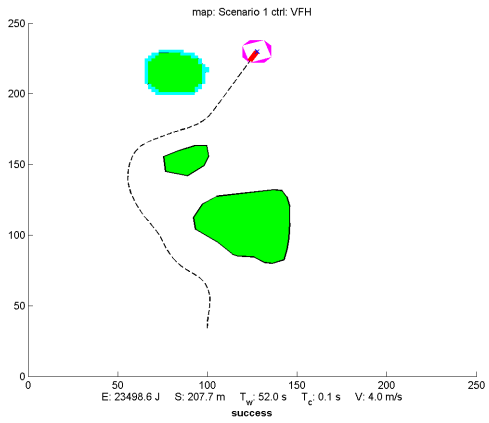


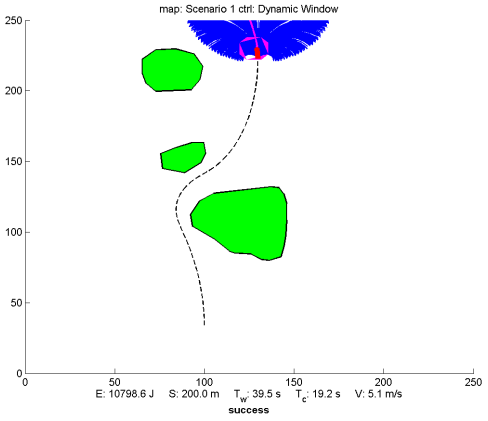
Figure 4.1: The environment of the first scenario consists of three islands. Passage is possible between all of the islands, so there is a wealth of possible solutions. The shape in the top of the figure is the goal, and the vessel can be seen at its initial position in the bottom of the figure.



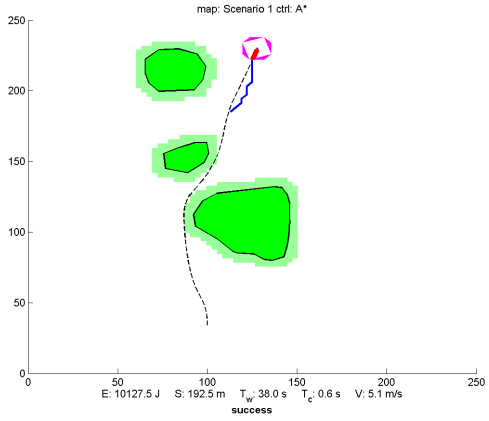
(a) Scenario 1 - VFF (success)



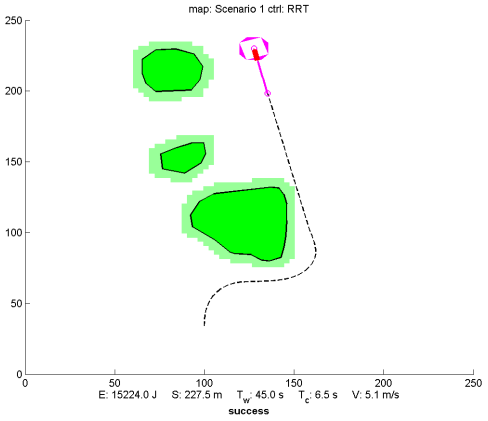
(b) Scenario 1 - VFH (success)



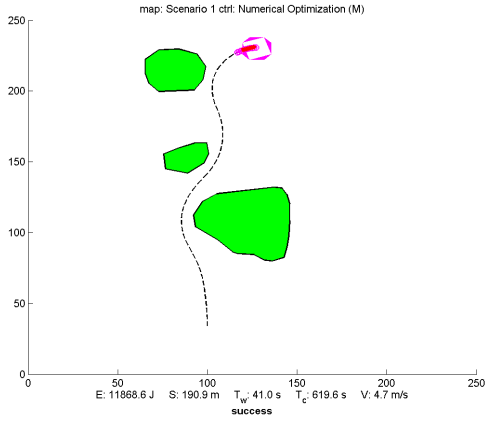
(c) Scenario 1 - DW (success)



(d) Scenario 1 - A* (success)

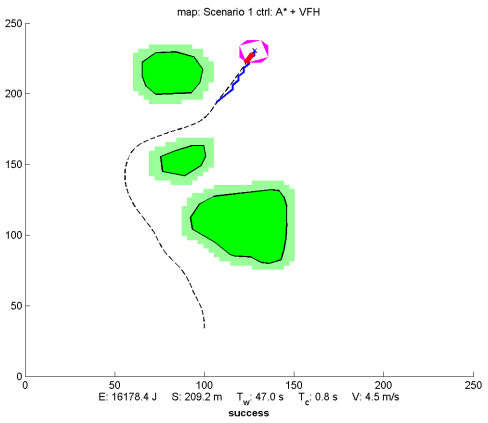


(e) Scenario 1 - RRT (100%)

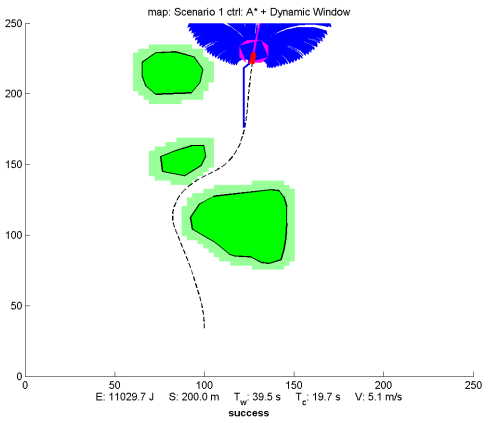


(f) Scenario 1 - MPC (success)

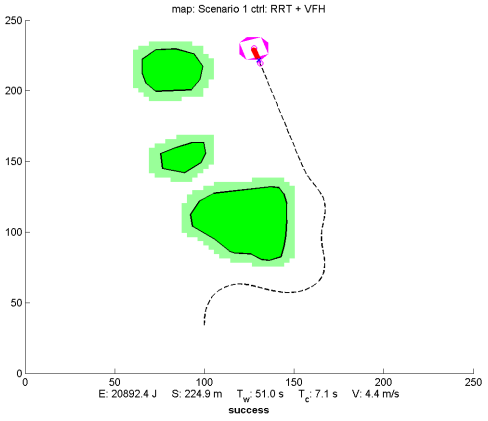
Figure 4.2: Local and global controllers separately navigating through Scenario 1.



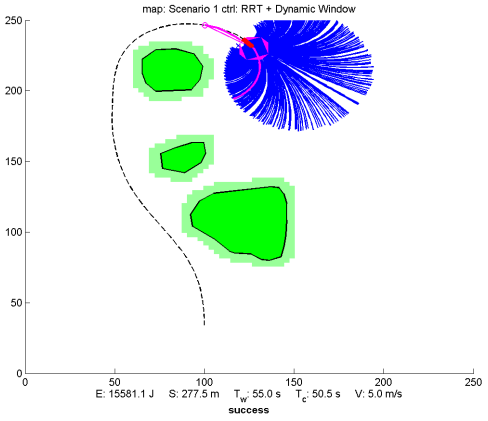
(g) Scenario 1 - A* + VFH (success)



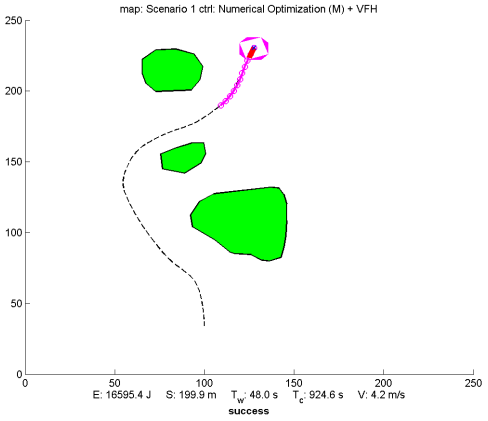
(h) Scenario 1 - A* + DW (success)



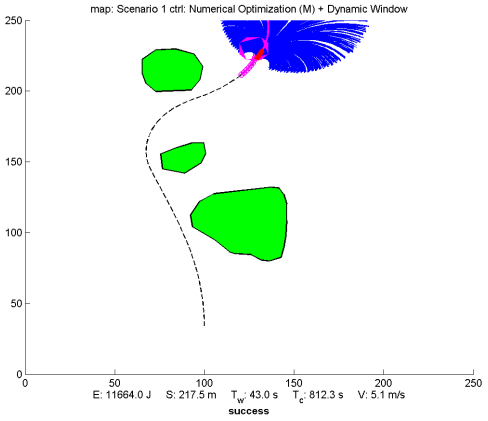
(i) Scenario 1 - RRT + VFH (100%)



(j) Scenario 1 - RRT + DW (95%)



(k) Scenario 1 - MPC + VFH (suc.)



(l) Scenario 1 - MPC + DW (suc.)

Figure 4.2: Hybrid controllers navigating through Scenario 1.

Results from Scenario 1

The results from Scenario 1 can be viewed in Table 4.2. As expected, the controllers managed well in this scenario. The only one that didn't make it was the combination *A*+VFF*, where the A* algorithm tried to pull the vessel through a passage too tight for the VFF algorithm. The vessel never crashed, but went into a limit-cycle, running around and around at the mouth of the passage. The poor performance of VFF can also be seen in Figure 4.2(a), where it generates a slow and bendy trajectory. Because VFF is not seen as a viable option, the plots of hybrid controllers using VFF are omitted.

The reason for the slow speed when using the VFF controller is that increasing the speed would require a greater repulsive force from the obstacles. This greater repulsive force would give the controller problems leading the vessel through narrow passages.

The MPC found the shortest path through the environment, A* used the least amount energy and traversed the environment fastest, and VFH required the smallest amount of computational power.

A* is successful here because it tries to force the vessel along the shortest path through the environment (as seen by the A* algorithm). Given a more complex environment, this could have caused the vessel to crash, as the vessel blindly tries to follow the path. This time however, the A* algorithm was lucky, and got very good results.

The main reason the VFH has a lower computation time than VFF is that the latter algorithm in these simulations used a map resolution of double resolution, resulting in four times more work. When the VFF also used more than double the time of VFH to complete the scenario, the difference in CPU time becomes obvious.

As explained in Section 4.1, the RRT controller will perform differently from simulation to simulation. This is acknowledged by reporting the average results over 100 simulations for all controllers involving RRT. This can be seen in Table 4.2. For every performance value, the standard deviation is presented in the parenthesis, giving an indication of the range of the results. The figures displayed for the RRT combinations are also affected by this: They are only examples of possible trajectories through the environment.

The RRT algorithm and its hybrids are underachieving in this scenario. This may be a result of it having problems finding feasible paths through the narrow passages between the islands. Each run of the RRT algorithm consisted of $N = 500$ iterations. Increasing this value would have improved the chances of the RRT algorithm finding better paths, although at the cost of a higher computation time.

	Success	Quality	Distance (S)	Avg. Vel. (V)	Energy (E)	Time (T_w)	CPU Time (T_c)
VFF	yes	poor	202.6	1.7	27.6k	121.0	0.8
VFH	yes	good	207.7	4.0	23.5k	52.0	**0.1**
DW	yes	good	200.0	5.1	10.8k	39.5	19.2
A*	yes	good	192.5	5.1	**10.1k**	**38.0**	0.6
RRT	100%		241.8 (28.6)	5.1 (0.0)	15.5k (2.5k)	47.9 (5.7)	13.3 (5.8)
MPC	yes	average	**190.9**	4.7	11.9k	41.0	619.6
A*+VFF	failed						
A*+VFH	yes	good	209.2	4.5	16.2k	47.0	0.8
A*+DW	yes	good	200.0	5.1	11.0k	39.5	19.7
RRT+VFF	97%		218.5 (34.2)	2.3 (0.1)	16.6k (5.5k)	97.3 (18.9)	41.9 (21.5)
RRT+VFH	100%		227.4 (29.9)	4.4 (0.3)	18.2k (2.5k)	51.9 (5.3)	18.3 (6.1)
RRT+DW	95%		262.7 (68.7)	5.0 (0.1)	16.3k (7.5k)	52.1 (13.7)	41.9 (13.0)
MPC+VFF	yes	bad	**254.6**	2.0	**33.0k**	**126.0**	**1409.3**
MPC+VFH	yes	good	199.9	4.2	16.6k	48.0	924.6
MPC+DW	yes	good	217.5	5.1	11.7k	43.0	812.3

Table 4.2: A summary of the results from Scenario 1. The quality of the paths generated by RRT and its hybrids vary, and cannot be based on the results shown in Figure 4.2. Note the average velocities reported. All the controllers had 5 $fracms$ as their upper limit, but still some of the methods have an average velocity that exceeds this.

4.2.2 Scenario 2: Local minimum

The second scenario was designed to demonstrate one of the main differences between the global and local methods. As previously mentioned, all local methods have a tendency of getting stuck at local minima in the environment. As we will soon see, this tendency makes it difficult for these methods to traverse more complicated environments.

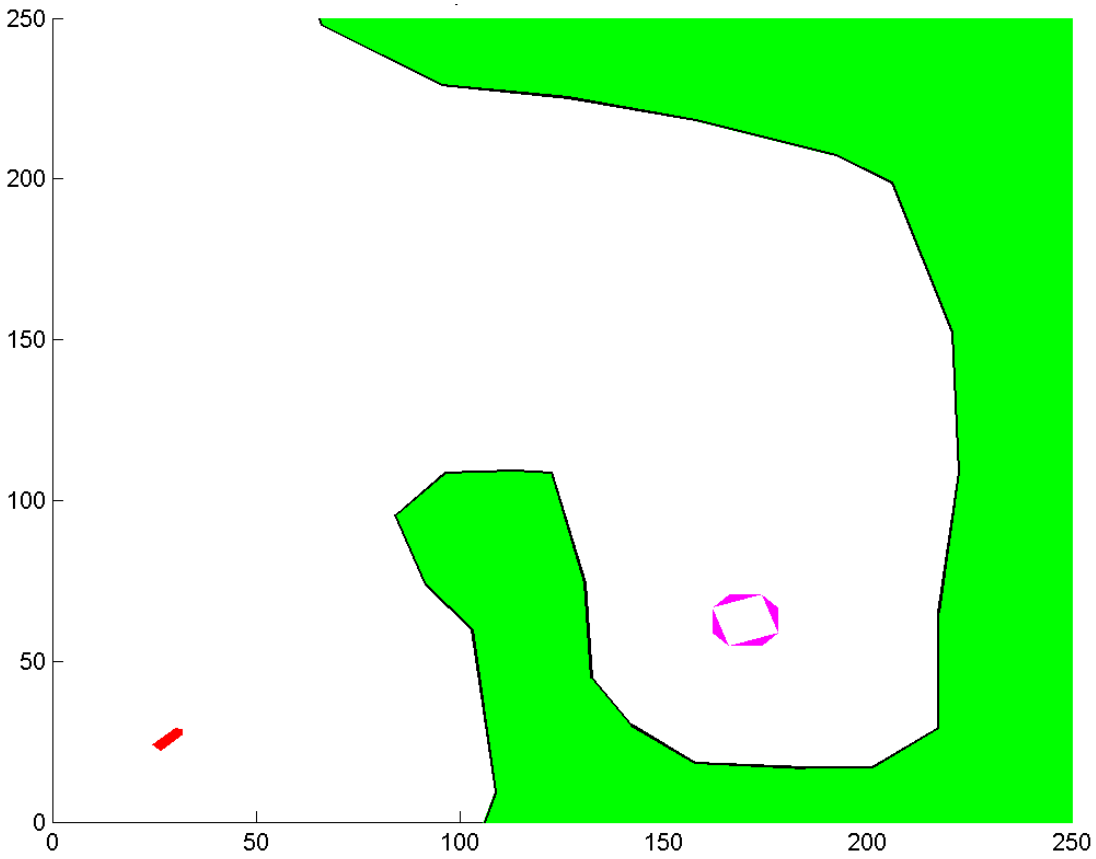
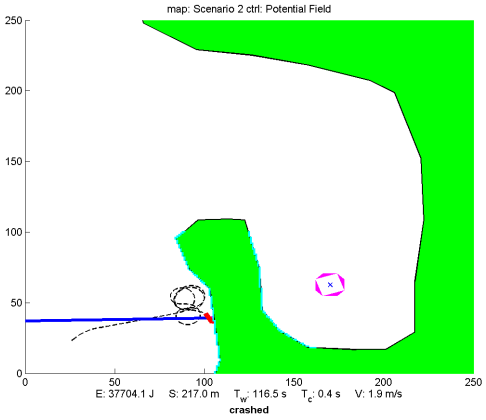
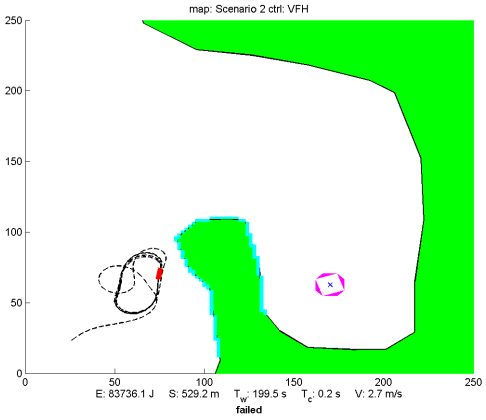


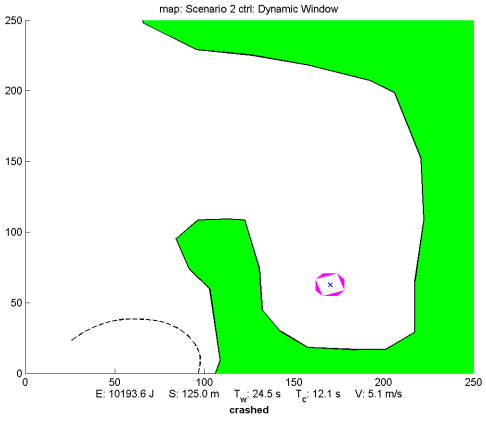
Figure 4.3: The second scenario takes place in this seemingly simple environment. The challenge is the tongue of land stretching north, which creates a local minimum in the environment.



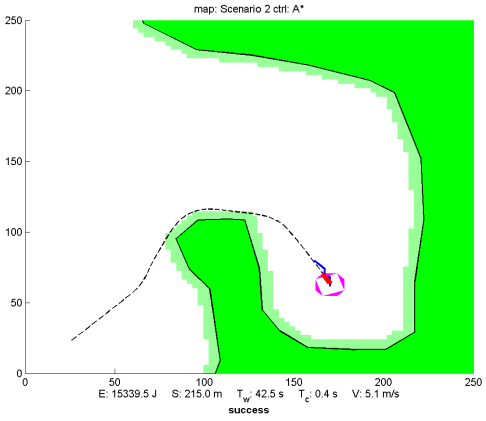
(a) Scenario 2 - VFF (failure)



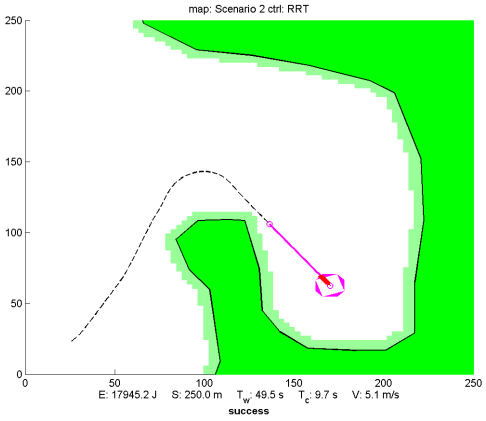
(b) Scenario 2 - VFH (failure)



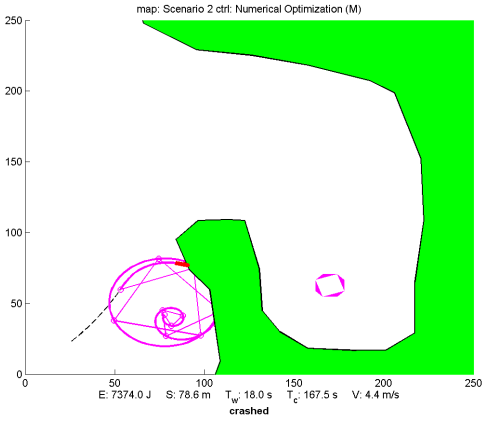
(c) Scenario 2 - DW (failure)



(d) Scenario 2 - A* (success)

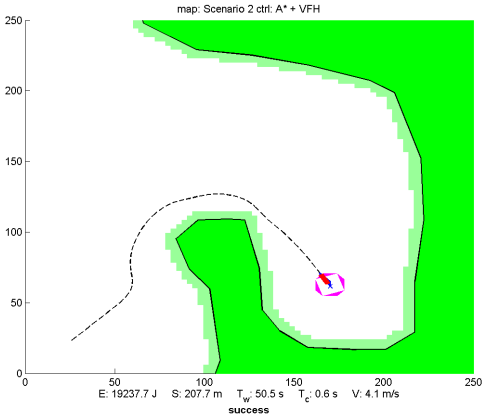


(e) Scenario 2 - RRT (100%)

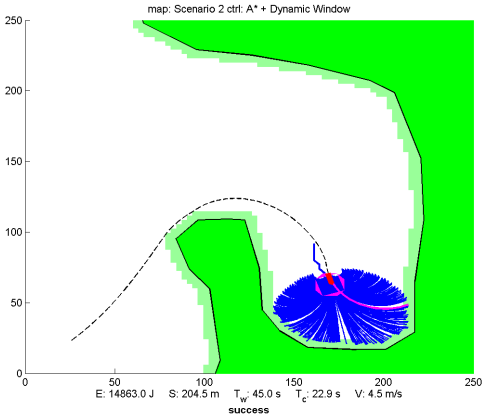


(f) Scenario 2 - MPC (failure)

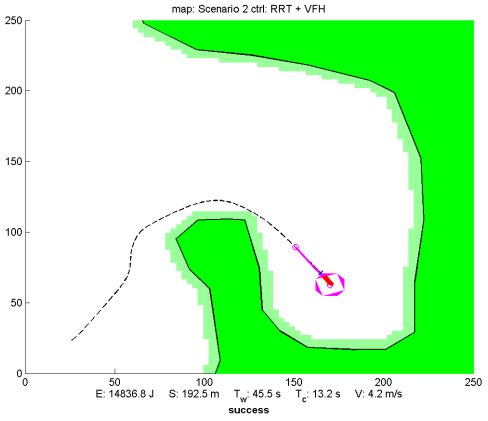
Figure 4.4: Local and global controllers separately navigating through Scenario 2.



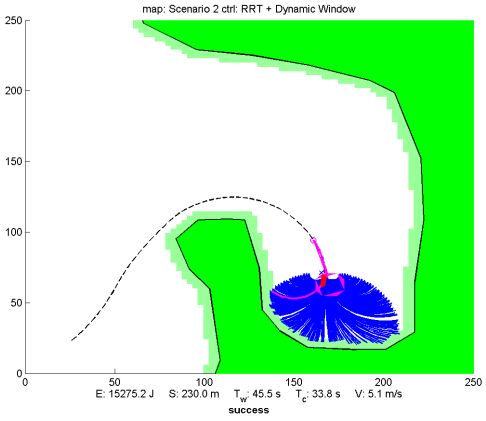
(g) Scenario 2 - A* + VFH (success)



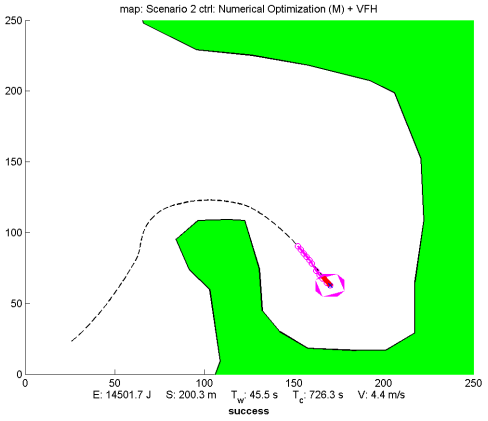
(h) Scenario 2 - A* + DW (success)



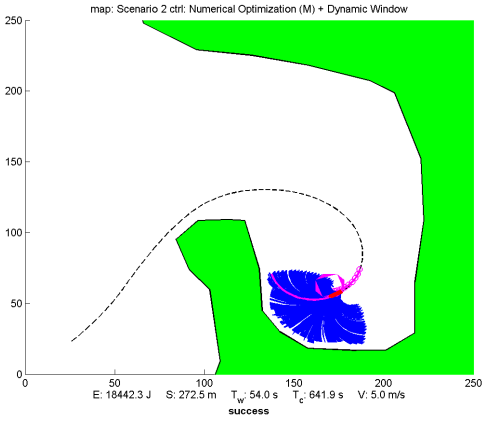
(i) Scenario 2 - RRT + VFH (99%)



(j) Scenario 2 - RRT + DW (100%)



(k) Scenario 2 - MPC + VFH (suc.)



(l) Scenario 2 - MPC + DW (suc.)

Figure 4.4: Hybrid controllers navigating through Scenario 2.

Results from Scenario 2

The major complaint with local collision avoidance methods has always been their problem dealing with local minima in an environment. This is made very clear by the results from this scenario.

All of the local methods fail in guiding the vessel to the goal. The problem is the tongue of land stretching out in front of the goal and bending outwards, effectively trapping the vessel when the local controllers are used. On the other hand, the global methods and the hybrids perform very well in this scenario. This is due to their ability to use information about the entire environment when deciding where to go.

The MPC has major problems in this scenario. This can seem a bit strange when the hybrids based on MPC are successful, as the local methods provide no help in getting out of the local minima. The reason for the apparent contradiction is that the result of the nonlinear optimization is *very* dependent on the initial conditions it is presented with. The VFH and Dynamic Window controllers make the vessel handle differently, changing the initial conditions of the nonlinear optimization problem. This should not be seen as the VFH and DW controllers making it easier to solve nonlinear optimization problems. Chance made the results end up like they did.

The results of the MPC in this scenario is a manifestation of the convergence problem described in Section 2.3.3. The controller is unable to find the optimum solution to (3.17), and even ends up presenting the vessel control systems with a path leading to collision. As previously mentioned, the emergence of new methods for solving constrained nonlinear systems, such as *pseudospectral methods* [7] may help solve this problem.

	Success	Quality	Distance (S)	Avg. Vel. (V)	Energy (E)	Time (T_w)	CPU Time (T_c)
VFF	failed						
VFH	failed						
DW	failed						
A*	yes	good	215.0	5.1	15.3k	**42.5**	**0.4**
RRT	100%		231.1 (10.2)	5.1 (0.0)	16.6k (1.2k)	45.7 (2.0)	10.0 (0.8)
MPC	failed						
A*+VFF	yes	good	**196.8**	2.1	13.6k	93.5	1.1
A*+VFH	yes	good	207.7	4.1	19.2k	50.5	0.6
A*+DW	yes	good	204.5	4.5	14.9k	45.0	22.9
RRT+VFF	100%		199.5 (2.5)	2.3 (0.0)	**12.3k** (0.9k)	88.2 (1.5)	24.5 (2.3)
RRT+VFH	99%		213.1 (20)	4.5 (0.3)	16.1k (2.4k)	47.0 (4.1)	11.0 (1.4)
RRT+DW	100%		237.7 (9.3)	5.1 (0.0)	15.8k (0.7k)	47.0 (1.9)	34.8 (1.3)
MPC+VFF	yes	bad	**324.0**	1.9	**31.3k**	**168.0**	**1893.5**
MPC+VFH	yes	good	200.3	4.4	14.5k	45.5	726.3
MPC+DW	yes	good	272.5	5.0	18.4k	54.0	641.9

Table 4.3: A summary of the results from Scenario 2.

4.2.3 Scenario 3: Blocked path

The third scenario ups the ante by introducing another vessel to the equation. The vessel will at some time block the path to the goal. This presents our methods with a challenge, as paths planned early in the scenario may eventually lead to a collision with the other vessel. Adaptability and prediction will be important here. If the algorithms can predict that the vessel is about to block the path, they may take this into their considerations early on. Else, they must rely on feedback and try to adapt when they later encounter the vessel.

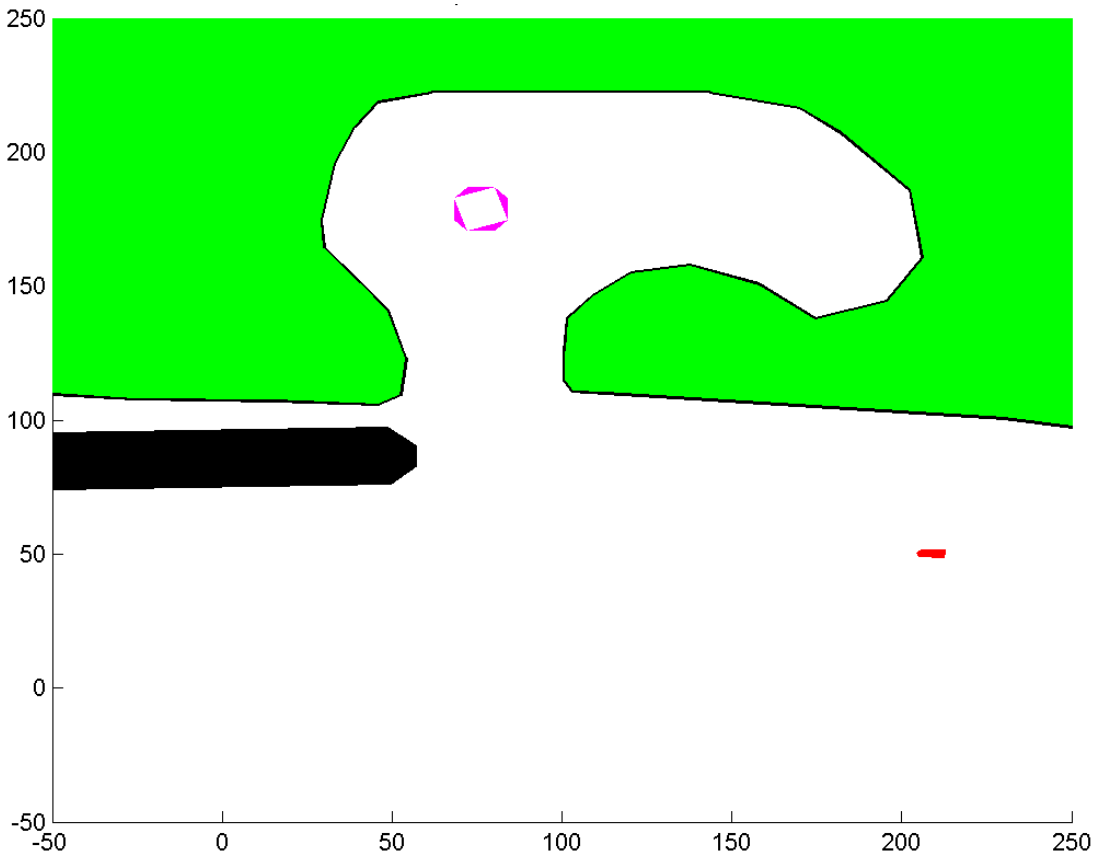
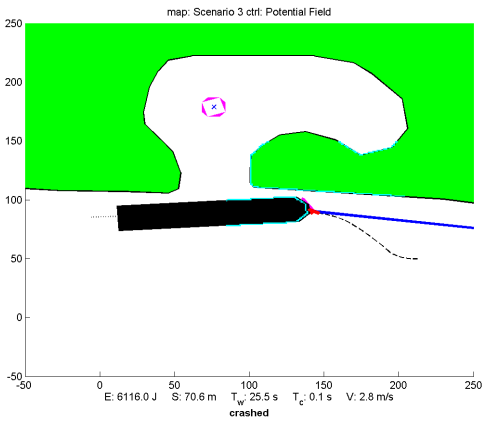
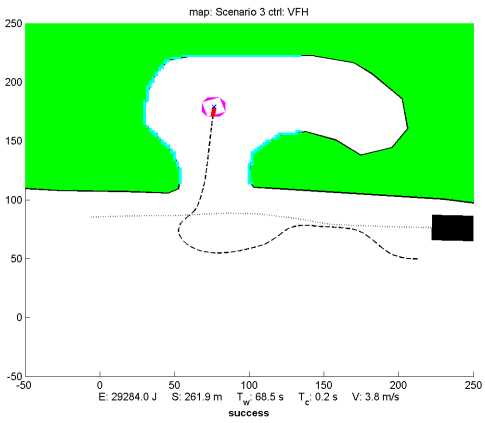


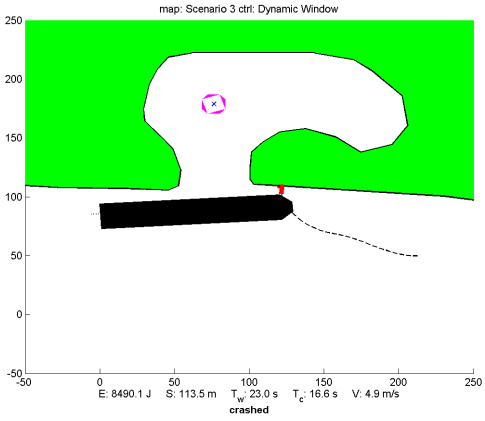
Figure 4.5: The environment in this scenario does not by itself present any major challenges. The static scenery is free of local minima, and should thus be navigable by all the methods introduced in Section 2.1. The complication is the large vessel coming from the left, traveling east across the map. It will block the entrance to the goal as our vessel approaches it.



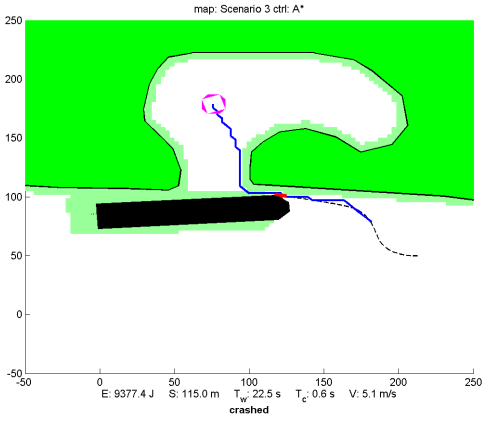
(a) Scenario 3 - VFF (failure)



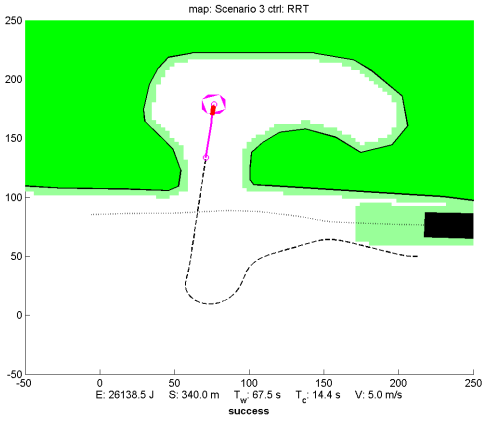
(b) Scenario 3 - VFH (success)



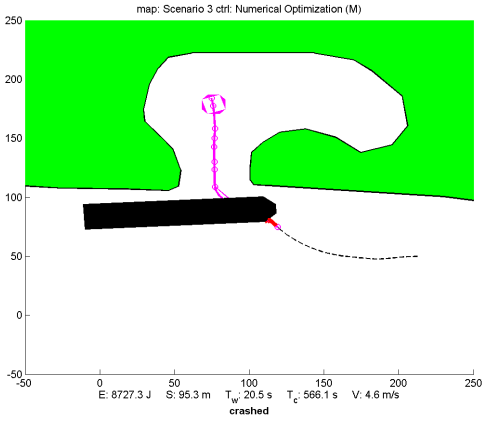
(c) Scenario 3 - DW (failure)



(d) Scenario 3 - A* (failure)

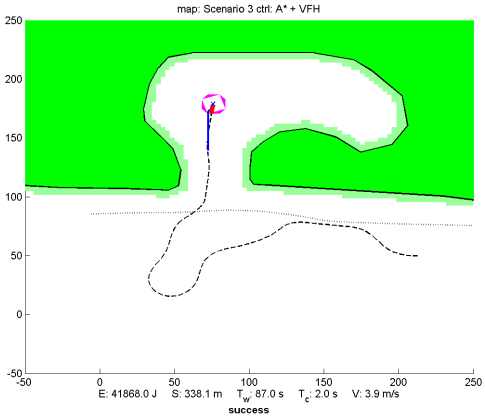


(e) Scenario 3 - RRT (100%)

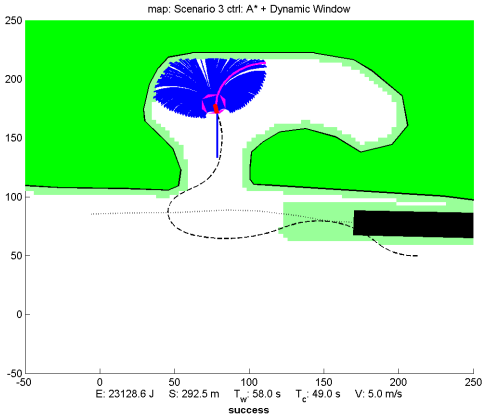


(f) Scenario 3 - MPC (failure)

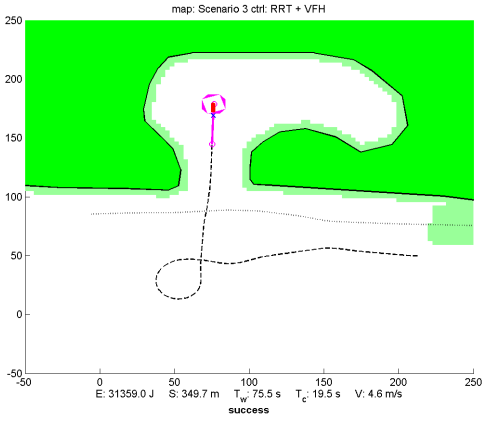
Figure 4.6: Local and global controllers separately navigating through Scenario 3.



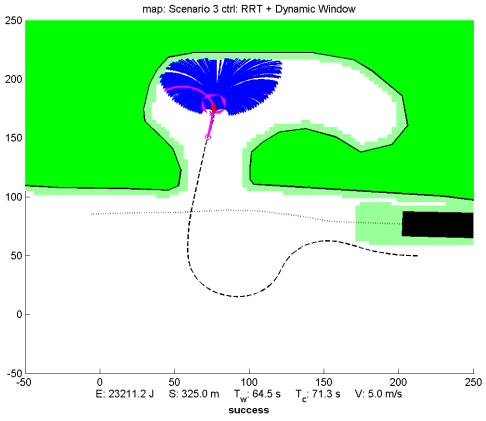
(g) Scenario 3 - A* + VFH (success)



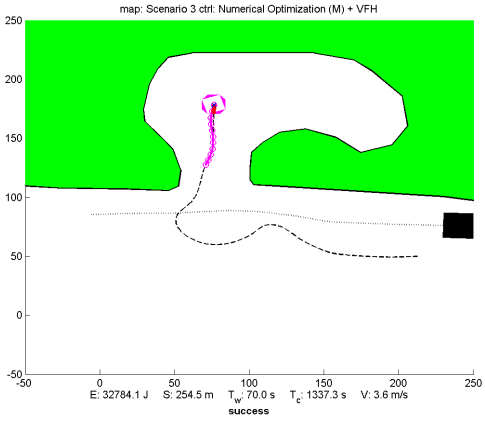
(h) Scenario 3 - A* + DW (success)



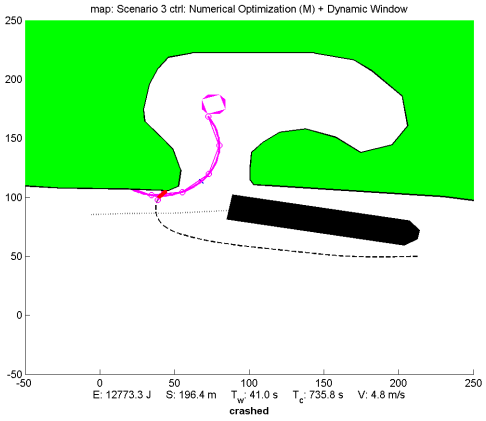
(i) Scenario 3 - RRT + VFH (99%)



(j) Scenario 3 - RRT + DW (96%)



(k) Scenario 3 - MPC + VFH (suc.)



(l) Scenario 3 - MPC + DW (failure)

Figure 4.6: Hybrid controllers navigating through Scenario 3.

Results from Scenario 3

The third scenario poses a serious challenge to our algorithms. When controlled by VFF, the vessel reacts too slowly to get out of the way of the larger vessel, and collides. The Dynamic Window controller gets the vessel trapped in a local minimum between the headland and the other vessel, also causing a collision.

The only local controller finishing the scenario was the VFH controller. The reason was the tendency of VFH to keep below its maximum velocity whenever facing an obstruction (the large vessel). This allowed the larger vessel to come closer, effectively removing the local minima trapping the DW controlled vessel.

Of the global controllers, the RRT controller was the only one to complete the scenario. In contrast to A*, RRT could predict that the large vessel would close off the entrance to the goal, and guide our vessel safely around it with an impressive 100% success rate. The MPC controller also has the ability of prediction, but again failed to find an optimal path.

As seen in Figure 4.6(g) and Figure 4.6(h), the hybrid controllers manage to lead the vessel to the goal even though the path generated by A* would lead to failure. This showcases one of the strengths of the hybrid approach, where the local algorithms can override the global algorithms to maintain a safe course.

This scenario also shows why the A* algorithm cannot be used by itself. It certainly has the speed to get effective feedback, but by not being able to predict what will happen along its path, it risks leading the vessel into a situation where a crash is unavoidable.

	Success	Quality	Distance (S)	Avg. Vel. (V)	Energy (E)	Time (T_w)	CPU Time (T_c)
VFF	failed						
VFH	yes	good	261.9	3.8	29.3k	68.5	**0.2**
DW	failed						
A*	failed						
RRT	100%		317.1 (32.1)	5.0 (0.0)	24.3k (3.7k)	62.9 (6.4)	14.4 (1.5)
MPC	failed						
A*+VFF	failed						
A*+VFH	yes	poor	338.1	3.9	**41.9k**	87.0	2.0
A*+DW	yes	good	**200.0**	5.0	23.1k	**58.0**	49.0
RRT+VFF	100%		202.2 (4.0)	2.1 (0.1)	**16.6k** (2.1k)	95.8 (4.7)	2.1 (0.1)
RRT+VFH	99%		332.8 (48.3)	4.4 (0.2)	31.8k (5.6k)	75.2 (10.2)	22.1 (5.0)
RRT+DW	96%		**365.5** (85.3)	5.0 (0.1)	26.7k (7.9k)	73.0 (17.2)	82.0 (19.4)
MPC+VFF	yes	bad	252.2	2.1	21.1k	**117.5**	**1956.4**
MPC+VFH	yes	good	254.5	3.6	32.8k	70.0	1337.3
MPC+DW	failed						

Table 4.4: A summary of the results from Scenario 3.

4.2.4 Scenario 4: Hostility

An interesting application for autonomous vehicles has always been missions in hostile environments. In this scenario, a total of five hostile vessels are introduced. Their objective is to hit our vessel, and they will always do their best to accomplish this. Their control algorithm calculates how far ahead of our vessel they have to aim to hit our vessel, making them quite capable.

To be successful in this scenario, a control algorithm must be able to react quickly. Our vessel does not know how the other vessels are controlled, so long-time prediction of their behavior will be erroneous, giving global algorithms depending on this information quite a challenge.

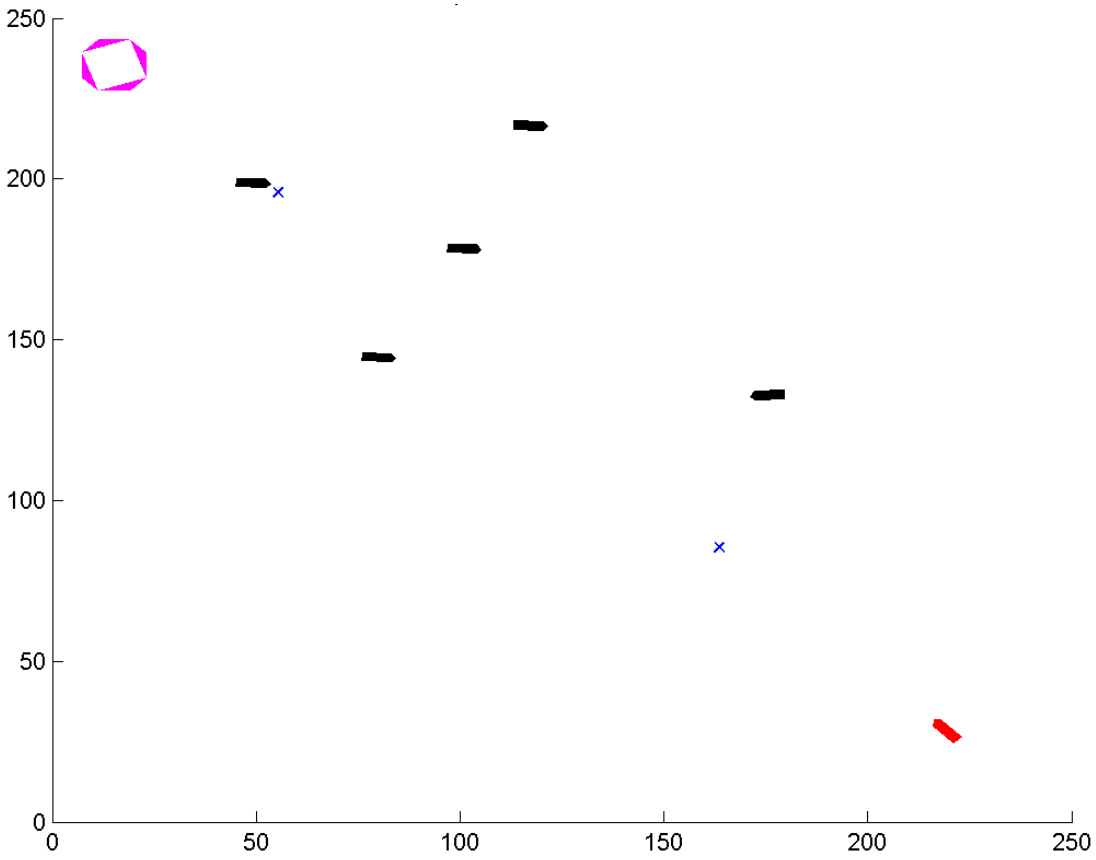
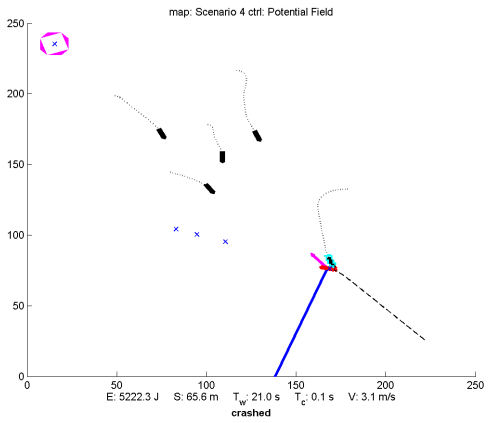
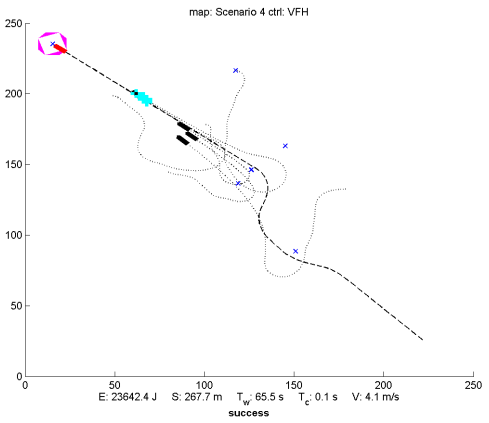


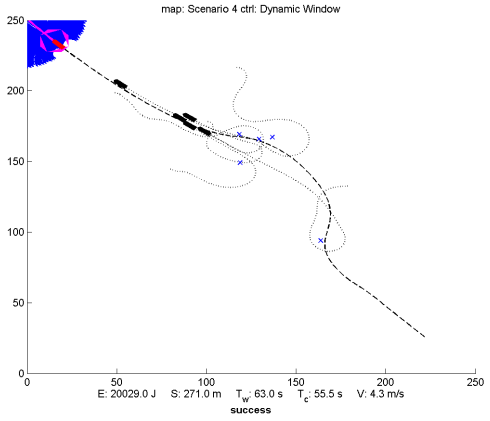
Figure 4.7: In the fourth scenario, our vessel starts out at the bottom-right of the map. It has to make it all the way to the upper-left corner, avoiding the hostile black vessels.



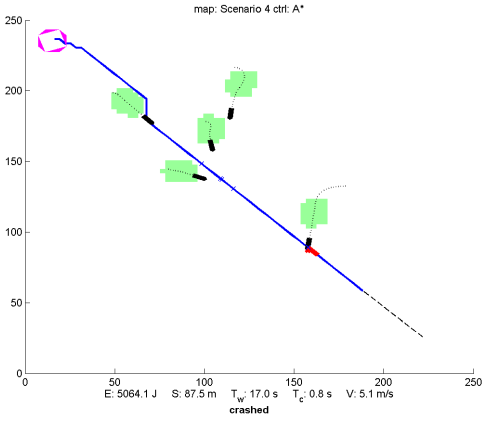
(a) Scenario 4 - VFF (failure)



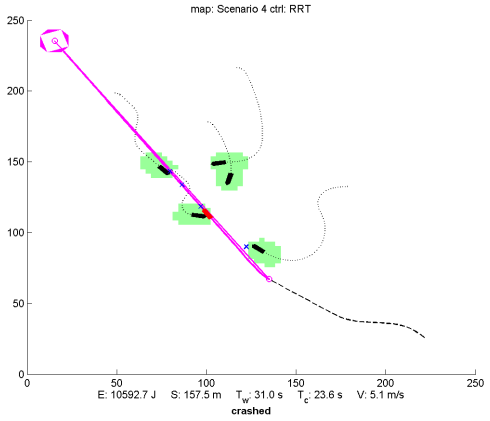
(b) Scenario 4 - VFH (success)



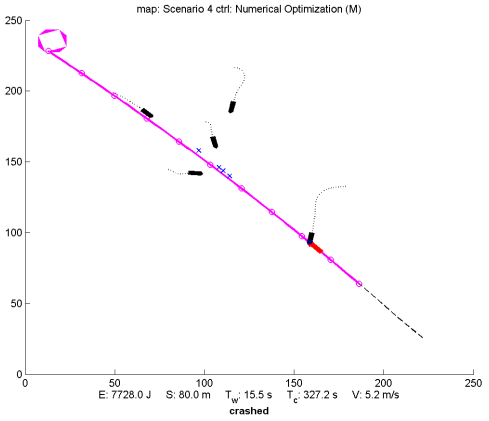
(c) Scenario 4 - DW (success)



(d) Scenario 4 - A* (failure)

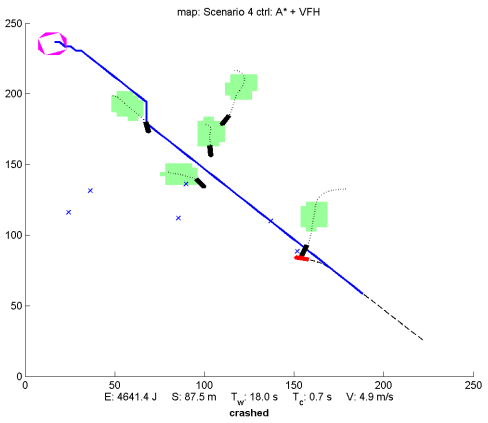


(e) Scenario 4 - RRT (6%))

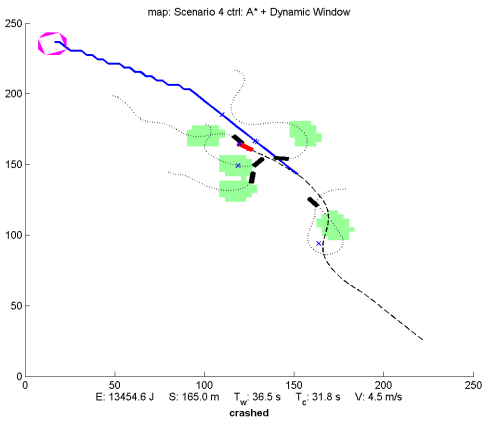


(f) Scenario 4 - MPC (failure)

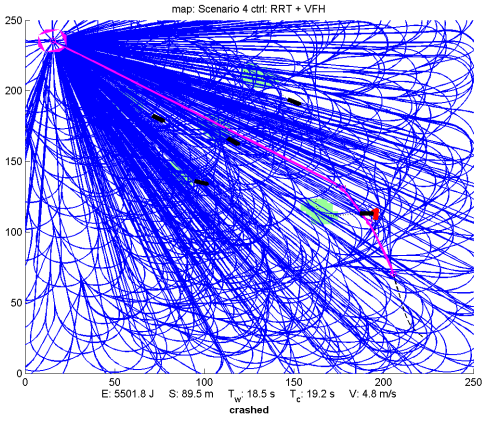
Figure 4.8: Local and global controllers separately navigating through Scenario 4.



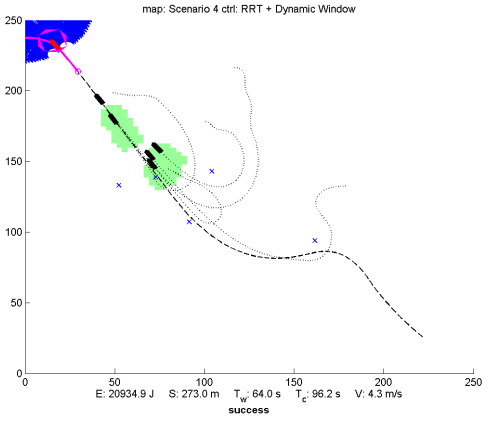
(g) Scenario 4 - A* + VFH (failure)



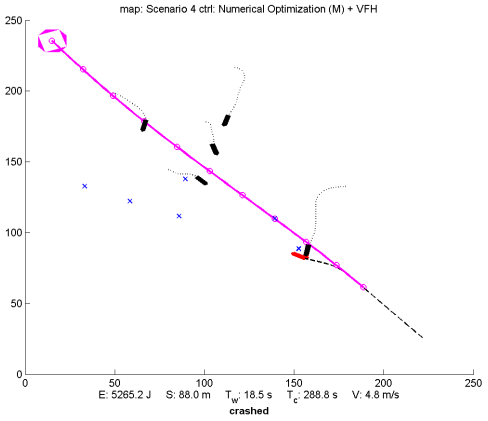
(h) Scenario 4 - A* + DW (failure)



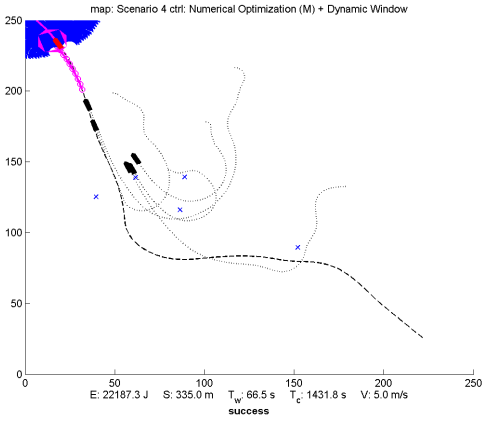
(i) Scenario 4 - RRT + VFH (21%)



(j) Scenario 4 - RRT + DW (49%)



(k) Scenario 4 - MPC + VFH (fail.)



(l) Scenario 4 - MPC + DW (suc.)

Figure 4.8: Hybrid controllers navigating through Scenario 4.

Results from Scenario 4

This scenario shows the value of having a control strategy able to react to fast changes in the environment. The low update frequency of the global methods makes them ill suited to avoid the hostile vessels, and as a result none of the global methods succeed in this scenario.

The local controllers fare much better. As seen in Figure 4.8(b) and Figure 4.8(c), both the VFH and DW controllers manage to escape the attacking vessels and make it to the goal.

Even though the good results shown in figures 4.8(b) and 4.8(c) look promising, too much weight should not be laid on them. It is easy to make small changes to the scenario, resulting in either of the controllers failing.

A better measure of performance for the local controllers can be found in the results from their hybrids with the RRT controller. Being randomized, the RRT always generates a different path for the local controllers to follow. This perturbs the scenario in different ways, and allows us to get a more accurate picture of the performance of the local controllers.

By itself, the RRT controller manages a low 6% success rate. Teaming it up with the VFH controller gives a success rate of 21%. Using Dynamic Window as the local controller in the hybrid increases the success rate even further, to 49%. The Dynamic Window controller is thus twice as likely to succeed the mission as the VFH controller.

Many of the cases where the Dynamic Window controller fails in this scenario is because of its notion of *admissible velocities*. When our vessel gets too close to the other vessels, and has no admissible velocities to choose from, it applies its maximum deceleration in an attempt to come to a halt. This causes the other vehicles to hit it. Removing or modifying this *braking* feature would quite possibly have increased the rate of success beyond 49%.

The RRT controller attempts to predict the future movement of the other vessels in the environment (see Section 3.2.3), but as it has no information about the intentions of the other vessels, the prediction is rather poor, and the only way of avoiding the oncoming vessels is by effective feedback. When the RRT is only allowed to run once every 10 seconds, the feedback is not responsive enough, which explains the low 6% success rate.

An interesting thought is giving the RRT an exact model of the behavior of the attacking vessels. Given that model, the predictions of the movements of the other vessels would have been perfect, and the RRT algorithm would only accept paths not ending in a collision. This would have resulted in a 100% success rate. The Dynamic Window and MPC controllers would also have benefited greatly from such a prediction (although it would have made the constrained nonlinear problem even more nonlinear and difficult to solve). This type of prediction is can be done in a simulated environment, but in a real-life situation, the limited information available makes this difficult. Enemy vehicles will not collaborate and share useful data.

	Success	Quality	Distance (S)	Avg. Vel. (V)	Energy (E)	Time (T_w)	CPU Time (T_c)
VFF	failed						
VFH	yes	good	**267.7**	4.1	23.6k	65.5	**0.1**
DW	yes	good	271.0	4.3	**20.0k**	63.0	55.5
A*	failed						
RRT	6%		316.7 (20.0)	5.0 (0.0)	21.2k (2.3k)	**62.8** (4.0)	5.0 (0.0)
MPC	failed						
A*+VFF	failed						
A*+VFH	failed						
A*+DW	failed						
RRT+VFF	0%						
RRT+VFH	21%		308.5 (35.1)	4.3 (0.3)	**27.4k** (7.0k)	**71.6** (11.5)	43.1 (9.5)
RRT+DW	49%		316 (49.7)	4.4 (0.3)	25.1k (7.0k)	71.3 (11.8)	104.6 (14.3)
MPC+VFF	failed						
MPC+VFH	failed						
MPC+DW	yes	good	**335.0**	5.0	22.2k	66.5	**1431.8**

Table 4.5: A summary of the results from Scenario 4.

4.2.5 Scenario 5: Surprise

The fifth and last scenario is special in that is a challenge needing the skills of both a global and a local method. The global method is only allowed to calculate a path once, and it has to stick to this path for the rest of the scenario. This is to simulate the time-delay in global methods, which may occur at an unfortunate time, limiting the vessels ability to respond to an event. This unforeseen event is represented by a large vessel changing course, and probably blocking the precalculated path.

To get around this problem, a local method is needed. With its higher bandwidth, it is able to respond to the change, and guide our vessel safely around the other vessel. The reason we need the global method in this scenario, is the presence of a local minimum.

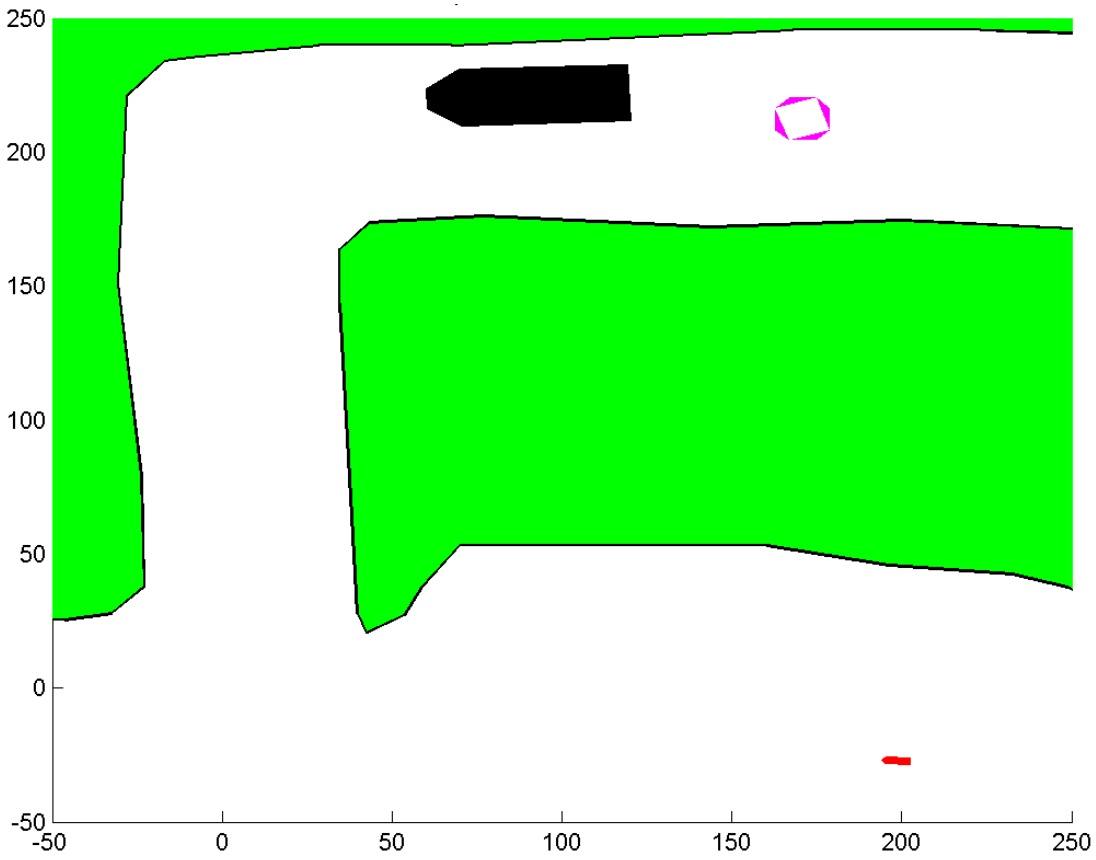
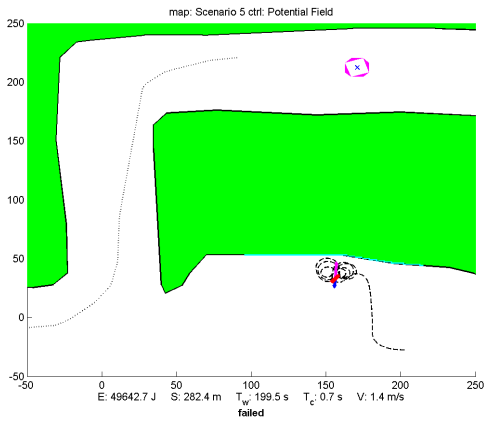
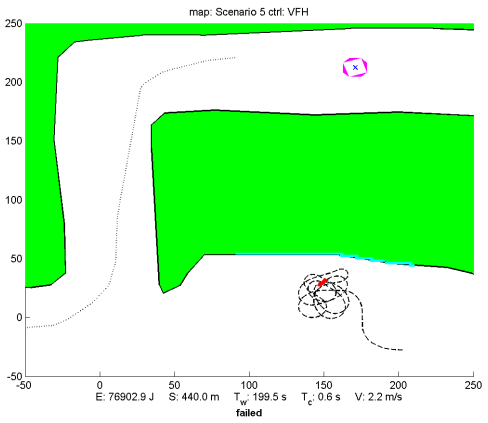


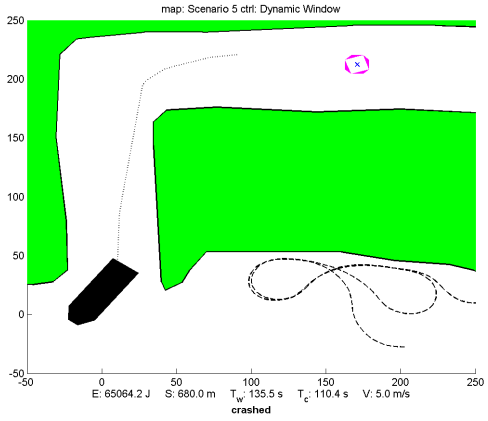
Figure 4.9: The last environment has our vessel traveling through a narrow channel. The large black vessel travels in the opposite direction, and has to be avoided for success.



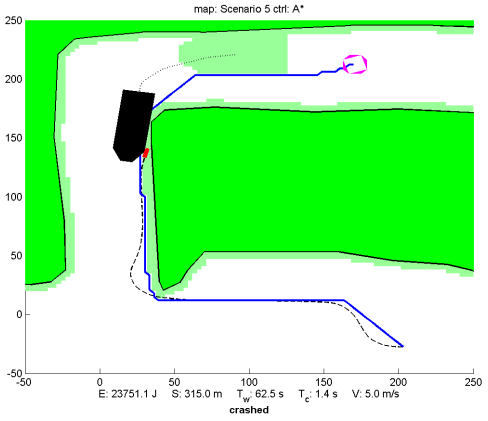
(a) Scenario 5 - VFF (failure)



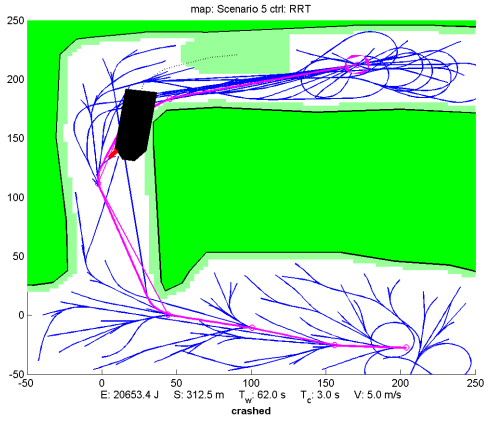
(b) Scenario 5 - VFH (failure)



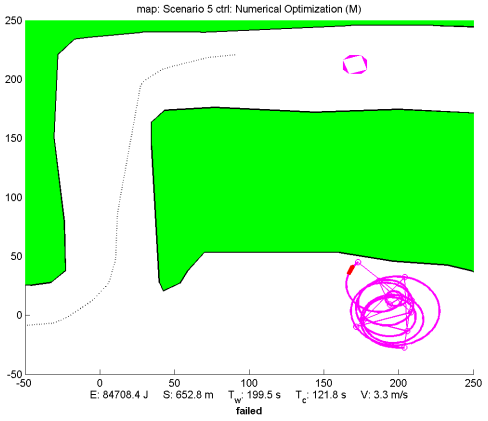
(c) Scenario 5 - DW (failure)



(d) Scenario 5 - A* (failure)

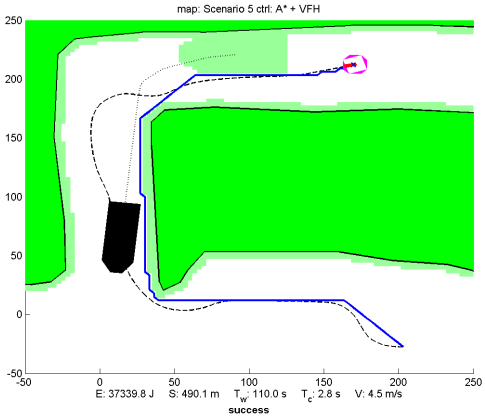


(e) Scenario 5 - RRT (22%)

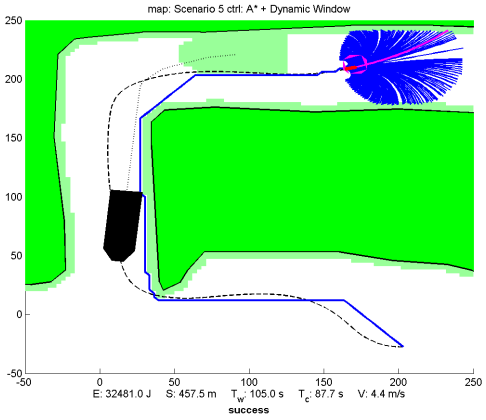


(f) Scenario 5 - MPC (failure)

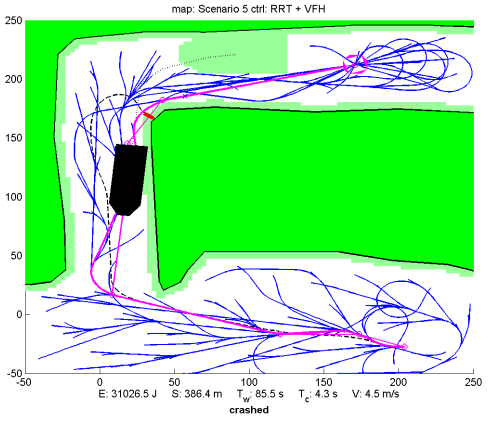
Figure 4.10: Local and global controllers separately navigating through Scenario 5.



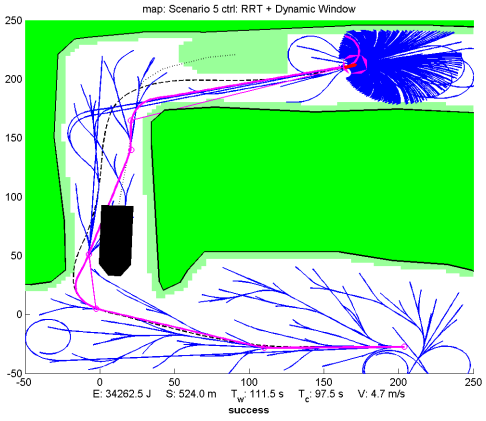
(g) Scenario 5 - A* + VFH (success)



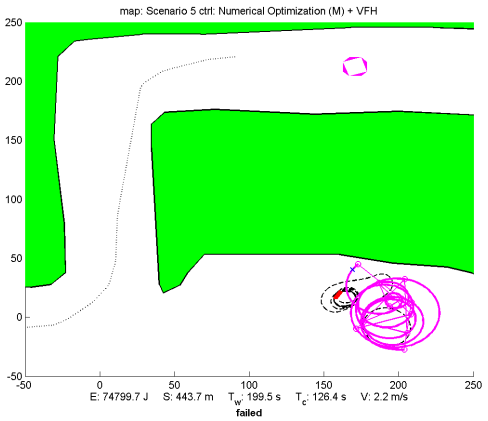
(h) Scenario 5 - A* + DW (success)



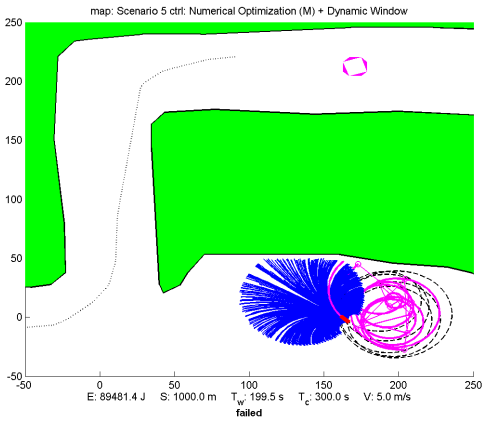
(i) Scenario 5 - RRT + VFH (89%)



(j) Scenario 5 - RRT + DW (100%)



(k) Scenario 5 - MPC + VFH (fail.)



(l) Scenario 5 - MPC + DW (failure)

Figure 4.10: Hybrid controllers navigating through Scenario 5.

Results from Scenario 5

As predicted, the hybrid methods fare best in this scenario. All the local methods fall prey to the local minimum in the environment. The MPC controller also fails to get out of this minimum. The A* and RRT controllers are able to find a path through the environment, but as they are unable to adapt the path they are mostly unsuccessful.

The hybrid methods are more successful in navigating the environment. The global algorithm generates a path, leading our vessel out of the local minima, while the local algorithm provides the responsiveness to avoid the oncoming vessel.

As in Scenario 4, there is a difference in the success rate of RRT+VFH and RRT+DW, pointing out the Dynamic Window algorithm as the better choice.

	Success	Quality	Distance (S)	Avg. Vel. (V)	Energy (E)	Time (T_w)	CPU Time (T_c)
VFF	failed						
VFH	failed						
DW	failed						
A*	failed						
RRT	22%		**557.4** (34.8)	5.0 (0.0)	**37.5k** (3.2k)	**111.0** (7.0)	3.6 (0.4)
MPC	failed						
A*+VFF	failed						
A*+VFH	yes	average	490.1	4.5	37.3k	110.0	**2.8**
A*+DW	yes	good	**457.5**	4.4	**32.5k**	**105.0**	87.7
RRT+VFF	0%						
RRT+VFH	89%		512.4 (21.3)	4.7 (0.2)	36.8k (2.9k)	108.8 (4.2)	5.1 (0.4)
RRT+DW	100%		526.3 (28.3)	4.8 (0.2)	33.1k (1.7k)	108.7 (4.0)	**96.3** (3.7)
MPC+VFF	failed						
MPC+VFH	failed						
MPC+DW	failed						

Table 4.6: A summary of the results from Scenario 5.

4.3 Discussion

After the results of these simulations, one thing should be clear: A complete collision avoidance system cannot consist of a local or global method by itself. Together however, they are gold.

The question of which combination is better remains. The VFF controller based on the Potential Field method was a constant underperformer. The trajectories it generated were excessively wobbly and had a tendency to end in collision. Because of this, VFF is not considered any further.

Based on the results presented in this report, the MPC is also disregarded. Controlling vehicles using constrained nonlinear optimization holds great promise, and will probably be the control approach of the future, but right now the method it is not very accessible. The rest of the discussion will be based on a comparison of the remaining controllers.

4.3.1 Vector Field Histogram vs. Dynamic Window

The Dynamic Window and Vector Field Histogram controllers are based on two rather different approaches. Where DW takes the vehicle dynamics into account, VFH disregards them completely. One of the results of this is that DW consistently uses less energy controlling the vehicle than VFH, even though DW holds a higher average velocity.

This, teamed up with prediction, also results in the DW being less likely to crash than VFH. Of the 9 failures when using RRT+DW in scenarios (1,2,3,5), only one of them was the result of a collision. The others were due to the vessel entering a limit cycle around the goal. On the other hand, the RRT+VFH method had 13 failures in these scenarios, and 12 of them were because of a collision. The RRT+DW was also more than twice as successful in Scenario 4 as RRT+VFH.

Looking at other aspects of performance, the DW consistently produces shorter paths than VFH when teamed up with A*. When using RRT, VFH takes the crown, but because DW is biased towards high velocities, and manages to keep its average velocity up, it still finishes in less time than VFH in most of the cases.

Of the three local methods reviewed, the Dynamic Window method appears to be superior. A problem that was noticed was its inability to select very hard turns unless it was forced to do so. This resulted in some failures where the vessel circled the goal and never managed to take the hard turn needed to actually get there. In an eventual live implementation of this controller, effort should be made to remedy this.

4.3.2 A* vs. RRT

As the DW and VFH controllers, the RRT and A* controllers are based on different approaches. RRT takes the full vehicle dynamics into account, generating a path the vehicle is guaranteed to be able to follow (given an accurate model). On the other hand,

the A* algorithm generates a shortest possible path through the environment, neglecting the dynamics of the vehicle.

The way RRT works, it will always create sub-optimal paths. On average, the controllers based on A* have a lower distance traveled, lower time to complete, and lower energy consumption than the RRT (VFH is a really bad match for A*, and pulls the other way). Looking at these measures of performance only would make A* the superior global method.

The picture is however a bit more complex, and Scenario 3 is a good example of this. As the A* controller performs no prediction, it may lead our vehicle along paths eventually leading to collision. In Scenario 3, the local controllers managed to correct the situation, but this may not always be possible.

The A* may also choose paths that are a bit risky. Implementing a large safety-zone around obstructions mends this, but will have an impact on the performance of the A* algorithm.

Looking at these results, the better method of A* and RRT cannot be chosen. The RRT method has its strengths in safety, while A* has better performance. A* is also more likely to find a path through a complex environment than RRT.

These simulations mark A*+DW and RRT+DW as the superior controllers, where A*+DW has the best performance of the two.

Chapter 5

Practical considerations

Several collision avoidance methods have now been evaluated, some more sophisticated than others. Some of the methods have been pretty successful, but there are still several considerations that must be made before any of them can be put into practical use. Sensor systems and the marine rules of the road are just some of the important issues. Several of these are discussed in this section.

5.1 Sensors

All of the collision avoidance methods depend on knowledge about the situation of the controlled vehicle to do their job. Information is needed both about the state of the vehicle and the surrounding environment. The amount and type of information required varies from method to method, but accuracy has an effect on all of the methods, as faulty information can actually be worse than no information at all.

5.1.1 Internal state sensing

The first sensors focus on getting an image of the state of the controlled vehicle. Using a 3 DOF model, the state of our vehicle can be represented by $\mathbf{x} = [x\ y\ \psi\ u\ v\ r]^T$. The sensors in this section will have the main responsibility of gathering enough data to get a good estimate of $\mathbf{x}$. The gathered data can then be fused using an *Extended Kalman Filter*, a *nonlinear observer*, or some other method. Variations of the Kalman Filter more suited for nonlinear systems, such as our vessel model in Section 3.1, may be the *Sigma-Point Kalman Filters* [45].

Compass

The compass [56] has been in use for several hundred years. It provides the user with the direction of *north*, and can thus be used to calculate the heading ψ of the vessel. The original magnetic compass relies on the earth's magnetic field for direction, and therefore actually indicates the direction of the *magnetic north*. The location of the

magnetic north varies, and generally deviates a great deal from the *true north*. When using a traditional compass, this must be compensated for.

The *gyrocompass* uses a different approach to finding north. As the name indicates, the gyrocompass is based on a gyroscope. It uses this gyroscope to keep aligned with north as the vehicle turns. It also has a mechanism for making sure that the compass is realigned with north if it falls out of alignment, e.g. due to the rotation of the earth.

The main advantages of the gyrocompass over the magnetic compass is that it points to the true north, and that it is unaffected by magnetic phenomena, for instance magnetic distortion caused by ferrous metal in a ship's hull [13].

GPS

The *Global Positioning System* [11, 13, 58], named *NAVSTAR GPS* was developed by the United States Department of Defense. It is based on a set of *at least* 24 GPS satellites, which periodically transmit their internal time, as well as data needed to calculate their coordinates. If a GPS receiver gets signals from at least 4 satellites, it can calculate an estimate of its own coordinates (x, y) from the received data. The z coordinate, or *heave*, of the vehicle is also calculated. For our 3 DOF model, the z value is not needed, but in other applications it may be.

Using a series of calculated coordinates, the velocities of the vehicle (u, v, w) can be calculated. The third value w denotes the velocity in the z direction, and is not included in our 3 DOF model. If the vehicle is in motion, the direction of travel can also be calculated. Depending on the vehicle model, the direction of travel can be a good estimate of the heading of the vehicle.

By having two GPS receivers at different locations on the vehicle, its true heading ψ can be calculated. This should not be seen as a system replacing the compass, but it can serve as a valuable addition to the sensor system of our vehicle.

Currently, NAVSTAR GPS is the only fully functional *Global Navigation Satellite System*. In an effort to reduce dependency on the US-controlled GPS system, two alternatives are under construction. The Russians have been working on their *GLONASS* system since the days of the Soviet Union, and are currently committed to finishing the system. In Europe, The European Union (EU) and the European Space Agency (ESA) are working on *Galileo*. Galileo aims at achieving a higher accuracy of measurements than both the GPS and GLONASS systems.

IMU

An *Inertial Measurement Unit* (IMU) [13, 60] is a component able to measure linear accelerations and angular velocities. To accomplish this, it relies on a three-axis accelerometer and a three-axis gyroscope. The accelerometer senses acceleration along all the three body axes ($\dot{u}$, $\dot{v}$, $\dot{w}$), and the gyroscope senses the angular velocities about the same axes: (p, q, r).

The velocities of the vehicle relative to the body axes can be acquired by integrating the linear accelerations once. The coordinates of the vehicle can also be calculated, by

integrating the velocities after applying a transformation to a suitable reference frame. Vehicle heading can be found by integrating $\dot{\psi}$.

The problem with these calculated velocities and coordinates is that they tend to drift. This is due factors such as noise, quantization errors and rounding errors. Because of this, an IMU is best used in combination with other sensors. The strength of the IMU is its bandwidth. When using an IMU in combination with a GPS and a compass, the IMU will improve the bandwidth of the total system. It will also help improve the accuracy of the estimates because the measurements now come from multiple sources. The GPS and compass are used to realign the coordinate and velocity estimates of the IMU to prevent drifting.

5.1.2 External environment sensing

As well as knowing the state of the controlled vehicle, the collision avoidance system needs information about the external environment. Especially, it needs to know about all the obstructions it has to avoid.

Radar

Radar is an acronym for *Radio Detection and Ranging* [61]. As its name implies, the radar uses radio waves for detection of objects. It can do this over a wide range of distances, and has since it entered use during the Second World War, provided a valuable tool for navigation and collision avoidance. For a brief history lesson, see [12].

In raw mode, the radar provides the distance to obstacles and their bearings. This raw data can be used directly in methods such as VFF (page 9) and VFH (Section 2.2.2), as this is the only information these methods need. The histogram grid employed by these methods also makes them resilient to sensor noise.

Modern radars have systems built upon them, as *automatic radar plotting aids* (ARPA) to identify and track other dynamic objects, e.g. vessels. In addition to distance and bearing, the course and speed of these objects are calculated. For the convenience of the radar operators, the *closest point of approach* (CPA) and the *time of closest point of approach* (TCPA) are also calculated. These represent where and when an eventual collision with an object on a collision course with our vehicle will occur.

The radar is an important provider of information, but it has its flaws. As information about the velocity and heading of other objects are based on relative data, it is only accurate when the other objects are moving at constant velocity and heading. When both our own vehicle and the other vehicle turns at the same time, which is often the case during an collision avoidance maneuver, this data is especially unreliable [3, 42]. The radar may also be relatively inefficient at very close ranges.

Digital nautical charts

Charts have always been an important tool for navigators, and even today they are essential for safe marine navigation. In digital form, they also become available for

Figure 5.1: A typical radar display unit. (Image courtesy of Selex)

our collision avoidance system. The nautical charts provide information on the static features of the environment, such as coastlines, piers, buoys, shallow water, etc.

To be able to determine the environment from a chart, the coordinates of our vehicle has to be known. One way of acquiring these is the use of a GPS system, see Section 5.1.1.

It is important to note that even a good chart may have flaws. This, in addition to the fact that charts cannot take dynamic obstructions into account, makes navigation exclusivly by charts potentially disastrous. Digital charts however provide a great value when combined with other sensors.

AIS

AIS is short for *Automatic Identification System*, and it allows vessels to exchange information required to perform effective collision avoidance [3, 43, 55].

Every vessel with an AIS transponder regularly transmits data about its own state, identity and basic handling characteristics. The information is transmitted over VHF, and all other AIS-enabled vessels in range receives it. This gives all AIS-enabled vessels an accurate view of the situation; more accurate than that acquired using radar only. The transmission of identification information and handling characteristics makes AIS especially powerful, as it gives us the ability to better predict the future behaviour of the other vessels. The transmitted ship draught will for example allows us to exclude trajectories through waters that are too shallow.

The downside of using the AIS system is that only vessels using AIS are visible. With some exceptions, only vessels exceeding 300 gross tonnes are required to be fitted with an AIS system [29]. Smaller vessels, hostile vessels, and other dynamic obstructions such as icebergs and kayaks, will not be detected.

Computer vision

Computer vision is an exciting technology. It has the potential of providing information on the three-dimensional structure of the environment and obstructions, as well as identification of the different objects in the environment. One example of computer vision

used for an autonomous system is the Mars Rover [18].

There are at least two different approaches to computer vision; monocular and stereo. Stereo vision systems mimic the way the eyes of humans work. By watching a scene from at least two slightly different angles at the same time, depth information can be registered. This is also known as *passive stereo* [21]. Using this approach, a three-dimensional model can be constructed of the environment. As the distance from the cameras increases, the accuracy of the depth information decreases. This is among other factors due to quantization errors in the acquired images and noise. An evaluation of stereo vision using CCD cameras and *forward looking infrared* (FLIR) can be found in [37].

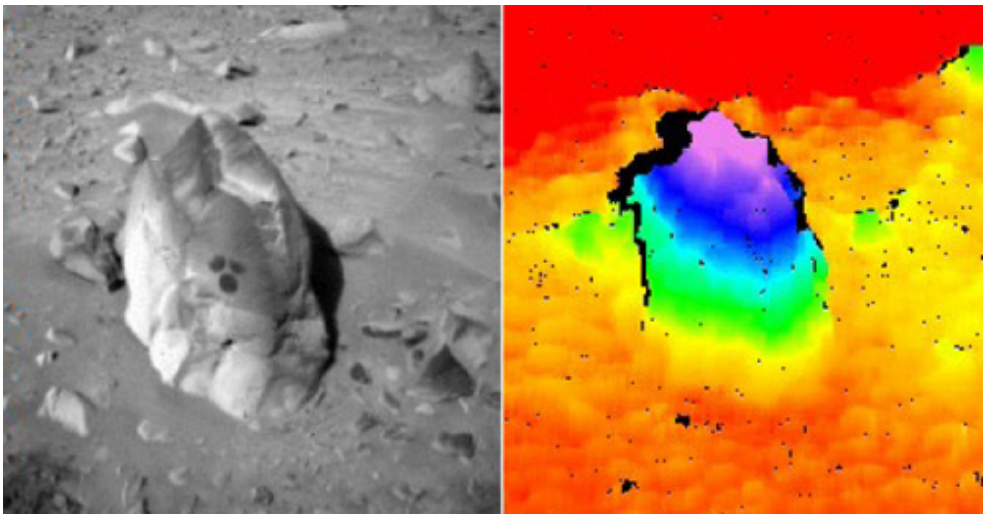


Figure 5.2: This is an example of stereo vision image processing as used on Mars Rovers. The left image shows the original image from one of the cameras, and the right image shows the computed elevation map. (Image courtesy of NASA)

A downside to using stereo vision is that it is very dependent the configuration of the two cameras, and thus requires calibration when the cameras are mounted. Camera movements can affect the calibration and make the gathered depth-information erroneous [3]. Range resolution and accuracy is also far below what can be achieved with active techniques such as *laser range finders*.

Monocular vision systems do not require the precise calibration of the stereo systems, but extracting environment information from the single images provided by monocular vision provides its own challenges. For USVs, detection of objects can be done as follows: For each frame received, a search is done to find the horizon. A search is also done to identify possible obstacles in the image. Given the height of the camera, the distance from the camera to the horizon, and the position of the obstacles in the image, a rough estimate of the distance to the obstacles can be calculated.

A strength of vision-based sensor systems is that they make automatic identification of objects possible. Since they are based on conventional cameras, they can also be fitted in fairly small packagings. A downside is that they depend on the light conditions of the environment. Vision systems based on visible light will for example be less effective at night and in dense fog. The vision systems based on infrared light (FLIR) are on the other hand effective both at day and nigth, and even in fog. Identification is more difficult with infrared systems however.

Laser range finder

As the name implies, a laser range finder uses one or more lasers to measure distance. According to [21], there are three basic technologies in active use: AM-CW systems measure the difference in phase between emitted and received beam, TOF (time of flight) systems measure the travel time of a pulse, and FM-CW systems use frequency shift to determine distance.

The TOF systems have the greater range, and ranges of several hundred meters should be possible with accuracies down to $\pm$ 5 cm. A downside to the TOF systems is a rather limited acquisition frequency compared to the other systems. A limit of a few thousand samples per second seems common.

5.2 The rules of the road

Collisions at sea have always been a great problem for the maritime industry. From the very start, efforts have been made to reduce this risk, including the use of lights and signals, and establishing working rules for mariners to follow when encountering other ships. As time went by, the need for a common set of rules was seen, and the result of this eventually became the *International Regulations for Avoiding Collisions at Sea*, or simply COLREGS [2, 12].

COLREGS is a set of international rules defining measures to be taken to avoid collisions. It defines everything from the lights a vessel is required to carry, to which course of action a vessel should follow when encountering other vessels at sea. Where the rules come short, the individual governments can make supplements. The COLREGs have been steadily evolving, e.g. by the recognition of the RADAR.

There are several reasons for why CA systems for vessels should adhere to the COLREGS, but mainly it is to produce predictable maneuvers compatible with the maneuvers of other vessels. Following is a review of the rules that affect our collision avoidance system in a significant way.

5.2.1 Head-on situation

Rule 14 defines the *head-on situation*. When in a head-on situation, a vessel should pass the other vessel on the port side. This situation is depicted in Figure 5.3.

Rule 14a:
When two power driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision each shall alter her course to starboard so that each shall pass on the port side of the other.

Rule 14b:
Such a situation shall be deemed to exist when a vessel sees the other ahead or nearly ahead and by night she could see the masthead lights in line or nearly in line and/or both sidelights and by day she observes the corresponding aspect of the vessel.

(a) Correct pass **(b)** Incorrect pass

Figure 5.3: Our vessel can never rely on the other vessel taking preventive action to avoid a possible collision. In (a), our vessel coming from the left makes a correct pass of the other vessel.

5.2.2 Crossing situation

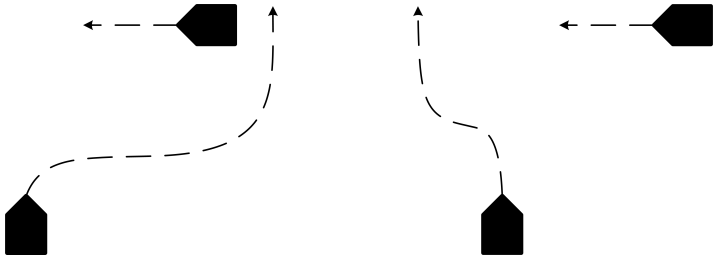
The *crossing situation* is defined in rule 15. When two vessels are about to cross paths in a way that risks a collision, the give-way vessel should pass behind the other vessel. This can be related to driving, where the right of way must be yielded to the driver coming from the right in an intersection

Rule 15:
When two power driven vessels are crossing so as to involve risk of collision, the vessel which has the other on her own starboard side shall keep out of the way and shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.

5.2.3 Overtaking situation

When overtaking another vessel, the overtaking vessel should keep well clear of the other vessel. Special rules apply when in a narrow channel or fairway, but these mainly consider signaling.

Rule 13a:
Notwithstanding anything contained in the Rules of Part B, Sections I and



(a) Correct crossing

(b) Incorrect crossing

Figure 5.4: Our vessel approaches the other vessel from the bottom of the figure. (a) shows the correct course of action.

> *II, any vessel overtaking any other shall keep out of the way of the vessel being overtaken.*
>
> Rule 13b:
> *A vessel shall be deemed to be overtaking when coming up with a another vessel from a direction more than 22.5 degrees abaft her beam, that is, in such a position with reference to the vessel she is overtaking, that at night she would be able to see only the sternlight of that vessel but neither of her sidelights.*

5.2.4 The give-way vessel

In many situations, including the *crossing situation*, one of the vessels has the right-of-way. This does not take away the vessels responsibility of avoiding a collision, but as long as a collision can be safely avoided in good time by actions of the other vessel, the vessel that has the right-of-way can continue on its path.

> Rule 16:
> *Every vessel which is directed to keep out of the way of another vessel shall, so far as possible, take early and substantial action to keep well clear.*
>
> Rule 17a:
> *(i) Where one of two vessels is to keep out of the way of the other shall keep her course and speed.*
> *(ii) The latter vessel may however take action to avoid collision by her maneuver alone, as soon as it becomes apparent to her that the vessel required to keep out of the way is not taking appropriate action in accordance with these Rules.*

In most cases, our vessel will not have the right of way. Rule 18a contributes to this. In addition, our vessel should keep clear of larger vessels, as forcing a larger vessel

to take evasive maneuvers is inefficient, and can be dangerous. A distinction should be made for smaller or equal-size vessels.

> Rule 18a:
> *A power driven vessel underway shall keep out of the way of: (i) a vessel not under command; (ii) a vessel restricted in her ability to maneuver; (iii) a vessel engaged in fishing; (iv) a sailing vessel;*

5.3 Time delays

A consideration that has not yet been made is the impact time delays will have on the final implemented collision avoidance system. In the simulator, world time is *frozen* while calculations are done, and the computation time of the implemented algorithms will therefore have no impact on the simulation. In the real world, the case is very different.

Say our vessel is moving along at some velocity. At time t_0, our Nonlinear Optimization controller starts calculating an optimal input sequence for the vessel, starting at state $\mathbf{x}(t_0)$. The input sequence is finished and ready to be applied to our vessel at time t_1. The only problem is that the input sequence was calculated to be optimal given an initial state $\mathbf{x}(t_0)$. Now the state of the vessel is $\mathbf{x}(t_1)$, and the result of applying the input sequence to our vessel will not be as expected.

If the control and dynamics of the vessel are predictable between times t_0 and t_1, e.g. because the vessel is following a previously calculated input sequence, $\mathbf{x}(t_1)$ can be estimated and the input sequence can be calculated from this state instead of from $\mathbf{x}(t_0)$. This would appear to be a solution to the problem, but in fact it is not entirely so. The first reason for this is that t_1 in itself may be very difficult to determine exactly. The other reason is that even given t_1, $\mathbf{x}(t_1)$ will prove very hard to determine exactly. This is due to waves and unmodelled dynamics, as well as possible intermediate control action performed by a lower level controller (such as a local controller in a hybrid setup).

The point is that using the input sequences directly (which the RRT also creates), is a bad solution. When these controllers are used by themselves, they should calculate a state trajectory using their input sequences, and then let the vessel follow this using an asymptotically stable controller. When used in hybrids, the local controllers should use this trajectory for guidance.

Tracking of the generated trajectory should not be time dependent, as the controller has no way of knowing t_1 exactly, and thus not the exact time of the vessel along the trajectory either, forcing the vessel to play catch-up or slow down to re-synchronize with the time along the trajectory.

There is always a chance that the situation of the vessel changes so drastically during path generation that the new path is infeasible or highly unwanted when it is to be put to use. Logic must be present to detect and resolve such a situation.

5.4 Failsafe system

As we have seen in many of the simulations, the collision avoidance algorithms can fail. This may for instance be the result of faulty assumptions, as in the A* algorithm, where vehicle dynamics is not considered. It may also be that the problem is too complex, making the algorithms give up (RRT) or even produce an infeasible path (MPC).

Many of these cases can be detected and resolved by adding logic to the CA system. Infeasible paths can for instance be detected by simulating the vehicle model forward along the path in an attempt to verify them.

The CA system will nevertheless be quite complex. Problems with the algorithms, such as those described above, are only one of the possible pitfalls. Significant environmental disturbances, unconsidered aspects, and even coding errors all represent threats. In a live implementation of such a system, protection against such threats is essential.

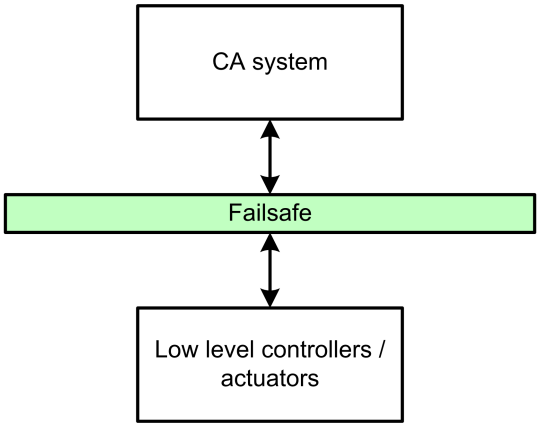


Figure 5.5: Possible implementation of a failsafe system. The failsafe component works as an interface between the CA system and the vessel.

A failsafe system can be implemented as shown in Figure 5.5. A separate component operates as an interface between the CA system and the low-level controllers of the vehicle. All commands from the CA system pass through this component on their way to the vehicle controllers. If the failsafe system notices a problem, it can override the signals from the CA system, and for example take the vehicle to a complete halt.

The failsafe should be implemented in another process than the CA system, preferably on another computer, to reduce the probability of a fault in the CA system taking down the failsafe as well. It should be as simple and robust as possible.

5.5 Cooperative CA

Cooperation presents a great opportunity for improving the efficiency of our collision avoidance system. Cooperative collision avoidance is a general term for multiple vehicles cooperating to avoid collisions between each other and the environment. Discussions of

cooperative collision avoidance can be found in [16, 17, 52]. The ability to accurately predict the future movements of other vehicles in an environment can drastically increase the efficiency of a collision avoidance system.

Cooperative collision avoidance can be implemented by making the cooperative vehicles share the predictions of their own trajectories. A vehicle can then use the received information as the prediction on the future movements of the other vehicles instead of relying on the simple prediction scheme presented in Section 3.2.3. Other methods include having a centralized motion planning system, generating feasible paths for all cooperating vehicles. The different vehicles can also be given priorities, so a vehicle in the planning phase only considers the trajectories of vehicles with higher or equal priorities. It is then assumed that the vehicles with lower priorities will plan their paths accordingly.

Any system based on information sharing relies on some form of trust between the different parties. This is especially true for cooperative CA, where the received information directly influences the decisions made by the CA system. Situations where an enemy poses as a friend and leads our vehicle astray must be prevented at all costs. Means of constructing a safe system include authentication of the included parties, and encryption of the transmitted data.

Encryption also helps prevent eavesdropping. It is not always desired to let everyone know the trajectory you are following. An example of this is when operating in enemy territory, where even giving away your coordinates can be fatal.

As mentioned in Scenario 4, better prediction can also be achieved by using knowledge about the behaviour of the other vessels in the environment. If our vessel for instance manages to identify an oncoming torpedo, it can be assumed that the torpedo will do its best to hit us. This information can be valuable for the CA system, which would normally expect the torpedo to continue straight ahead.

Chapter 6

Discussion

Six collision avoidance algorithms have been reviewed, and some practical considerations made. A question that can be asked is whether or not autonomous vehicles, and autonomous surface vehicles in particular, are possible in the near future. The answer is twofold, and contains both technological and social aspects.

6.1 Technological aspects

For a CA system to be effective, complete and accurate information on the state of the vehicle and the environment is required. The state of the vehicle can be estimated using off-the-shelf sensors (see Section 5.1.1) and software. In the *Dynamic Positioning* industry this information is very important, and much has been done to make it reliable.

Environment sensing has been a hot research area for many years, and some commercial systems exist. These systems are mainly for decision making support, and may not be accurate enough for our purposes. The sensor systems must be accurate enough to detect a lone kayaker, and even a swimmer, to be safe enough for autonomous travel. To achieve this, sensors more accurate than the radar, such as vision systems, are required.

There is a lot of research activity in the area of automatic vision systems, but currently there are no major commercial solutions available for marine environment sensing. The image recognition algorithms are not entirely mature either.

Looking at the computation time recorded from the simulations, most of the algorithms are able to run in real-time or near real-time. Large parts of the algorithms are written in the Matlab programming language, and will get a significant speed-up when ported to C/C++. The ability to take advantage of the new dual- or quad core systems for parallelization, versus Matlab which is mostly single-threaded, gives the potential for an additional speedup. All in all, making a collision avoidance system fast enough to run in real-time should not pose a problem.

6.2 Social aspects

When will people accept autonomous vehicles? Autonomy gives a vehicle or robot the ability to make its own decisions. The decision range of the robot can be constrained, either virtually through programming, or physically by limiting its degrees of freedom, but is this enough? Can we trust in the correctness of the robot program? Do we accept the robots gaining ever more control over our lifes?

In addition to acceptance, there are several legal issues concerning autonomous vehicles. *Who is responsible if an autonomous vehicle for instance runs over and kills a human? The owner, or the creators of the vehicle?*

The US military has been using *Unmanned Aerial Vehicles* (UAV) for reconnaissance and surveillance for several years. In contrast to today's UAVs, the next generation of UAVs, the *Uninhabited Combat Aerial Vehicle*, will be armed. This allows them to take on more active missions, both offensive and defensive, but also introduces many legal issues to be resolved [33]. Even though these vehicles are remotely controlled, there is always a chance of losing radio contact with them, effectively forcing them to operate autonomously. Protocols and restrictions must be defined for the safe operation of such vehicles.

The legal groundwork for UAVs has come much further than that for other unmanned vehicles because of their many military applications. Much work needs to be done before USVs can operate legally under autonomous control.

It may also be quite some time before fully autonomous robots are accepted by the common public. There are however several uses for collision avoidance systems that are not as controversial as full autonomy.

An obvious application is decision-making support for vessel operators. Automatic radar plotting aids already provide some form of support when other vessels are in the vicinity, including calculation of the CPA and TCPA. Collision avoidance algorithms would be able to provide the operator with trajectories to follow to get out of difficult situations, and maybe even the choice of automatically following a selected trajectory. This would ease the task of the operator, and provide a higher chance of avoiding a collision. Using cooperative techniques, the collision avoidance systems of the different vessels could also agree on trajectories to follow, reducing uncertainty and providing high quality service.

A related application is having a system that automatically kicks in and performs an evasive maneuver if necessary. For such a system to be safe, it must be positively certain about the necessity of the maneuver before application, otherwise the system would quickly loose trust. Volvo is currently working on such a system for cars [22].

The COLREGS have already been discussed in Section 5.2. No matter which application a collision avoidance system is used in, it is important that it does its best

to follow the rules humans have to follow. This ensures predictability, and makes the system act more *like a human*, which is important when it comes to acceptance.

Chapter 7

Conclusion

The results of this report clearly indicate the need for a hybrid controller. Individually, current local and global methods have weaknesses, but used together as a hybrid these weaknesses become efficiently neutralized.

Among the local methods, the Dynamic Window method proved superior, on average finishing the courses in less time than the other local methods. The global methods were more difficult to assess. The controller based on Constrained Numerical Optimization was the underachiever, as it had major problems finding optimal and feasible solutions. When in combination with a local controller, both the A* and RRT methods gave good results. A* gave the best performance, while RRT proved to be the more flexible and safe.

Of all the simulated controllers, the A*+DW and RRT+DW controllers were selected as the best. More work has to be done before these controllers can be used in a real-time application however. The objective function of the DW method has to be manipulated to allow it to slow down to take sharper turns, also in cases when it is not about to crash. The algorithms should also be implemented completely in C/C++ for efficiency.

The sensors needed for efficient collision avoidance are available today. Using a combination of RADAR, AIS and nautical charts for long-range environment sensing, and computer vision and laser range finders for short-range sensing, should give a good impression of the environment. Sensor fusion must be employed to create an environment model from the sensor readings.

In the coming years, collision avoidance systems will likely only be used in experimental autonomous systems, and in more advanced systems for decision support and operator assistance. A common denominator is that these vehicles will be supervised by humans. It will likely be some years before we give our car the shopping list and let it drive to the mall alone.

Autonomy is the way of the future and collision avoidance is a requirement for that to happen. Competitions like the Urban Grand Challenge [1] help bring focus to this area of research and possibly make people more susceptible to the involved technology.

Appendix A

DVD contents

The accompanying DVD contains the source code needed to run the simulations that have been presented in this report. It also contains images and movies from the simulations that the results of the report are based on. The source code and some of the results can also be found on the Internet at `http://www.divo.no/projects/colav07/`.

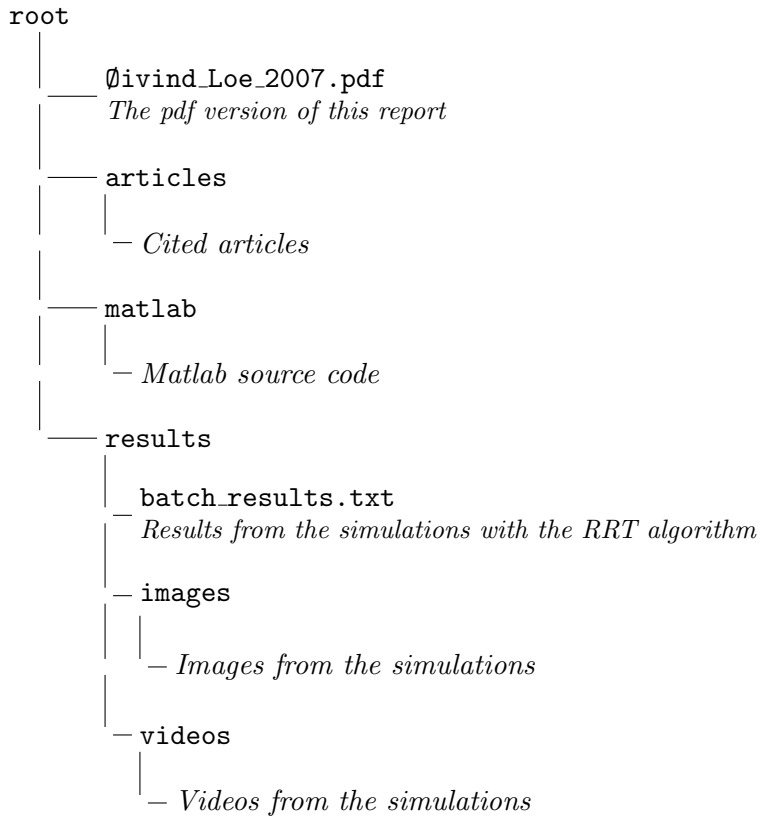


Figure A.1: The directory structure of the accompanying DVD.

Appendix B

Simulator user guide

This is a short introductory guide to the simulator that was developed during this project. It is by no means complete, but will provide enough information to get started with the simulator. If you have problems, send me an e-mail: `oivind.loe@gmail.com`.

B.1 Introduction

Before we can start a simulation, the environment must be set up. It is recommended that the code is run from a local hard-drive with both read and write access. The simulator needs this access to write images and movies, and to compile the mex code.

Set the current directory to the folder with the code in, and initialize the simulator by calling *init*. This will compile some files and initialize the TOMLAB Optimization Environment if it is installed.

```
init;
```

We are now ready to run a simulation. Use the *runScenario* command to start a new simulation. The *map* parameter gives the map name, and *controller* determines which controller should be used. The most important options for *runScenario* are provided in Table B.1. For all commands accepting options, the options can be omitted.

```
runScenario(map, controller, [options]);
```

The following command starts a simulation on the map named *s1* using the Dynamic Window controller. Table B.2 shows available maps, and Table B.3 shows available controllers. It also records a movie of the simulation, which is named *testmovie* and put in the *results* folder. Note that the filename is given without an extension.

```
runScenario('s1', 'dynwnd', ...
    'storeMovie', 'results/testmovie');
```

While a simulation is running, press *S* on your keyboard to pause it. Pressing the key again will resume the simulation. There are a lot of other things that can be done during a simulation, including taking screen shots, panning and zooming. For a complete

list of commands, press *H* during a simulation. The list will be printed to the Matlab command window.

Most of the results presented in this report can be recreated by running the *batch* command. It will run all the simulations in order and produce movies and images from them.

```
batch;
```

The simulations using the RRT controller are run a large number of times to calculate means and variances of the simulation results. Because of this, running the batch script may take a while. On the computer presented in Table 4.1, the entire batch used 61 hours and 47 minutes of CPU time to complete. Much of this time can be credited to visualization, which can be disabled by adjusting the *update* parameter of *runScenario*.

Option	**Description**
`storeImage`	Store an image of the result of the simulation. Value is filename without extension.
`storeMovie`	Record and store a movie of the simulation. Value is filename without extension.
`update`	Set the rate of which the visualizations should be updated. Value is integer between 1 and Inf. A value of 1 results in a continuous update, while Inf only updates the graphics at the beginning and the very end of the simulation.
`tFinal`	The simulation starts at time 0 and ends at this time. Default is 200.
`boatType`	Represents the type of boat to be guided through the map during the simulation. Default is `'Viknes'`.

Table B.1: Options for the *runScenario* command.

Map	**Description**
`s1`	Map used in *Scenario 1*.
`s2`	Map used in *Scenario 2*.
`s3`	Map used in *Scenario 3*.
`s4`	Map used in *Scenario 4*.
`s5`	Map used in *Scenario 5*
`Islands`	A map with lots of islands, no local minima.
`Rooms`	A map with local minima.
`Narrow`	The vessel starts out in a very narrow passage, posing a problem for some of the controllers.

Table B.2: Maps that can be used with the *runScenario* command.

Controller	Description
`keyboard`	Control the vessel with the arrow keys on your keyboard.
`const`	The vessel is given a constant set-point for velocity and heading.
`path`	The vessel tracks a predefined path.
`pot`	The Potential Field controller (VFF).
`vfh`	The VFH controller.
`dynwnd`	The Dynamic Window controller
`astar`	The A* controller.
`rrt`	The RRT controller.
`optm`	MPC. Uses a Matlab solver.
`optt`	MPC. Uses a TOMLAB solver.
`path+pot`	Hybrid with path tracker and Potential Field (VFF).
`path+vfh`	Hybrid with path tracker and VFH.
`path+dynwnd`	Hybrid with path tracker and Dynamic Window.
`astar+pot`	Hybrid with A* and Potential Field (VFF).
`astar+vfh`	Hybrid with A* and VFH.
`astar+dynwnd`	Hybrid with A* and Dynamic Window.
`rrt+pot`	Hybrid RRT and Potential Field (VFF).
`rrt+vfh`	Hybrid RRT and VFH.
`rrt+dynwnd`	Hybrid RRT and Dynamic Window.
`optm+pot`	MPC and Potential Field (VFF). [Matlab]
`optm+vfh`	MPC and VFH. [Matlab]
`optm+dynwnd`	MPC and Dynamic Window. [Matlab]
`optt+pot`	MPC and Potential Field (VFF). [TOMLAB]
`optt+vfh`	MPC and VFH. [TOMLAB]
`optt+dynwnd`	MPC and Dynamic Window. [TOMLAB]

Table B.3: Controllers that can be used with the *runScenario* command.

B.2 Creating a new map

Creating a new map is simple, but requires some steps. All maps are defined in *runScenario.m*. Navigate to the switch-case statement in the *Map* section of the file. Every case in this switch-case represents a map, so to create a new map, make a copy of one of the existing maps, and add it to the switch-case. `map0` is empty, and can be used as a template:

```
case 'map0'
    wmap.name = 'Map0';
    wmap.xLimits = [-45, 85];
    wmap.yLimits = [-45, 85];
    wmap.initState = [7.129, -20.0184, pi, 5, 0, 0];
    wmap.targetState = [36.1613, 46.9632];
```

```
wmap.objects = { ...
    objGoal( ...
        'target', 'mainBoat', ...
        'coords', wmap.targetState, ...
        'action', ['disp(''GOAL REACHED''); ' ...
            'gWorld.events.endSim = true; ' ...
            'gWorld.events.endReason = ''success'';']), ...
};
```

When running a map with the *runScenario* command, it is identified by its case label. This *must* be lower-case. The displayed name of the map is given by *wmap.name*.

The main boat, named *mainBoat* is always created. In the *wmap.objects* cell array, you can define additional objects to be added to the simulation. There has already been added an *objGoal* object. This will announce victory and end the simulation when it is reached by *mainBoat*. A number of other objects can be added, including obstructions, other vessels and controllers for the other vessels. A summary is given in Table B.4.

To add a single obstruction to the map, add the following object to the *wmap.objects* array:

```
objPath( ...
    'path',  [ -10 10 ; 20 10 ; 20 -10 ; -10 -10 ], ...
    'solid', true, ...
    'fillColor', 'green'), ...
```

This will add a green, rectangular obstruction to your map. The path making up the polygon can be automatically generated. During a simulation, press S to pause it. Then press P to enter *polygon-mode*. Click a couple of times in the simulation window to generate a new polygon. When you are satisfied, press *return*. The path will be printed to the command window.

B.3 Creating a new controller

Controllers are implemented in the files named *ctrl*.m*. A controller *m*-file has three main functions. The first is named the same as the file, and constructs the controller. It takes options from the user, creates the controller structure, and returns it to the caller.

The second function is called *Init*, and this is the place to create any additional objects your controller needs. This can be sub-controllers, visualization objects, plots, etc.

The third function is *Update*. This is the function where all the control action is performed. Use it to collect data from the environment, process the data, and optionally apply the resulting steering commands to the target vehicle. This function is called once every iteration.

To use a controller with a vehicle, place it above the object in the *wmap.objects* cell array. Adding the following two objects to the objects array associates a `ctrlAttacker` controller with a vessel as in Scenario 4:

Object	Description
`objArrow`	The arrow object visualizes an arrow. Used by the Potential Field controller to visualize virtual forces.
`objBoat`	This object implements the vessel model of Section 3.1, as well as visualizing a vessel. Different vessel types can be defined within *objBoat*
`objGoal`	The goal object allows a custom action to be run when a target object comes within a certain circle of acceptance of the goal coordinates. Also provides visualization of the goal.
`objMap`	This object provides a visualization of a map. Map data must be provided by other means.
`objPath`	The path object has multiple purposes. It can visualize a path, and it can visualize a polygon. When used as a polygon, the path object can be set to be *solid*, and be an obstruction for our vehicle.
`objQuiver`	Provides visualization of force fields. Used to generate the force arrows seen in Figure 2.3
`objXYTrace`	Can be used to plot a trace after moving objects. Objects based on objBoat automatically create a trace object if not told otherwise.
`ctrlAstar`	Implements the A* path planner.
`ctrlAttacker`	The control strategy used by the hostile vessels in *Scenario 4*.
`ctrlConst`	A controller which maintains constant values for the setpoints of a target object.
`ctrlDynWnd`	Implements the Dynamic Window controller.
`ctrlKeyboard`	Implements the Keyboard controller.
`ctrlOptim`	Implements the Nonlinear Optimization controller.
`ctrlPathTracker`	A path tracker which can be used by itself or to fuse global and local methods.
`ctrlPotentialField`	Implements the Potential Field controller.
`ctrlRRT`	Implements the RRT controller.
`ctrlVFH`	Implements the VFH controller.

Table B.4: Objects that can be used in a simulation. Objects with the *ctrl* prefix are controllers.

```
ctrlAttacker( ...
    'target', 'firstboat', ...
    'targetA', 'mainBoat', ...
    'u_max', 3), ...
objBoat( ...
```

```
    'initState', [178.7634 132.6687 pi 3 0 0], ...
    'boatType', 'viknes', ...
    'name', 'firstboat', ...
    'color', 'black'), ...
```

Controllers can also be chained, for instance letting a VFH controller be guided by an A* controller. In the following example, a `ctrlPathTracker` control used to choose a set of goal coordinates for the VFH controller from the path generated by the A* controller:

```
ctrlAStar('name', 'PATHGEN', ...
    'target', 'firstboat', ...
    'goal', wmap.targetState, ...
    'updateT', ctrlUpdateT), ...
ctrlPathTracker('name', 'GOALGEN', ...
    'target', 'firstboat', ...
    'pathSource', 'PATHGEN.path', ...
    'u_sp', 3, ...
    'skipToLastVisible', false, ...
    'radiusAccept', 30, ...
    'drawPath', false), ...
ctrlVfh('target', 'firstboat', ...
    'goalSource', 'GOALGEN.goal', ...
    'plotRawPolarHist', true, ...
    'drawMap', false), ...
objBoat( ...
    'initState', [178.7634 132.6687 pi 3 0 0], ...
    'boatType', 'viknes', ...
    'name', 'firstboat', ...
    'color', 'black'), ...
```

The controllers that can be used with the main boat are defined in the *Controller* section of *runScenario.m*. As with the levels, the case label defines the name of the controller. To make them control the main boat, their *target* option must be set to *mainBoat*.

B.4 Creating a new vehicle

The vessel object `objBoat` has been implemented based on the equations in Section 3.1. Using it as a template, vehicles with other equations of motion can easily be implemented.

For speed, the *Update* function in objBoat, which integrates the vessel model and updates the object, has been ported to C. This code can be found in *objBoat_Update.c*.

If a vessel with different characteristics is wanted, this can be implemented directly in objBoat. In its constructor, it has a switch-case statement, selecting model parame-

ters based on the *boatType* option. By adding a case and setting the boatType option accordingly, arbitrary sets of parameters can be used with the vessel model.

Bibliography

[1] The darpa grand challenge. `http://www.darpa.mil/grandchallenge/`. Accessed 2007.12.09.

[2] International regulations for avoiding collisions at sea. `http://www.boatsafe.com/nauticalknowhow/boating/colregs.html`. Accessed 2007.12.09.

[3] Spawar. `http://www.spawar.navy.mil/robots/surface/usv`. Accessed 2007.12.10.

[4] Viknes. `http://www.viknes.no/`. Norwegian. Accessed 2007.12.09.

[5] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. *Proceedings of the ICRA'96*, pages 113–120, 1996.

[6] Jérôme Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[7] K. Bollino, L. Lewis, P. Sekhavat, and I. Ross. Pseudospectral optimal control: A clear road for autonomous intelligent path planning. *Proceedings of AIAA'07*, 2007.

[8] Johan Borenstein and Yoram Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7:278–288, 1991.

[9] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. *Proceedings of the ICRA'99*, pages 341–346, 1999.

[10] James Colito. Autonomous mission planning and execution for unmanned surface vehicles in compliance with the marine rules of the road. Master's thesis, University of Washington, 2007.

[11] Jay A. Farrell and Matthew Barth. *The Global Positioning System & Inertial Navigation*. McGraw-Hill Professional, 1999.

[12] A. E. Fiore, R. E. Anderson, and L. J. Kapanka. Historical approach to collision avoidance. *Journal of The Institute of Navigation*, 1971.

[13] Thor I. Fossen. *Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs, and Underwater Vehicles.* Marine Cybernetics AS, Trondheim, 2002.

[14] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[15] Emilio Frazzoli. Quasi-random algorithms for real-time spacecraft motion planning and coordination. *Acta Astronautica*, 53:485–495, 2003.

[16] A. Fujimori, Y. Ogawa, and P. N. Nikiforuk. A modification of cooperative collision avoidance for multiple mobile robots using the avoidance circle. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, pages 291–299, 2002.

[17] Atsushi Fujimori, Masato Teramoto, P. N. Nikiforuk, and Madan M. Gupta. Cooperative collision avoidance between multiple mobile robots. *Journal of Robotic Systems*, 17:347–363, 2000.

[18] Steven B. Goldberg, Mark W. Maimone, and Larry Matthies. Stereo vision and rover navigation software for planetary exploration. *Proceedings of the 2002 IEEE Aerospace Conference*, pages 2025–2036, 2002.

[19] Petter Ögren and Naomi E. Leonard. A provably convergent dynamic window approach to obstacle avoidance. 2002.

[20] J. Guldner, V. I. Utkin, H. Hashimoto, and F. Harishima. Obstacle avoidance in $\mathbb{R}^n$ based on artificial harmonic potential fields. *Proceedings of the ICRA'95*, pages 3051–3056, 1995.

[21] M. Hebert. Active and passive range sensing for robotics. *Proceedings of the ICRA'00*, pages 102–110, 2000.

[22] Jonas Jansson, Jonas Johansson, and Fredrik Gustafsson. Decision making for collision avoidance systems. 2002.

[23] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on the ICRA'96*, pages 566–580, 1996.

[24] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings of the ICRA'85*, pages 500–505, 1985.

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

[26] Jin-Oh Kim and Pradeep K. Khosla. Real-time obstacle avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation*, 338-349:12, 1992.

[27] Y. Koren and J. Borenstein. Analysis of control methods for mobile robot obstacle avoidance. *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, pages 457–462, 1990.

[28] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1398–1404, 1991.

[29] Kystverket. Automatisk identifikasjons system - ais. `http://www.kystverket.no/?did=9140988`. Norwegian. Accessed 2007.12.11.

[30] Jacoby Larson, Michael Bruch, and John Ebken. Autonomous navigation and obstacle avoidance for unmanned surface vehicles. *Proceedings of the International Society for Optical Engineering*, 2006.

[31] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[32] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[33] Lt. Col. Anthony J. Lazarski. Legal implications of the uninhabited combat aerial vehicle. *Air & Space Power Journal*, pages 74–83, 2001.

[34] Laird-Philip Ryan Lewis. Rapid motion planning and autonomous obstacle avoidance for unmanned vehicles. Master's thesis, Monterey, California, 2006.

[35] Thomás Locano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[36] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[37] L. Matthies, T. Litwin, K. Owens, A. Rankin, K. Murphy, D. Coorobs, J. Gilsinn, T. Hong, S. Legowik, M. Nashman, and B. Yoshimi. Performance evaluation of ugv obstacle detection with ccd/flir stereo vision and ladar. *Proceedings of the ISIC'98; held jointly with the CIRA and ISAS*, pages 658–670, 1998.

[38] Xiao ming Zheng and Masanori Ito. Planning a collision avoidance model for ship using genetic algorithm. *Proceedings on the 2001 International Conference on Systems, Man, and Cybernetics*, pages 2355–2360, 2001.

[39] Xiao ming Zheng, Masanori Ito, and Etsuro Shimizu. Planning and keeping the safe course to avoid collision at sea with genetic algorithms. *Proceedings of the IECON'00*, pages 2388–2393, 2000.

[40] J. Minguez and L. Montano. The ego-kinodynamic space: Collision avoidance for any shape mobile robots with kinematic and dynamic constraints. *Proceedings of the IROS'03*, pages 637–643, 2003.

[41] J. Minguez, L. Montano, and O. Khatib. Reactive collision avoidance for navigation with dynamic constraints. *Proceedings of the IROS'02*, pages 588–594, 2002.

[42] Morten Nielsen and Johannes Petersen. Collision avoidanca at sea - practice and problems. *Proceedings of 20th European Annual Conference on Human Decision Making and Manual Control*, pages 81–90, 2001.

[43] Bohdan Pillich and Gert Büttgenbach. Ecdis - the intelligent heart of the hazard and collision avoidance system. *Proceedings of the 2001 IEEE International Conference on Intelligent Transportation Systems*, pages 1116–1119, 2001.

[44] E. Plaku, K.E. Bekris, B.Y. Chen, A.M. Ladd, and L.E. Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[45] Rudolph van der Merwe. *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. PhD thesis, Oregon Health & Science University, 2004.

[46] Reid Simmons. The curvature-velocity method for local obstacle avoidance. *Proceedings of the ICRA'96*, pages 3375–3382, 1996.

[47] Anthony Stentz. Optimal and efficient path planning for partially-known environments. *Proceedings of the ICRA'94*, pages 3310–3317, 1994.

[48] Chiew Seon Tan, Robert Sutton, and John Chudley. An incremental stochastic motion planning technique for autonomous underwater vehicles. *Proceedings of IFAC Control Applications in Marine Systems Conference*, pages 483–488, 2004.

[49] Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition edition.

[50] Sune Trudslev. Path finding in c#. `http://www.codeproject.com/KB/recipes/csharppathfind.aspx`. Accessed 2007.12.10.

[51] I. Ulrich and J. Borenstein. Vfh+: reliable obstacle avoidance for fast mobile robots. *Proceedings of the ICRA'98*, pages 1572–1577, 1998.

[52] Igor Škrjanc and Gregor Klančar. Cooperative collision avoidance between multiple robots based on bézier curves. *Proceedings of the 29th International Conference on Information Technology Interfaces*, pages 451–456, 2007.

[53] Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers, and Keying Ye. *Probability & Statistics for Engineers & Scientists*. Prentice Hall, 2007.

[54] Wikipedia. A* search algorithm — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/A%2A`. Accessed 2007.12.10.

[55] Wikipedia. Automatic identification system — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Automatic_Identification_System. Accessed 2007.12.10.

[56] Wikipedia. Compass — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Compass. Accessed 2007.12.15.

[57] Wikipedia. Darpa grand challenge — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/DARPA_Grand_Challenge. Accessed 2007.12.16.

[58] Wikipedia. Global positioning system — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Gps. Accessed 2007.12.15.

[59] Wikipedia. Halton sequence — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Halton_sequence. Accessed 2007.12.10.

[60] Wikipedia. Inertial measurement unit — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Inertial_Measurement_Unit. Accessed 2007.12.10.

[61] Wikipedia. Radar — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Radar. Accessed 2007.12.10.