Collision Avoidance for Unmanned Surface Vehicles

Øivind Aleksander G. Loe

Master of Science in Engineering Cybernetics

Submission date:	June 2008
Supervisor:	Morten Breivik, ITK
Co-supervisor:	Vegard Evjen Hovstein, Maritime Robotics

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

The candidate will consider the problem of collision avoidance for unmanned surface vehicles (USVs). The following elements must be considered:

1. Develop a COLREGS-compliant collision avoidance system for USVs that is able to avoid collisions with both static and dynamic objects. Evaluate the performance of the suggested scheme by numerical simulations.
2. Assess the collision avoidance capabilities of the Maritime Robotics Viknes 830 vessel (actuators, sensor suite, computational capacity). Suggest suitable updates to the existing configuration by performance-price appraisals.
3. Implement the suggested collision avoidance system on the Maritime Robotics hardware-in-the-loop (HIL) USV simulator, and evaluate all relevant performance aspects thoroughly.
4. Implement the collision avoidance system on the Maritime Robotics Viknes 830 vessel and perform full-scale experiments in the Trondheim fjord.
5. Evaluate the stability of the closed-loop USV system theoretically, and take into account, e.g., maneuverability considerations in your analysis.

Assignment given: 07. January 2008
Supervisor: Morten Breivik, ITK

Preface

Already before I started working on collision avoidance, I dreamed about what could be achieved with a vehicle able to autonomously navigate its environment. Collision avoidance is a requirement for achieving autonomy, which will become ever more present in the future. The work on designing and developing the collision avoidance system presented in this thesis has been hard but also very rewarding. Being able to try the system on a full-scale vessel and seeing it work in the real world, and not only in the simulations, was all along the big carrot, and it all paid off in the end when the system actually worked.

I would like to express my gratitude to some of the people that made this thesis possible. First of all, my supervisor Morten Breivik, who guided me in the right directions and provided me with a lot of help. The people working at Maritime Robotics in Trondheim; Arild Hepsø, Eirik E. Hovstein, Stein Johansen and Vegard E. Hovstein, generously made their vessels available for testing through the entire period, and were also of great assistance. Last but not least, i would like to thank two people from my graduation class; Åsmund Våge Fannemel for great help on some vessel dynamics issues and control analysis, and Håvard Halvorsen for the collaboration on vessel identification.

Øivind Loe
Trondheim, June 2008

Abstract

Considerable progress has been achieved in recent years with respect to autonomous vehicles. A good example is the DARPA Grand Challenge, a competition for autonomous ground vehicles. None of the competing vehicles managed to complete the challenge in 2004, but returning in 2005, a total of five vehicles were successful. Effective collision avoidance is a requirement for autonomous navigation, and even though much progress has been done, it still remains an open problem.

The focus of this thesis is on the development of a collision avoidance system for unmanned surface vehicles (USVs), which is compliant with the International Regulations for Avoiding Collisions at Sea (COLREGS). The system is based on a modified version of the Dynamic Window algorithm, taking both acceleration and lateral speeds into account for reactive collision avoidance. Path planning is provided by the Rapidly-Exploring Random Tree (RRT) algorithm, extended to use the A* algorithm as a guide, which significantly increases its efficiency.

Extensive simulations have been performed in order to determine the value of the modifications done to the original algorithms, as well as the performance of the total control system. Full-scale experiments have also been carried out in an attempt to verify the results from the simulations. The collision avoidance system performed very well during the simulations, finding near-optimal paths through the environment and evading other vessels in a COLREGS-compliant fashion. In the full-scale experiments, important sensor data was erroneous, resulting in reduced avoidance margins. However, the collision avoidance system still kept the controlled vessel safe, showing significant robustness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Automation has been a part of the human nature for a long time. Whether for making life easier, or to get more work done in less time, we constantly try to find ways to automate processes. In most cases, automation results in higher quality, faster execution and more cost-effectiveness. It is not hard to see why automated processes are taking over.

This trend has been visible for years in the process industry. Ever since the industrial revolution, factories have seen human labour being more or less replaced by automated machines, moving people into surveillance and management of the autonomous control systems.

It is not only factories that are seeing humans replaced by machines however. Vehicle control is slowly but surely relying more and more on automation. The evolution seen in aircraft is a good example of this. In the first planes, humans controlled the plane manually by wires connected to the actuators of the plane. For slow-going aircraft, this is feasible, but as planes got faster and unstable designs became more common, the reaction time of a human was suddenly too slow to be able to control the aircraft. In these systems the pilot gives commands to an aircraft computer, which then tries to control the plane according to the commands of the pilot.

In current fighter-plane designs, humans are often the limiting factor for the manoeuvres possible to execute. To improve the performance further, humans have to be removed from the planes. There are then two possible solutions: The plane can be remotely controlled by a pilot stationed on the ground, or it can be controlled autonomously by an onboard control system, receiving goals from an operator and achieving them on its own.

Because of the opportunities of saving human lives and reducing costs, the military has been at the forefront of development of autonomous vehicles for a long time. From autonomous aerial drones to armed land-based vehicles, such tools can give an important edge on the enemy.

Figure 1.1: The RQ-4 Global Hawk is an unmanned US surveillance aircraft designed to operate autonomously. (Image courtesy of Northrop Grumman [25])

As the technology for controlling vehicles semi- or fully-autonomously matures and gets more available, we will se applications in civilian sectors becoming more common. The DARPA Grand Challenge [1, 54] is an important contributor to civilizing autonomously controlled vessels.

The focus of this thesis is on the development of a collision avoidance system for autonomous unmanned surface vehicles (USV). Being able to operate a vessel autonomously has a lot of benefits. As humans are no longer needed on the vessel, it can have a more compact design and focus on its equipment. The manoeuvrability and efficiency of the vessel is increased, as vessel motions don't have to be reduced for the comfort of the crew. By not carrying other supplies than fuel, and not carrying people, the vessel can operate for extended periods of time. Given enough fuel, for instance by nuclear fission or some future technology, the vessel can potentially operate for years without stop.

The most important benefits however may be those of safety and cost. An unmanned vessel can operate in dangerous areas without the risk of losing human life. This may be anything from reconnaissance in hostile areas to operations in dangerous weather conditions. The issue of cost is what might actually be responsible for the first civilian adoption of autonomous vehicles. Being able to remotely operate a vehicle may severely reduce the number of people required per vessel. With fully autonomous vessels, one could also imagine a single operator having the responsibility for several vessels, reducing the operating costs even further.

One of the main obstacles to autonomously operated vehicles is the lack of a legal framework. For military applications this may be of less concern, but for civilian applications, where the autonomous vehicles have to interact with and avoid other civillian

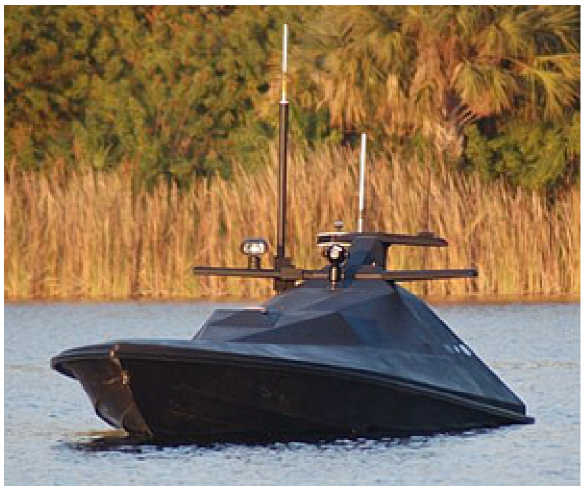


Figure 1.2: The Interceptor is an unmanned surface vessel (USV). Commanded by an on-board computer it is designed to autonomously complete the goals it is given by an operator [49]. Applications for USVs include patrol and surveillance, for instance for harbour security and customs, mapping of the seabed, geological surveys, etc. (Image courtesy of 5G Marine Systems [6])

vehicles, safety and regulatory frameworks must be in place. The legal groundwork for UAVs has come much further than that for other unmanned vehicles, and the Global Hawk shown in Figure 1.1 was the first UAV cleared for routine operation in US national airspace [14]. Much work need to be done before USVs can operate legally under autonomous control however.

Effective collision avoidance is a prerequisite for an autonomous vehicle, since it has to be able to navigate its environment safely. Much work has already been done in the field of collision avoidance, but it is still an open problem. It is in fact impossible to create a system that is able to guarantee 100% collision avoidance. If a *hostile* vehicle deliberately tries to collide with the controlled vehicle, collision avoidance is only possible if the controlled vehicle is more manoeuvrable than the hostile. If the controlled vehicle has a limited endurance however, e.g., is limited on fuel, an escape may not be possible even though it has better manoeuvrability.

There are still major gains to be achieved by higher quality collision avoidance systems. The only way for autonomous vehicles to be commonly accepted is if they are shown to be as safe, or hopefully safer, than vehicles under manual control.

1.2 Contribution

The contribution of this thesis is the development of a collision avoidance system compliant with the *International Regulations for Avoiding Collisions at Sea* (COLREGS). The system is based on the Dynamic Window and Rapidly-Exploring Random Tree (RRT) algorithms, which have both been modified giving a documented improvement. The Dynamic Window algorithm has been modified to include acceleration and lateral movement into its predictions, making it much more suitable for controlling surface vessels, and the RRT has been given multiple extensions to improve its performance. The COLREGS implementation is somewhat based on the ideas presented in [13].

Extensive simulations have been done to show the qualities of the system. Combined with the results from a couple of full-scale experiments, they show that the collision avoidance system works as expected. An analysis has also been done of the system.

1.3 Outline

This thesis is made up of several distinct chapters. Chapter 2 goes through basic theory around collision avoidance and vessel models. It also lists the basic parts of the COLREGS that applies to a collision avoidance system. Chapter 3 discusses the hybrid collision avoidance system developed as part of this thesis, detailing the modifications made to the original algorithms. Chapter 4 shows results from simulations with the collision avoidance system done in Matlab. Chapter 5 goes through the identification of the model for a physical vessel, the target for our full-scale experiments, and gives a short overview of the control-system for the vessel. Results from full-scale experiments, both with a *Hardware-in-the-Loop* simulator and with the physical vessel are given in Chapter 6. A short analysis of the collision avoidance system is done in Chapter 7, while a discussion and final conclusion follows in Chapter 8 and Chapter 9.

1.4 Notation

$\mathbf{x}$	State vector
$\mathbf{u}$	Input vector; vector of manipulated variables
u	Surge velocity of a vehicle
u_{sp}	Set-point for surge velocity
ψ	The heading of a vehicle
ψ_{sp}	Set-point for heading
r	Yaw rate of a vehicle
r_{sp}	Set-point for yaw rate
$\mathcal{W}$	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
CDP	Control Design Platform
CG	Center of Gravity
COLREGS	International Regulations for Avoiding Collisions at Sea
DOF	Degree Of Freedom
DP	Dynamic Positioning
DW	Dynamic Window
GPS	Global Positioning System
ICD	Industrial Control Design
IMO	International Maritime Organization
IMU	Inertial Measurement Unit
LOS	Line Of Sight
ROT	Rate Of Turn
RRT	Rapidly-Exploring Random Tree
TCPA	Time of Closest Point of Approach
USV	Unmanned Surface Vehicle

Chapter 2

Theoretical background

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* [31, 33] to more involved methods as the Rapidly-Exploring Random Tree method [36]. A thorough review is given in [38].

In order to give a better understanding of the presented collision avoidance algorithms, some distinctions must first be made. The most important may be the difference between global and local collision avoidance methods.

2.1.1 Global vs. local methods

Global methods are often referred to as *path planning* or *motion planning* methods [37]. They store information about the surrounding environment in a map, and use this map to find a route from a given *start* to a given *goal*. The goal can either be a set of spatial 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 that is not directly *sensable* from the current state of the vehicle, but exists in a *memory* available to the global method.

The major downside to the global methods is their long computation time. The amount of required processing power 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 react to changes very quickly. The local methods are fast because they only consider the immediate environment of a vehicle. They are also *memoryless*, as they generally have no knowledge of the environment not currently covered by the vessel sensors.

Local methods can make good progress towards the goal, but they can never guarantee that the goal is reached, as they lack global information. 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* [9, 17].

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 illustrated in Figure 2.1, where the global method generates a nominal path which the local method tries to follow.

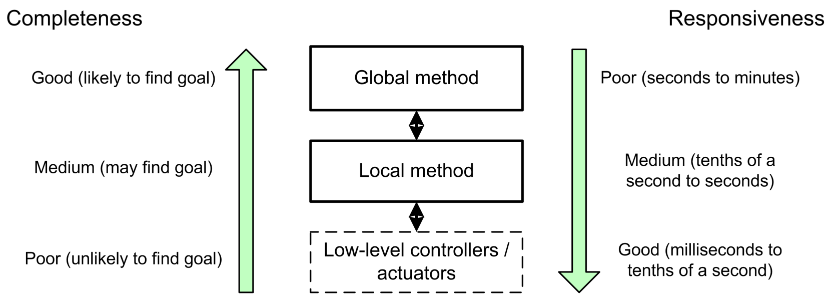

Completeness
Responsiveness
Good (likely to find goal)
Medium (may find goal)
Poor (unlikely to find goal)
Global method
Local method
Low-level controllers / actuators
Poor (seconds to minutes)
Medium (tenths of a second to seconds)
Good (milliseconds to tenths of a second)

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 method's ability to find a solution to a problem if such a solution exists. Low-level controllers are included for reference.

The global method uses global information and memory to ensure global convergence to a given goal. Running at a higher rate than the global method, the local method 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.

The superiority of the hybrid approach was demonstrated by the simulations in [38].

2.1.3 Operating spaces

Another thing worth mentioning is the use of different operating spaces in the collision avoidance algorithms. An operating space is a world representation. We mainly distinguish between two types of operating spaces: The workspace and the configuration space.

The workspace

The physical two- or three-dimensional environment of the vehicle is called the *workspace* $\mathcal{W}$ of the vehicle. The workspace is global in the sense that it applies to all physical objects. Unlike the next space to be presented, it is not specific to any vehicle or dynamic system.

The workspace can be defined as $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$ depending on the number of world dimensions we are working with. 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* $\mathcal{C}$ [37] is required.

Also known as the *C-space*, the configuration space of a physical system encompasses all its possible configurations. A 6 DOF vehicle for instance results in a 6-dimensional configuration space, with one dimension for each system state. A C-space is thus partially specific to a given system, as opposed to the general workspace.

$\mathcal{C}$ also contains regions that are forbidden due to 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:

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

$\mathcal{C}_{obs}$ thus contains all the configurations of our vehicle where it would have been intersecting 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 [40], and the more advanced *Ego-KinoDynamic* space [39] are examples of other spaces that can make collision avoidance even simpler. They are both derived from the configuration space $\mathcal{C}$, and add features such as removing parts from $\mathcal{C}_{free}$ that our vehicle cannot access without entering $\mathcal{C}_{obs}$ at a later time.

2.1.4 Vessel model

Many control systems depend on having a mathematical model of the system to be controlled. For the simpler controllers, the model is used to tune the control system. More advanced controllers however, such as the one to be presented in this thesis, use the model actively to predict how the controlled system will react to control inputs Then the accuracy of the model can have a big impact on the performance of the controller. On the other hand, simple models are often preferred because they can make controller synthesis simpler. A model with less parameters is also preferred because there are less parameters to determine.

One of the simplest surface-vessel models is the 1st-order *Nomoto* model [20] which can be seen in (2.3). The model describes the steering dynamics of a vessel travelling at a given forward speed as a 1st-order transfer function from the rudder angle δ to the yaw rate r of the vessel. While it is not very accurate, it is linear, has only two parameters, and is thus very easy to work with. Examples can be found in [26, 27].

$$\frac{r}{\delta}(s) = \frac{K}{1+Ts}. \tag{2.3}$$

The quality of the model can be improved with gain-scheduling, but in this thesis, a more involved model which captures more of the vessel dynamics will be used.

The equations of motion

The work in this thesis is based on a 3-DOF surface-vessel model, using a total of 6 states to describe the motions of the vessel.

The states of the vessel are given according to the SNAME notation as shown in Table 2.1. They consist of the vessel's generalized coordinates $\boldsymbol{\eta} = [x\ y\ \psi]^T$ given in a world frame, and the velocity of the vessel $\boldsymbol{\nu} = [u\ v\ r]^T$, defined in the body frame of the vessel. The generalized coordinates consist of the spatial coordinates of the vessel $(x,\ y)$ and its heading ψ. The forward/surge/longitudinal speed of the vessel is given by u, while the sideways/sway/lateral speed is given by v. The vessel's rate of turn (ROT), or yaw rate, is given by r.

The equations of motion are given as [20]:

	forces/ moments	**linear/angular speeds**	**positions/ angles**
Motions in the x-direction (surge)	X	u	x
Motions in the y-direction (sway)	Y	v	y
Rotations about the z-axis (yaw)	N	r	ψ

Table 2.1: The notation of SNAME for marine vessels. Only the values applicable for a 3-DOF model are shown. The vessel model in this thesis has the forces, moments and velocities defined in the body frame of the vessel, and the positions and angles in a world frame.

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{R}(\psi)\boldsymbol{\nu} && (2.4) \\ \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \boldsymbol{w} + \boldsymbol{\tau}. && (2.5) \end{aligned}$$

To make this system and its matrices as simple as possible, we let the body origin coincide with the vessel's center of gravity, and the body axes with the longitudinal, lateral and normal symmetry axes of the vessel.

In (2.5), $\boldsymbol{R}(\psi)$ is a rotation matrix used to transform the vessel-fixed velocities to the world frame:

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (2.6)$$

$\boldsymbol{M}$ is the inertia matrix. In addition to rigid-body mass, the mass of the physical vessel, it contains the *added mass* of the vessel: $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$. Added mass is a hydrodynamic phenomenon. It can be understood as pressure-induced forces and moments due to a forced harmonic motion of the vessel body proportional to its acceleration [20]. By assuming an ideal fluid, zero relative velocity to the fluid and zero frequency of motion due to water surface effects, the added mass matrix is constant and strictly positive [44].

The rigid-body inertia matrix of the vessel is given as:

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix}, \qquad (2.7)$$

where m is the vessel mass, and I_z is the moment of inertia about the z-axis of the vessel. The added mass matrix for the simplified case defined above is defined as:

$$\boldsymbol{M}_A = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & N_{\dot{v}} & N_{\dot{r}} \end{bmatrix}, \qquad (2.8)$$

$\boldsymbol{C}(\boldsymbol{\nu})$ is a coriolis and centripetal matrix, and as the mass matrix it is given as $\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_A(\boldsymbol{\nu})$, where the last term counts for added coriolis. The rigid-body coriolis and centribetal matrix is for our system:

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix}, \tag{2.9}$$

and the added coriolis for our system is defined as:

$$\boldsymbol{C}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -Y_{\dot{v}}v + \frac{Y_{\dot{r}}+N_{\dot{v}}}{2}r \\ 0 & 0 & X_{\dot{u}}u \\ Y_{\dot{v}}v + \frac{Y_{\dot{r}}+N_{\dot{v}}}{2}r & -X_{\dot{u}}u & 0 \end{bmatrix}. \tag{2.10}$$

The last matrix, namely $\boldsymbol{D}(\boldsymbol{\nu})$ is a damping matrix. It accounts for hydrodynamic damping effects such as *skin friction* and *vortex shedding.* There are many ways to define the damping matrix, and simplifications always have to be made. In this thesis it is defined as follows:

$$\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{D}_L\boldsymbol{\nu} + \mathbf{D_{NL}}(\boldsymbol{\nu})\boldsymbol{\nu}, \tag{2.11}$$

where $\boldsymbol{D}_L$ accounts for linear damping, and is given as:

$$\boldsymbol{D}_L = -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix}, \tag{2.12}$$

and $\mathbf{D_{NL}}$ accounts for nonlinear damping:

$$\mathbf{D_{NL}}(\boldsymbol{\nu})\boldsymbol{\nu} = -\begin{bmatrix} X_{|u|u}|u|u + X_{uuu}u^3 \\ Y_{|v|v}|v|v + Y_{vvv}v^3 \\ N_{|r|r}|r|r + N_{rrr}r^3 \end{bmatrix}. \tag{2.13}$$

Environmental forces such as wind, waves and currents are represented by $\boldsymbol{w}$. In this thesis such influences are not considered, giving $\boldsymbol{w} = 0$.

The actuators of the vessel influence it through the generalized force vector $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_X \\ \tau_Y \\ \tau_N \end{bmatrix}, \tag{2.14}$$

where τ_X and τ_Y are the forces applied along the longitudal and lateral axes of the vessel, respectively. Torque applied about the z-axis of the vessel is given by τ_N.

The target vessel has only two actuators; a propeller and a 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.

With the propeller generating a force F_x in the longitudal direction, and the rudder generating a lateral force F_y, the generalized forces in (2.14) become:

$$\begin{aligned} \tau_X &= F_x && (2.15) \\ \tau_Y &= F_y && (2.16) \\ \tau_N &= l_r F_y, && (2.17) \end{aligned}$$

where l_r is the distance from the vessel's center of gravity to the *point of attack* of the rudder force. The forces F_x and F_y are available for manipulation by the *low-level controllers.*

Low-level controllers

In most cases, it is not desirable to have higher level controllers such as collision avoidance algorithms controlling the actuators of a vessel directly. It is much more convenient to let those algorithms control values such as desired vessel speed and turn-rate. This is accomplished by having a set of low-level controllers drive the actuators trying to fulfill the commands from the higher level algorithms. Dividing a control system into separate components is often preferred over having a large monolithic controller. This is because the resulting modularity helps reduce complexity and can make controller development more manageable.

The resulting control structure is visualized in Figure 2.2.

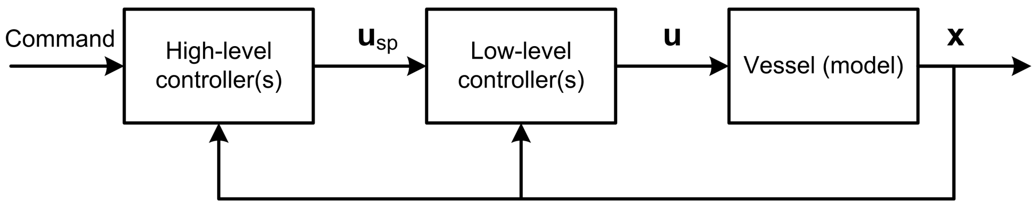


Figure 2.2: This figure shows the control structure employed for the vessel. Starting from the left, high-level controllers receive commands from an external source. Combining this with a state measurement (or estimate) $\mathbf{x}$, the high-level controllers generate set-points $\mathbf{u}_{sp}$ for the low-level controllers, which then provide the vessel with actuator commands $\mathbf{u}$.

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

For controlling the vessel model (2.5) during simulations, the following low-level controllers will suffice. When controlling (u_{sp}, r_{sp}) the controller equations are:

$$\begin{aligned} F_x &= \overbrace{-X_u u - X_{|u|u}|u|u - X_{uuu}u^3 - mrv}^{FF} + \overbrace{K_{p,u}m(u_{sp} - u)}^{P\,FB} && (2.18) \\ l_r F_y &= \underbrace{-N_r r - N_{|r|r}|r|r - N_{rrr}r^3}_{FF} + \underbrace{K_{p,r}I_z(r_{sp} - r)}_{P\,FB}, && (2.19) \end{aligned}$$

Since the exact vessel model, parameters and vessel state is known at all times, these controllers can cancel most of the terms of the vessel model using feed forward (FF), compensating for unwanted motions by prediction before they can be sensed. By removing the nonlinear damping terms and coriolis, the controllers turn the nonlinear vessel model into a linear system, which is controlled by the proportional feedback (P FB) in the controllers.

The act of removing nonlinear elements from a system by feed forward is called *feedback linearization*. The problem with this approach is that it requires a rather accurate system model, so for a real vessel, other controllers must be used.

When controlling (u_{sp}, ψ_{sp}), the yaw rate controller is replaced with the following heading controller:

$$l_r F_y = \underbrace{K_{p,\psi}I_z(\Gamma(\psi_{sp} - \psi) + T_{d,\psi}(\dot{\psi}_{sp} - r))}_{PD\,FB}, \qquad (2.20)$$

which is a general *proportional-derivative* (PD) controller, that does not use any kind of feed forward. The function $\Gamma : \mathcal{R} \rightarrow \langle -\pi, \pi]$ maps the heading error to the interval $\langle -\pi, \pi]$, making sure that the proportional term of the controller always causes the vessel turn the correct way in order to reduce the heading error.

The derivative of the set-point is often set to zero, but when the derivative is known and continuous, using it in the controller can give the system better performance.

2.2 Collision avoidance methods

As already mentioned, many methods for avoiding collisions exist. A review of some of the most important methods is given in [38]. Of the reviewed methods, the combination of the *Dynamic Window* algorithm and the *Rapidly-Exploring Random Tree* algorithm came out on top, neck and neck with the combination of *Dynamic Window* and the *A** algorithm.

A description of these collision avoidance methods is given in the following sections. The methods are presented as they have been described and implemented in the literature, and provide a basis for the collision avoidance methods used in this thesis. To improve the performance of these algorithms, a number of modifications and extensions have been developed, which are presented in Chapter 3.

2.2.1 Local method: 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 [21]. 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 velocity of our vehicle, including linear velocity and rate of turn, is constant within a given time-interval. Neglecting sway motion, the trajectory of our vehicle during a time-interval can then be estimated as a straight line or a constant-radius arc. The arc centers for different velocities can be calculated as:

$$a_x^i = x - \frac{u_i}{r_i}\sin\psi \tag{2.21}$$

$$a_y^i = y + \frac{u_i}{r_i}\cos\psi, \tag{2.22}$$

where u_i is the surge speed and r_i is the rate of turn of the vehicle during interval i, $(x,\ y)$ is the position of the vehicle, and ψ is its heading. The radius of the arc is given by:

$$a_r^i = \left|\frac{u_i}{r_i}\right|. \tag{2.23}$$

Admissible velocities

As sway is neglected, the pair (u_i, r_i) now represents the velocity 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 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:

$$V_a = \left\{u, r \,\middle|\, |u| \le \sqrt{2 \cdot dist(u,r) \cdot \dot{u}_b}, \quad |r| \le \sqrt{2 \cdot dist(u,r) \cdot \dot{r}_{max}}\right\}, \tag{2.24}$$

where $dist(u, r)$ is the distance the vehicle can travel along the given arc before hitting an obstacle, $\dot{r}_{max}$ is the maximum angular acceleration of the vehicle, and $\dot{u}_b$ is the maximum braking acceleration, given by:

$$\dot{u}_b = \begin{cases} \dot{u}_{min}, & u \geq 0 \\ \dot{u}_{max}, & u < 0 \end{cases}, \tag{2.25}$$

where $\dot{u}_{max} > 0$ is the maximum forward acceleration of the vehicle, and $\dot{u}_{min} < 0$ is the maximum reverse acceleration.

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 put on the choice of velocities is that they have to reside in a *dynamic window.* The dynamic window consists of all the velocities that can be reached during the next time interval using the limited accelerations of the vehicle. This window is centered around the current velocities of the vehicle (u_a, r_a). With T_a being the time allowed for acceleration during the next interval, the dynamic window V_d is defined as:

$$V_d = \left\{u, r \mid u \in [u_a + \dot{u}_{min}T_a, u_a + \dot{u}_{max}T_a],\ r \in [r_a - \dot{r}_{max}T_a, r_a + \dot{r}_{max}T_a]\right\}. \tag{2.26}$$

Let $u_{min} \leq 0$ be the maximum reverse speed of the vehicle, $u_{max} > 0$ the maximum forward speed, and r_{max} the maximum angular rate. The set V_s, denoting the velocities reachable by our vehicle can then be defined as:

$$V_s = \left\{u, r \mid u \in [u_{min}, u_{max}],\ r \in [-r_{max}, r_{max}]\right\}. \tag{2.27}$$

The set of valid velocities V_r for the next time interval is then given by (2.28), which constitutes the search space of the algorithm:

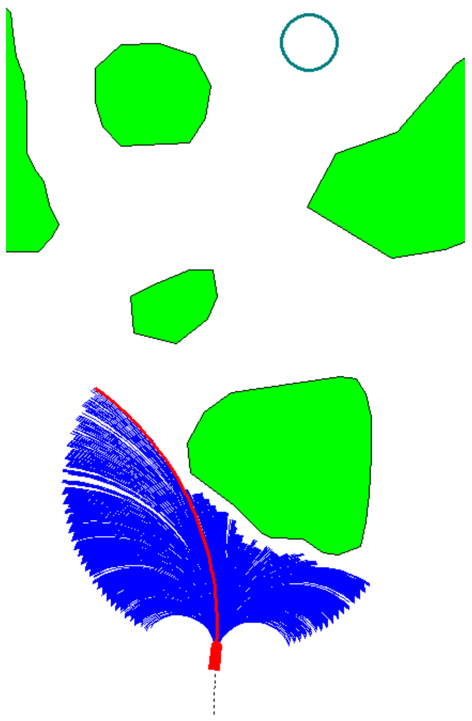
$$V_r = V_s \cap V_a \cap V_d. \tag{2.28}$$

Selecting the optimal velocities

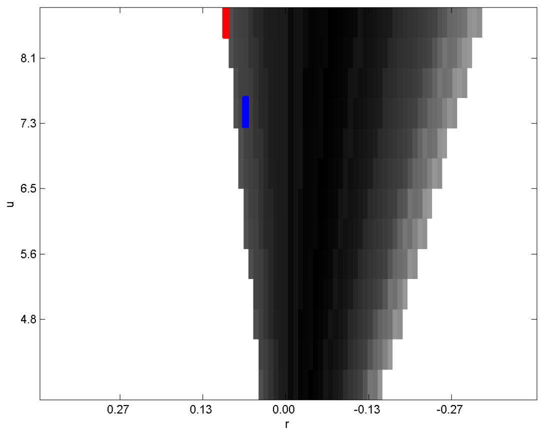
When the search space has been determined, it is time to select the optimal velocities in the space. This is done by maximizing the following objective function over the search space:

$$G(u, r) = \sigma\left(\alpha \cdot \text{heading}(u, r) + \beta \cdot \text{dist}(u, r) + \gamma \cdot \text{speed}(u, r)\right), \tag{2.29}$$

where $\sigma(...)$ is a low-pass filter to help reduce fluctuations. The objective function is a linear combination of the three functions *heading*, *distance* and *speed*, with the importance of the functions being decided by the constants α, β and γ. With correctly chosen weights, maximizing the objective function should make sure our vehicle travels in the right direction while avoiding obstacles and keeping its velocity high.

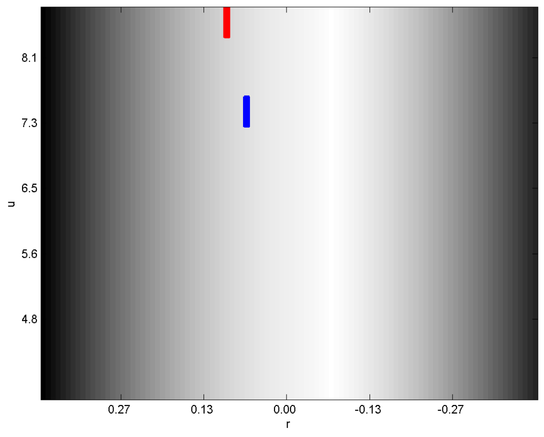


(a) Vessel and obstacles

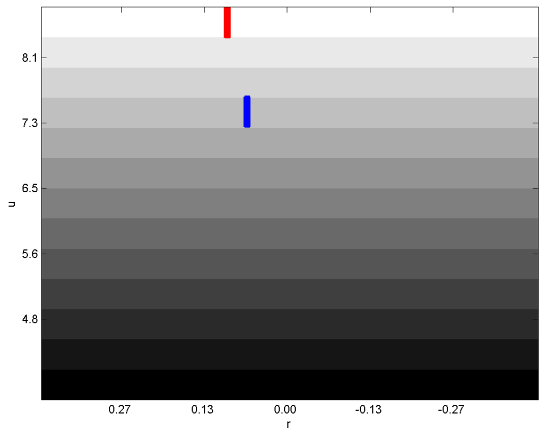


(b) Distance map

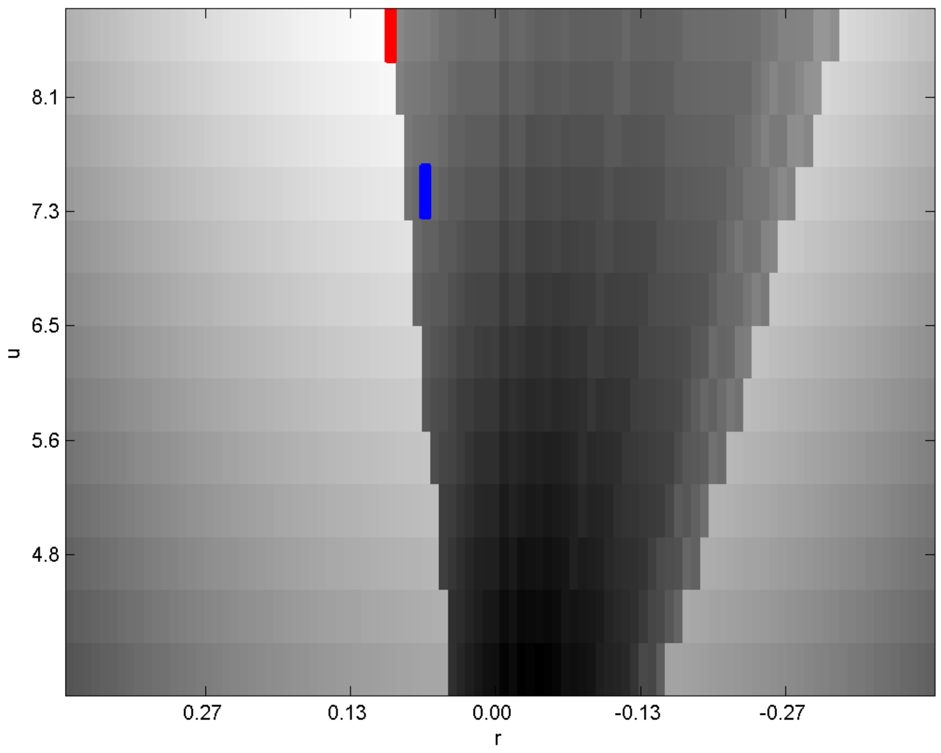
Figure 2.3: In (a) our vessel is navigating the map using the Dynamic Window method. The distance map is shown in (b). The choices of angular rates lie along the x-axis, and the different surge speeds lie along the y-axis. The brighter a cell in the distance map is, the further our vehicle can travel using that velocity command. Brighter is better. The blue rectangle marks the current velocity of the vehicle, and the commanded velocity is marked by the red rectangle. The blue lines in (a) show the predicted trajectories for different velocity choices. The red line shows the chosen trajectory.



(c) Heading map



(d) Speed map



(e) Final dynamic window map

Figure 2.3: The heading-map is displayed in (c), and the speed map is shown in (d). Combined with the distance map, they result in the dynamic window map shown in (e). This map is the basis for the commanded velocity shown in red, ideally resulting in the vessel following the red trajectory in (a).

The distance and speed functions should be self-explanatory. The heading function is also pretty simple, and is given as $180 - \theta$, where θ is the angle between a vehicle heading estimate $\hat{\psi}$ and a desired heading ψ_{sp}. This promotes choices where the heading estimate is closer to the target heading, hopefully making the vessel travel in the target direction.

The vehicle direction $\hat{\psi}$ estimate is calculated as the direction obtained by holding the yaw rate r for a given time $T_r > 0$, and then doing a full angular deceleration using the maximal angular acceleration of the vehicle $\dot{r}_{max} > 0$. With ψ being the current heading of the vehicle, the estimate becomes:

$$\hat{\psi} = \psi + rT_r + \frac{1}{2 \cdot \dot{r}_{max}} r|r|. \tag{2.30}$$

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.

Summary

The Dynamic Window algorithm is a collision avoidance algorithm that only takes the immediate surroundings of the vehicle into account. In contrast to simpler methods such as the *Virtual Force Field* (VFF) [32, 38] and *Vector Field Histogram* (VFH) [9, 32, 38], the method takes vehicle dynamics into consideration when deciding on how to steer the vehicle. Consequently it requires a good deal more computational power than these simpler methods, but also gives better results.

The Dynamic Window algorithm tries to keep a given target heading. As shown in [38], the algorithm is often able to lead a vehicle to a goal in a simple environment. An example of this is shown in Figure 2.3. For more complex environments however, the lack of memory and a global view of the environment causes problems, often leading the algorithm into local minima in the environment, making further progress impossible.

When used in combination with a global path-planner, responsibility of the Dynamic Window algorithm is *dynamic* collision avoidance along a generated path.

2.2.2 Global method: RRT

The *Rapidly-Exploring Random Tree* method (RRT), was introduced by LaValle in 1998 [36]. 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 [47], while Frazzoli uses it for spacecraft motion planning in [22].

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 workspace-coordinates of our vehicle, but all its states, i.e., coordinates, heading and velocities. The result is a path through the C-space of the vehicle, or more specifically the admissible part of the C-space, namely $\mathcal{C}_{free}$. If a path has been generated by the RRT method, we thus know it is a feasible path, at least if no changes have occurred in the environment after the path was generated.

The algorithm

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

```
1: $\mathcal{T} \leftarrow$ TREE($\{\mathbf{x}_{init}, 0\}$)
2: if CAN_CONNECT($\mathbf{x}_{init}$, $\mathbf{x}_{goal}$) then
3:     $\mathcal{T} \leftarrow \{$CONNECT_STATE(), CONNECT_INPUT()$\}$ as child of $\mathbf{x}_{init}$
4:     goal found
5: end if
6: for $k \leftarrow 1 \ldots N$ do                                   ▷ The main loop
7:     $\mathbf{m}_c$ = CANDIDATE_MILESTONE()
8:     if $\mathbf{m}_c \in \mathcal{C}_{free}$ then
9:         $\mathbf{x}_{near}$ = NEAREST_NEIGHBOUR($\mathbf{m}_c$, $\mathcal{T}$)
10:        if CAN_CONNECT($\mathbf{x}_{near}$, $\mathbf{m}_c$) then
11:            $\mathbf{u} \leftarrow$ CONNECT_INPUT()
12:            $\mathbf{x}_{next} \leftarrow$ CONNECT_STATE()
13:            $\mathcal{T} \leftarrow \{\mathbf{x}_{next}, u\}$ as child of $\mathbf{x}_{near}$
14:            if CAN_CONNECT($\mathbf{x}_{next}$, $\mathbf{x}_{goal}$) then
15:                $\mathcal{T} \leftarrow \{$CONNECT_STATE(), CONNECT_INPUT()$\}$ child of $\mathbf{x}_{next}$
16:                goal found
17:            end if
18:        end if
19:    end if
20: end for
21: if goal found then
22:     return cheapest path from $\mathbf{x}_{init}$ to $\mathbf{x}_{goal}$ in $\mathcal{T}$
23: else
24:     return cheapest path from $\mathbf{x}_{init}$ taking us closest to $\mathbf{x}_{goal}$
25: end if
```

The concept of *milestones* is the heart of the algorithm. At each iteration, a *candidate milestone* $\boldsymbol{m}_c$ is chosen. It represents a potentially *unexplored* configuration of the controlled vehicle, for instance an unvisited location in the environment. The algorithm then tries to predict whether it can get the vehicle to $\boldsymbol{m}_c$ from any of the configurations previously explored, i.e., the configurations currently in the *tree*. If the algorithm manages to do this, the candidate is added to the tree as an explored milestone/node. An attempt is then made to *connect* this new node to the goal configuration $\mathbf{x}_{goal}$.

CAN_CONNECT($\mathbf{x}$, $\mathbf{m}_c$) is the function that does most of the work. It uses a local motion planner to try to connect the vehicle state/configuration $\mathbf{x}$ with the new candidate milestone $\mathbf{m}_c$. For the local planner, a compromise has to be made between its accuracy and computational complexity. An accurate planner will more often be able to connect $\mathbf{x}$ with $\mathbf{m}_c$, while a fast planner will be able to make more attempts with different candidate milestones in the same amount of time.

A vehicle state is connected to a milestone by integrating the vehicle from $\mathbf{x}$ while applying a series of inputs $\mathbf{u}$. If the vehicle comes within a predefined distance of $\mathbf{m}_c$ within a given time and without exiting $\mathcal{C}_{free}$, 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 $\mathbf{m}_c$.

CANDIDATE_MILESTONE() 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($\mathbf{m}$, $\mathcal{T}$) returns the node in $\mathcal{T}$ whose state vector is closest to the milestone $\mathbf{m}$ by some metric. Using the Euclidean distance metric is common since 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 found.

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 the problem is to re-use the previously-generated path, which 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.

Summary

The RRT method finds a path through the environment by trying a number of different paths and selecting the most optimal. The algorithm takes vehicle dynamics into account, so we can be sure the path it generates is feasible, i.e., possible to follow for the controlled vehicle.

The algorithm actually tries to optimize an objective function when generating a new path, so by manipulating the objective function, the properties of the final path can be manipulated in various ways. This makes RRT a very flexible algorithm. The objective can for instance be manipulated to favor time optimality, fuel optimality, shortest path, etc.

Figure 2.4 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 going on around the RRT algorithm, and attempts are still made to improve it. A promising method showing off good results can be found in [42]. The method is well suited for parallel processing systems, which is a property that will become more important in the years to come.

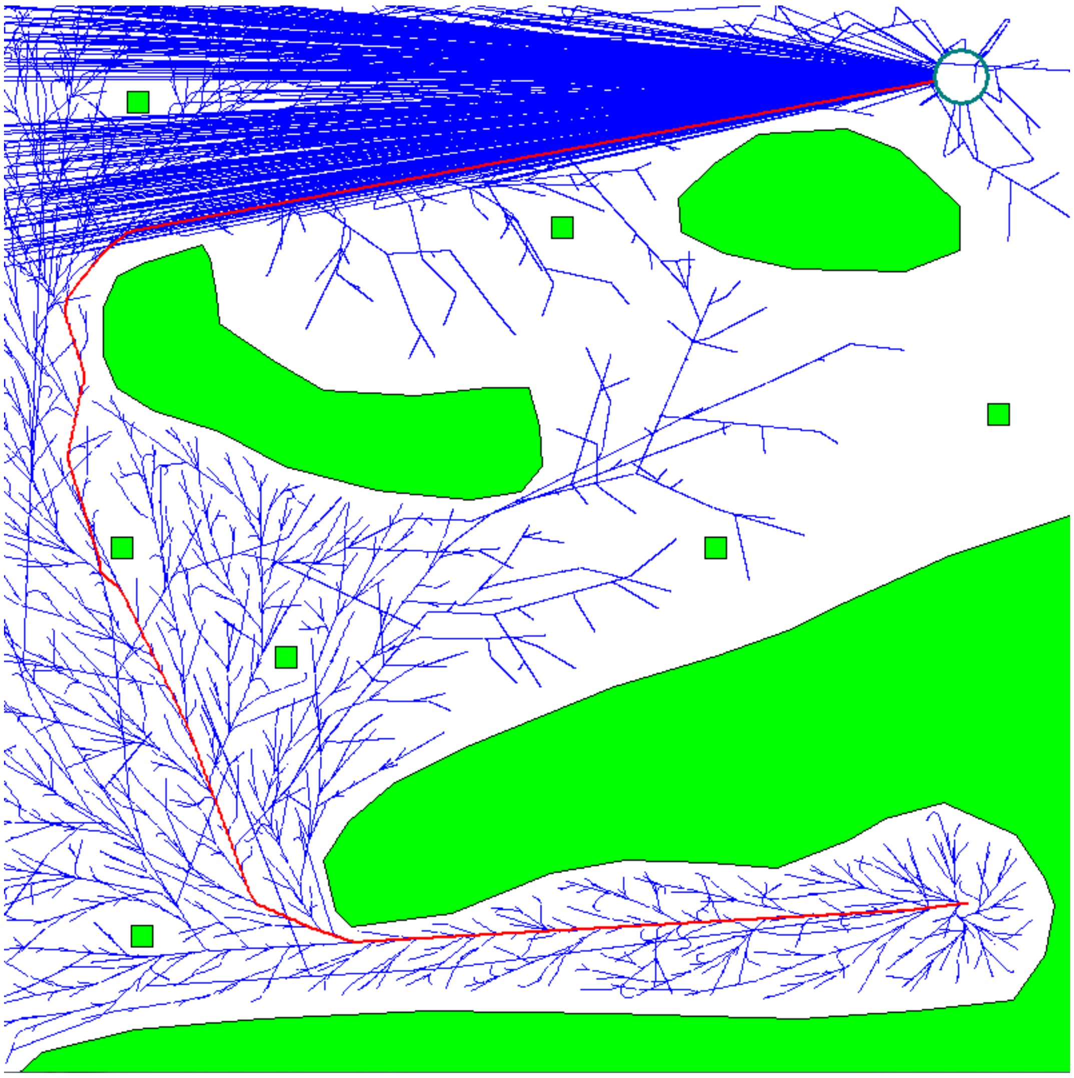


Figure 2.4: The figure shows a path through an environment generated by the RRT algorithm. The blue tree structure originating at the vessel in the lower right shows the paths explored by the RRT algorithm while searching the configuration space. The red line shows the best of the explored paths ending at the goal in the upper right. It is not optimal, but not that bad either.

2.2.3 Global method: 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 [48, 52].

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 rectangular 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.

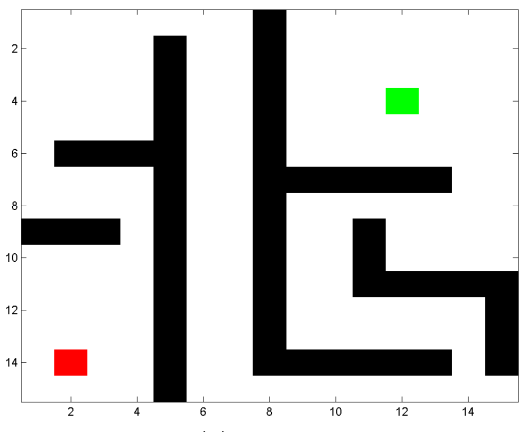
This representation can easily be extended to three or more dimensions, to include height information, time, etc. Note however, that the memory requirement grows exponentially with the number of dimensions included.

The A* algorithm

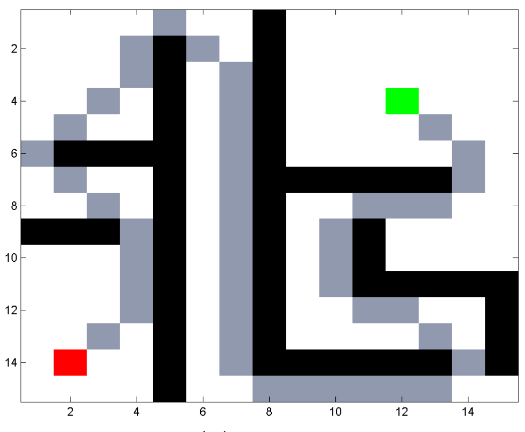
The algorithm starts exploring the map at the start coordinates. It uses a *closed* and an *open* set. The closed set contains all the nodes that the algorithm knows a path to, while the open set contains all the nodes that have been encountered but not yet explored. At every iteration, the algorithm 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 and the explored node is added to the closed. Figure 2.5 shows the A* algorithm solving a general path-planning problem.

```
1:  $closed \leftarrow \emptyset$
2:  $cost(start) \leftarrow 0$
3:  $open \leftarrow$ MIN_HEAP($start$)                          ▷ Sorted by $h(node) + cost(node)$
4:  while $open \neq \emptyset$ do
5:      $node \leftarrow$ EXTRACT_MIN($open$)
6:      if $node = goal$ then
7:          return GET_PATH($node$)                       ▷ Minimal path to goal found
8:      end if
9:      for $nb \leftarrow$ GET_NEIGHBOURS($node$) do
10:         $cost \leftarrow cost(node)+$ MOVECOST($node$, $nb$)          ▷ Total cost to $nb$
11:         if $(nb \in open) \wedge (cost \leq cost(open(nb)))$ then
12:             continue                                 ▷ Goto next neighbour
13:         end if
14:         if $(nb \in closed) \wedge (cost \leq cost(closed(nb)))$ then
15:             continue                                 ▷ Goto next neighbour
16:         end if
17:         $parent(nb) \leftarrow node$
18:         $cost(nb) \leftarrow cost$
19:         $h(nb) \leftarrow$ heuristic estimate of minimal distance from $nb$ to goal
```

```
20:         $open \leftarrow nb$
21:     end for
22:     $closed \leftarrow node$
23: end while
```



(a) Problem



(b) Solution

Figure 2.5: (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).

Completeness, admissibility and computational optimality

A* is a complete algorithm, meaning that it will always find a solution if one exists. This is because it will look for a path until the entire search-space has been searched, which is possible because the search-space is discrete and of limited size. Whether a solution actually exists in our case depends on the 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, i.e., never over-estimates the cost to the goal, then A* will always find the optimal path from the start to the goal with respect to the cost of the path.

Summary

The A* method is able to calculate the shortest path from the controlled vehicle to the goal through an environment graph, using limited computational power, but possibly

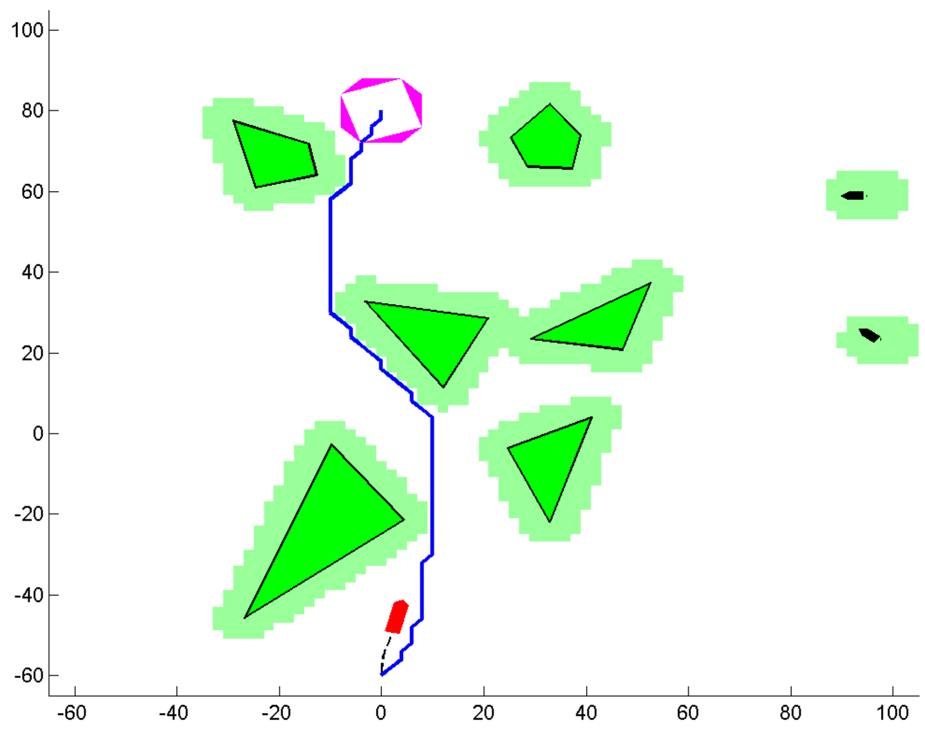
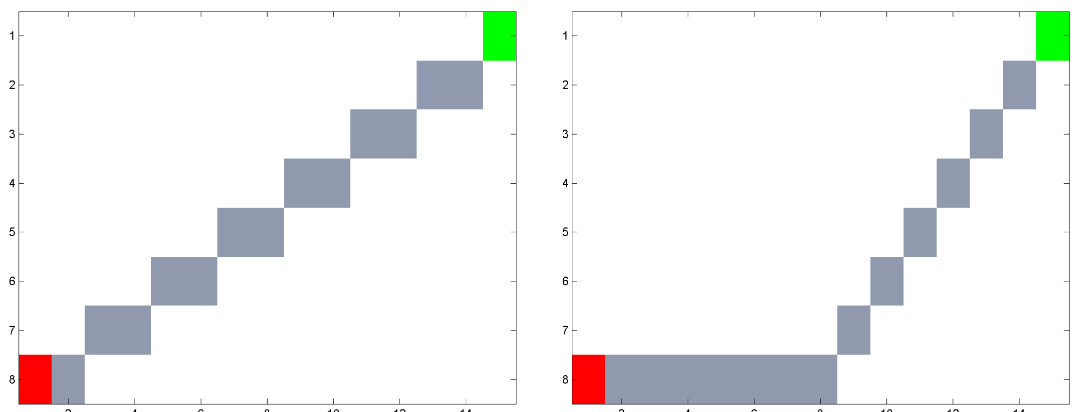


Figure 2.6: The figure shows a path through the map generated by the A* algorithm. The path is generated in the *configuration space* $\mathcal{C}$ of the vessel. The colored regions around the obstacles are part of $\mathcal{C}_{obs}$, and can thus not be traversed by the A* algorithm. The limited resolution of the environment map can clearly be seen in the jagged path generated by the algorithm.

large amounts of memory. Whether this path is actually the shortest path through the environment depends on the type and resolution of the graph.

A* uses magnitudes less computation time than the RRT algorithm, as it does not take vehicle dynamics into account when generating a path. While the efficiency of A* makes the algorithm a good choice, the neglection of vehicle dynamics can be a problem for vehicles with limited manoeuvring capabilities, as the algorithm may construct paths that the vessel is unable to follow. This is especially true when navigating cluttered environments. The algorithm is therefore not complete for the vehicle control problem, as it may generate infeasible paths. Another problem is the algorithm's limited concept of distance, as shown in Figure 2.7.

The A* algorithm has been used extensively as the deliberate, path planning part of larger navigation and collision avoidance systems, and an example can be found in [35]. An example showing the A* algorithm guiding a vesel through an environment is shown in Figure 2.6.



(a) Good **(b)** Poor

Figure 2.7: For the A* algorithm, the paths shown in (a) and (b) are of equal length. For the controlled vessel however, (a) is preferred, because it is straighter on average. When following (a), the vessel will most likely travel in a near stright line from start to finish, evening the path. Travelling along (b) however will take the vessel detour. The inability of the A* algorithm to destinguish between the two can result in the A* algorithm producing less-than-optimal paths.

2.3 COLREGS

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 eventually became the *International Regulations for Avoiding Collisions at Sea*, or simply COLREGS. The regulations were adopted by the International Maritime Organization (IMO) member countries on 20 October 1972, and brought into force on 15 July 1977 [2, 19, 59].

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 has been steadily evolving, e.g., by recognizing the radar as an important aid in collision avoidance and including it in the regulations.

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.

2.3.1 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 2.8.

> 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.*

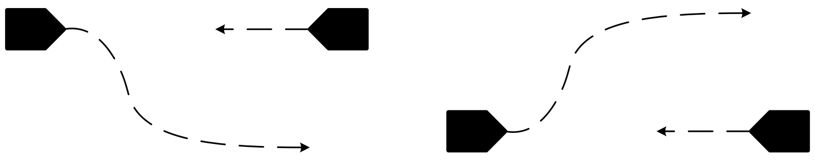

(a) Correct pass **(b)** Incorrect pass

Figure 2.8: 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.

2.3.2 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 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 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 stern light of that vessel but neither of her sidelights.*

2.3.3 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.*

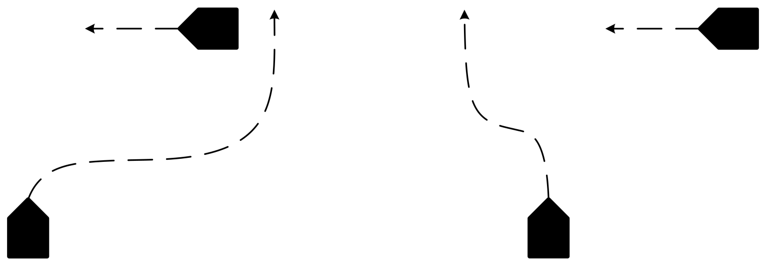

(a) Correct crossing **(b)** Incorrect crossing

Figure 2.9: Our vessel approaches the other vessel from the bottom of the figure, and (a) shows the correct course of action. Another option is to slow down, letting the other vessel pass before crossing.

2.3.4 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 this vessel's responsibility to avoid a collision, but as long as a collision can be safely avoided in good time by actions of the give-way 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 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-sized vessels. The essence of the rules is that the most manoeuvrable vessel should take responsibility and avoid collision with the other vessel.

> 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.*

2.3.5 Action to avoid collision

As seen, COLREGS requests specific actions depending on what type of situation a vessel encounters, and also defines some general rules related to these actions. Rule 8b aims to reduce uncertainty when in a situation with another vessel:

> Rule 8b:
> *Any alteration of course and/or speed to avoid collision shall, if the circumstances of the case admit, be large enough to be readily apparent to another vessel observing visually or by radar; a succession of small alterations of course and/or speed should be avoided.*

Rule 8d requests that the action taken should provide a buffer zone between the vessels, and that the situation should be monitored carefully. When in a situation with a vessel unwilling to comply with COLREGS, e.g., with a hostile ship, this rule gives our vessel a better chance of avoiding a potentially dangerous situation:

> Rule 8d:
> *Action taken to avoid collision with another vessel shall be such as to result in passing at a safe distance. The effectiveness of the action shall be carefully checked until the other vessel is finally past and clear.*

2.3.6 Summary

Of the rules defined by COLREGS, the *head-on* and *crossing* rules are the main to consider when developing a COLREGS-compliant collision avoidance system. These rules define very specific courses of action to apply in the given situations and are thus easy to concretize.

The *overtaking* rule on the other hand does not specify any course of action, except from keeping clear of the other vessel. Keeping clear of other vessels is an underlying criterion for any collision avoidance system, so this rule will not be considered further.

Providing significant alterations in course and/or speed to signal the intended course of action to the other vessel is important. As is keeping a safe distance to other vessels and continuously monitoring the situation. A COLREGS-compliant collision avoidance system should follow these rules whenever possible.

Chapter 3

A novel hybrid collision avoidance concept

The simulation results from [38] declared the combination of *Dynamic Window* and *Rapidly-Exploring Random Tree*, and the combination of *Dynamic Window* and *A** the best of the reviewed collision avoidance systems. In both the systems, the Dynamic Window algorithm successfully performs its duties as a local collision avoidance component. However, there is a significant difference between the characteristics of the global components.

The advantage with the A* algorithm is that it consistently finds a short route for the vessel using very little processing power, but this comes at a cost. The A* algorithm has no way of knowing whether or not the vessel is actually capable of following the generated path, and also lacks the ability to determine the time along the path, i.e., when our vessel will be at the different locations along the path. This makes avoidance of dynamic objects by prediction difficult.

On the other hand, the RRT algorithm simulates the vessel along all examined paths, guaranteeing that the paths it generates can be followed by the vessel. Exact time is also known along the path, making prediction and avoidance of dynamic objects easy. The downside of the RRT algorithm is that it selects paths at random, and will thus only find the optimal path given an infinite amount of processing time.

The ability of the RRT algorithm to take dynamic objects into account gives it a higher compatibility with the Dynamic Window algorithm, as both algorithms act in the same way to objects in the environment. Theoretically, this tighter integration should make it the better choice, as the RRT algorithm selects a rough but good path through the environment and lets the Dynamic Window improve on it. A* on the other hand has a higher probability of choosing paths that are incompatible with the Dynamic Window algorithm, forcing the Dynamic Window to find a new path on its own.

The controller developed as a part of this thesis tries to incorporate the best of the two approaches. As seen in Figure 3.1, the new hybrid makes use of both the A* and RRT global algorithms, as well as the local Dynamic Window algorithm. The new algorithm also attempts to follow the rules defined by the COLREGS as best it can.

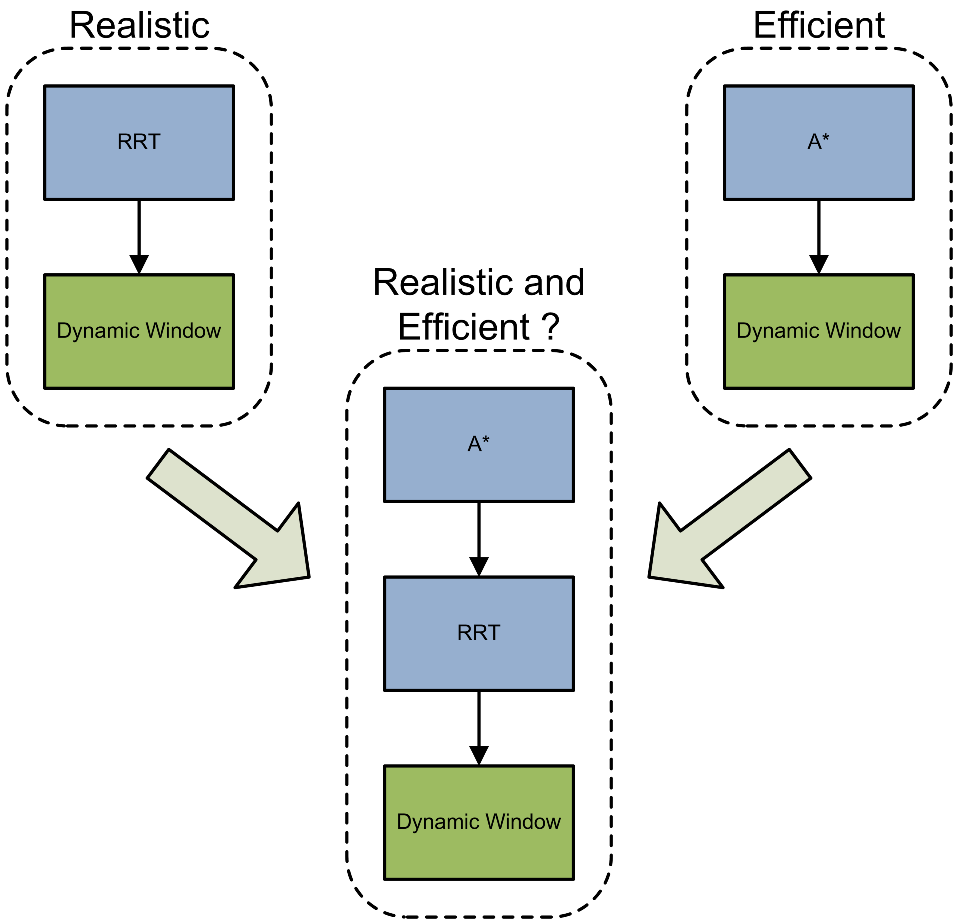


Figure 3.1: The RRT algorithm uses vehicle dynamics and different cost functions to generate a path that is known to be feasible and have a given set of properties. This is very time-consuming however. The A* algorithm on the other hand is able to generate a relatively optimal path through the environment very quickly. The problem with A* is that the path may not be feasible. By combining these approaches, an algorithm that is both efficient and produces feasible paths may be created.

3.1 Local collision avoidance

As explained in 2.1.1, local methods are all about reaction time. They need to be fast enough to react to unforeseen events in the environment, but still be sophisticated enough to make good control decisions. As seen in [38], there are many local methods that are fast enough, and most work well under normal conditions. Serious weaknesses appear in the simpler algorithms under more difficult conditions however, and many are difficult to extend with advanced features like COLREGS compliance.

The *Dynamic Window* algorithm performs the duties of the local controller in this hybrid because it showed good performance in [38]. Being based on the evaluation of a utility function, it is also easy to extend the controller to implement additional features.

The controller is implemented as explained in Section 2.2.1 with details and modifications as covered in the following sections.

3.1.1 No longer on arcs

The original Dynamic Window algorithm predicts vehicle motion as movement along constant-radius arcs at constant velocities. The algorithm expects the vehicle to achieve an instant jump in its velocities from (u, r) to the velocities along the arcs (u_i, r_i). This assumption results in a problem: For the prediction to be accurate, the choices of (u_i, r_i) must lie very close to the current velocities of the vehicle, preferably within the velocities reachable by the vessel within a short time interval T_a. As the arcs predict the motion of the vehicle over a much larger interval, this severely limits the choices available to the controller, since its commanded velocities must come from the set of tested velocities.

In [38], the set of velocity choices was expanded by artificially increasing T_a, increasing the dynamic window of the algorithm. The downside to this however was severely reduced prediction accuracy for velocities some distance from the current velocities of the vehicle. The increased window size gave good results, but simulations have shown that in situations where abrupt changes in velocities are required, the mispredictions significantly increase the probability of collision.

When applied to the control of surface vessels, the Dynamic Window algorithm also has another flaw. It does not take lateral speeds into account when predicting vehicle trajectories. This is ok for wheel-based vehicles such as cars, as they experience minimal lateral speed during normal operation, but for a vessel it can be devastating.

To make the control algorithm safer, the predictions had to be improved. The increased dynamic window is required for good performance, so reducing T_a was not an option. Another possibility was to include vessel accelerations in the dynamic window predictions, going away from constant velocity arcs. An estimate of lateral speed was also included in the controller, providing far more accurate predictions. The result is

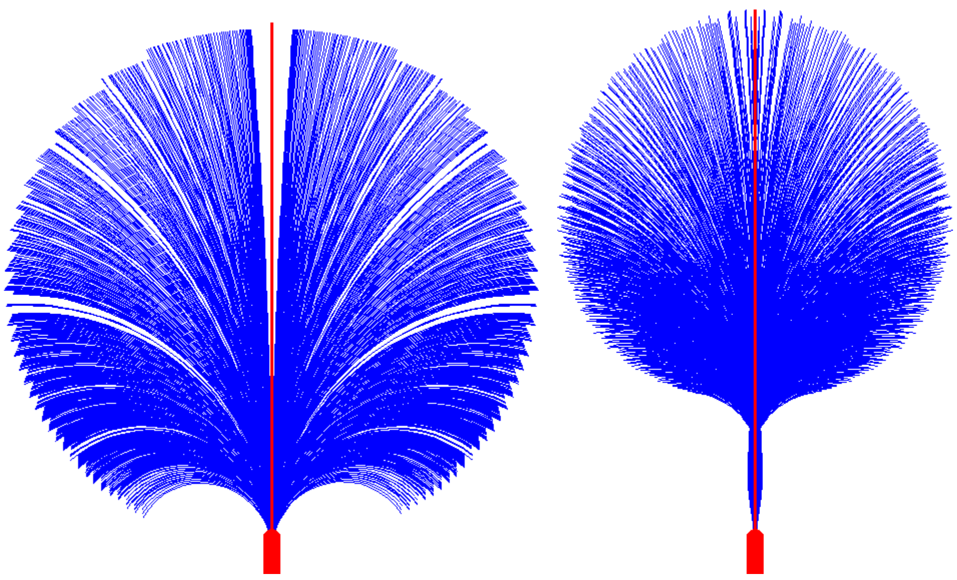

(a) Original **(b)** Modified

Figure 3.2: In the original Dynamic Window algorithm, movement is approximated as constant radius, constant velocity arcs, as seen in (a). This either results in large inaccuracies, or a very small dynamic window. To increase the accuracy of the Dynamic Window algorithm, lateral velocities and vehicle accelerations are included in the predictions. As seen in (b), this gives a more realistic and accurate approximation. The vessel is travelling forward at 10 m/s in both figures.

shown in Figure 3.2.

The new predictions are based on the assumption that the vessel will use its maximum accelerations to reach the commanded velocities (u_i, r_i). From the model in Section 2.1.4, the maximum accelerations can be estimated as:

$$\dot{u}_{max}(u) = \frac{1}{m}\left(F_{x,max} + X_u u + X_{|u|u}|u|u + X_{uuu}u^3\right) \tag{3.1}$$
$$\dot{r}_{max}(r) = \frac{1}{I_z}\left(F_{y,max}l_r + N_r r + N_{|r|r}|r|r + N_{rrr}r^3\right). \tag{3.2}$$

The lateral speed of the vessel is estimated as the steady-state speed at given values of u and r. It is assumed that the damping in sway can be described using only the first and second order damping parameters, giving $Y_{vvv} = 0$. The lateral speed can then be estimated as:

$$\hat{v}(u,\ r) = \operatorname{sgn}(mur - Y_r r)\left[\frac{Y_v + \sqrt{Y_v^2 - 4Y_{|v|v}|mur - Y_r r|}}{2Y_{|v|v}}\right] \tag{3.3}$$

Square roots are generally expensive in terms of the processing power needed to calculate them. The introduction of lateral speed estimation increased the calculation time of the Dynamic Window algorithm by about 20%. In applications where this is not tolerable, approximations to the square root, such as a Taylor series expansion could be used.

To reduce the processing time needed to simulate the vessel along the trajectories determined by the different combinations of (u_i, r_i), the simulation time-step T_s is set to the lowest value required to make sure the vessel does not skip cells in the world map, failing to observe a collision. At high angular velocities, the system becomes relatively stiff, so to improve the prediction of the vessel trajectories, the *modified Euler method* [16] is used for integration.

The modified Euler method uses the derivatives halfway through the integration step when integrating a differential equation. For reference, the traditional Euler method uses the derivatives at the beginning of the integration step. For a vessel travelling in a circle, the error in the Euler integrator would have given a position prediction spiraling outwards, as the derivatives change during the integration step. By using the derivatives halfway through the step, the modified Euler method estimates the average derivative through the step, generally giving a much better prediction. In the case of a perfect circle, the prediction should be exact, and as the paths in the Dynamic Window approach circles, the modified Euler method should be superior to the standard Euler method.

The heading and velocities of the vessel halfway into the integration step is approximated as:

$$\begin{aligned}
\psi_{k+\frac{1}{2}} &= \psi_k + r_k\frac{T_s}{2} + \dot{r}_k\left(\frac{T_s}{2}\right)^2 && (3.4)\\
u_{k+\frac{1}{2}} &= u_k + \dot{u}_k\frac{T_s}{2} && (3.5)\\
r_{k+\frac{1}{2}} &= r_k + \dot{r}_k\frac{T_s}{2} && (3.6)\\
v_{k+\frac{1}{2}} &= \hat{v}(u_{k+\frac{1}{2}},\ r_{k+\frac{1}{2}}). && (3.7)
\end{aligned}$$

The vessel states can now relatively accurately be integrated using the following equations:

$$
\begin{aligned}
x_{k+1} &= x_k + \left(u_{k+\frac{1}{2}} \cos\left(\psi_{k+\frac{1}{2}}\right) - v_{k+\frac{1}{2}} \sin\left(\psi_{k+\frac{1}{2}}\right)\right) T_s && (3.8)\\
y_{k+1} &= y_k + \left(u_{k+\frac{1}{2}} \sin\left(\psi_{k+\frac{1}{2}}\right) + v_{k+\frac{1}{2}} \cos\left(\psi_{k+\frac{1}{2}}\right)\right) T_s && (3.9)\\
\psi_{k+1} &= \psi_k + r_{k+\frac{1}{2}} T_s && (3.10)\\
u_{k+1} &= u_k + \dot{u}_k T_s && (3.11)\\
r_{k+1} &= r_k + \dot{r}_k T_s. && (3.12)
\end{aligned}
$$

The initial condition for the integration is the current vessel state. The accelerations $\dot{u}_k$ and $\dot{r}_k$ are chosen such that they provide the maximal acceleration as given by $\dot{u}_{max}(u)$ and $\dot{r}_{max}(r)$, but they are reduced when u_k and r_k approach their goals u_i and r_i to avoid overshoots.

The Dynamic Window from (2.26) is also redefined to allow different ranges for the linear and angular velocities:

$$V_d = \{u, r \,|\, u \in [u_a - \dot{u}_{max} T_{a,u}, u_a + \dot{u}_{max} T_{a,u}]\,,\ r \in [r_a - \dot{r}_{max} T_{a,r}, r_a + \dot{r}_{max} T_{a,r}]\}\,. \tag{3.13}$$

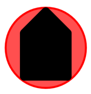
where $\dot{u}_{max} = \dot{u}_{max}(u_a)$ and $\dot{r}_{max} = \dot{r}_{max}(r_a)$. As well as making the window more realistic, this scheme allows the resolution of surge speeds in the dynamic window to be reduced without spreading the velocities too much by also reducing $T_{a,u}$.

3.1.2 World representation

The dynamic window approach uses a configuration space representation of the world. Our vessel has three states, resulting in a C-space $\mathcal{C} \in \mathbb{R}^3$. To simplify the 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 $\mathcal{C} \in \mathbb{R}^2$. This approach is the same as pretending that the vessel is a circle centered in the origin of the vessel with a radius that covers the entire vessel. Configurations where this circle intersects obstacles are not allowed. The downside to this scheme 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 vessels, i.e., vessels with a large *length-to-breadth* L/B ratio, this is unnecessarily conservative as shown in Figure 3.3, but normally it should not be a problem.

The modified C-space is implemented in practice by expanding all obstacles with the radius of the controlled vehicle as shown in Figure 3.4. A map is generated of this C-space, and the vessel center is not allowed to enter any of the expanded objects. This constraint will avoid collisions, at least with static obstacles, but the controlled vehicle


(a) Small error

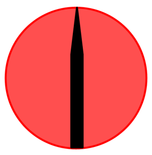
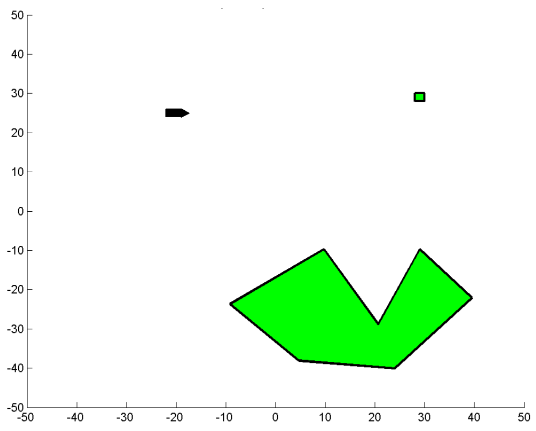

(b) Large error

Figure 3.3: The error in the C-space approximation depends on the *length-to-breadth* ratio of the vessel. For the vessel in (a), the circle approximation is good, resulting in a small error. For the vessel in (b), the error is quite large as shown by the size of the red area.

is allowed to travel arbitrarily close to them. The next step is to add an *unsafe* region to the map, where the vessel is allowed to enter, but only if absolutely necessary.


(a) Workspace

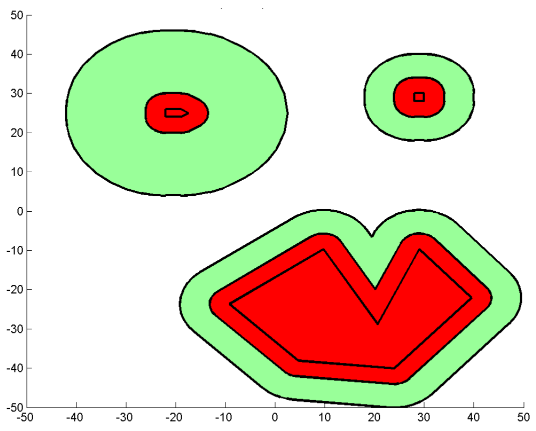

(b) Collapsed C-space with safety reagion

Figure 3.4: The environment, or workspace, of the vehicle is shown in (a). This is the physical representation as we know it. In (b), the red areas show the corresponding collapsed configuration space of the vehicle. The outer colored area is a safety-region which the controlled vessel should steer clear of if possible.

3.1.3 Prediction

To make the collision avoidance method more effective, prediction of dynamic obstacles (other vehicles) is introduced. It is assumed that our vehicle knows the position, heading and velocity of all other vehicles 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.

However our vehicle can normally not predict the future steering commands of the other vehicles. An exception is with cooperating vehicles [23, 24, 38, 50]. 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 so 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 is used in this implementation. For all other vehicles, it is assumed that $r_{sp} = 0$ and $u_{sp} = u$. Then the vehicle states are integrated using the vessel model of Section 2.1.4. Our collapsed C-space is then expanded to include a *time dimension*, using the predictions of the other vehicles for future times $t > t_0$.

As time is known along the predicted paths in the dynamic window, these can now be collision tested against the new three-dimensional $\mathcal{C}$, providing us with simple, yet effective prediction. The shorter the prediction window, the more accurate the prediction will be, and since the prediction window of the Dynamic Window algorithm is relatively short, the prediction should be sufficiently accurate in most cases.

3.1.4 Following the rules of the road

A strength of the Dynamic Window algorithm is its utility function. By adding terms to the function, it is easy to modify the behaviour of the controller, and this is exactly what is done to implement COLREGS compliance.

To be able to apply the rules defined by COLREGS to an autonomous vessel, they have to be concretized. Till now it has always been up to the vessel captain to use his good judgement to determine whether the vessel is approaching a collision situation or not, and decide on a corresponding action. The decisions of the captain are based on available information, regulations and experience. When a computer is in charge of a vessel, the experience of the captain must be replaced with a set of rules for the computer to follow.

The collision cone

The regulations given in Section 2.3 are a means of resolving situations where two vessels are on a collision course. We therefore need a way of determining whether the controlled vessel is on collision course with another vessel or not. The *Collision Cone Approach*, introduced in 1998 by Chakravarthy and Ghose [11], provides an elegant method for solving this problem.

Figure 3.5 shows the geometry of the collision cone approach. The collision cone can be used to predict a collision between a point and a circle moving at constant velocities.

Given

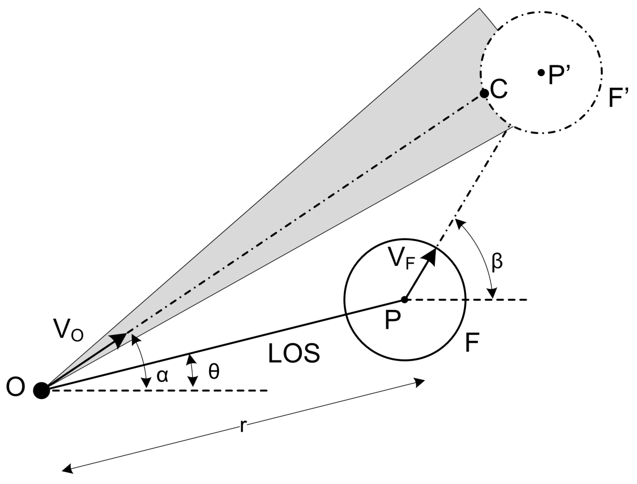


Figure 3.5: Our vessel starts at O, and has a constant velocity vector $\boldsymbol{V_O}$. The other vessel starts at P, and has a constant velocity vector $\boldsymbol{V_F}$. The radius of the circle F is set to the combined safety radiuses of both vessels. C will be the point where the safety radiuses of the vessels intersect. The grey collision cone includes the values of α that will result in an intersection.

$$\begin{aligned} V_r &= \dot{r} = \|V_F\|\cos(\beta-\theta) - \|V_O\|\cos(\alpha-\theta) && (3.14) \\ V_\theta &= r\dot{\theta} = \|V_F\|\sin(\beta-\theta) - \|V_O\|\sin(\alpha-\theta), && (3.15) \end{aligned}$$

where r is the distance between the centers of the vessels, and R is the combined safety radius of the vessels, the vessels are on a collision course if

$$\begin{aligned} r^2V_\theta^2 &\leq R^2(V_r^2+V_\theta^2) && (3.16) \\ V_r &< 0. && (3.17) \end{aligned}$$

The second criterion says that to be on a collision course, the distance between the vessels must be decreasing. We thus have $\dot{r} = V_r < 0$. The first criterion is a bit more complex, but by introducing $V = \sqrt{V_r^2 + V_\theta^2}$ it can be simplified to:

$$\underbrace{\frac{r}{|V|}}_{T_{pass}} \leq \underbrace{\frac{R}{|V_\theta|}}_{T_{evade}} . \qquad (3.18)$$

V is actually the speed of vessel P relative to the controlled vessel O, and the left side of the inequality thus represents the time until the two vessels pass each other. The right side of the equation is the time required for the vessels to move out of each other's way, and if the vessels pass *before* they have done this, resulting in $T_{pass} < T_{evade}$, they will collide.

Situation determination

When the control system has determined that the vessel is on collision course with another vessel, the next step is to determine a course of action. COLREGS defines a set of situations and corresponding actions, so the algorithm needs to be able to distinguish between the different situations in a reliable manner. If the heading of the approaching vessel is known, this can easily be done by calculating α_h, the angle of the controlled vessel relative to the approaching vessel. If the coordinates and heading of the approaching vessel are given by $(x_o,\, y_o)$ and ψ_o, and the controlled vessel is located at $(x,\, y)$, α_h is given by:

$$\alpha_h = \operatorname{atan2}(y - y_o,\ x - x_o) - \psi_o \tag{3.19}$$

According to regulation 13b, a vessel is overtaking another vessel when it is *coming up with another vessel from a direction more than 22.5 degrees abaft her beam.* This gives a sector of 130° centered at the rear of the other vessel labeled *overtaking.* The size of the head-on sector is more difficult to determine. A sector of 30° was used in [7, 13] though, and this choice seems reasonable.

One may wonder why the head-on sector should be that much smaller than the overtaking sector. If the head-on sector was large, that would cause our vessel going to great lengths to pass the approaching vessel on the correct side. This would often include our vessel crossing directly in front of the other vessel, creating potentially dangerous situations. When approaching at larger angles, the case should be treated as crossing situations, making our vessel pass behind the other vessel and not care about whether it passes on the starboard or port side.

The sectors are shown in Figure 3.6. When the controlled vessel is approaching another vessel, the type of situation can be determined from which sector it is approaching from.

Penalty

The final step is to enforce the rules. Both the RRT and Dynamic Window algorithms are driven by a scalar cost/utility function. This function will now be modified to add a penalty whenever the controlled vessel violates any of the rules. As both algorithms rely on calculating the cost/utility for a number of routes before a choice is made, routes that break the rules get a low score, and will be avoided by the algorithms as long as better options are available.

Whenever the controlled vehicle is on collision course with another vehicle, the event is registered and tracked until it is resolved. The way the penalty is calculated depends on the type of situation. In a head-on situation the controlled vehicle should pass with the approaching vehicle on its left. Passing on the other side results in a penalty. The penalty will thus be applied if the following relationship holds:

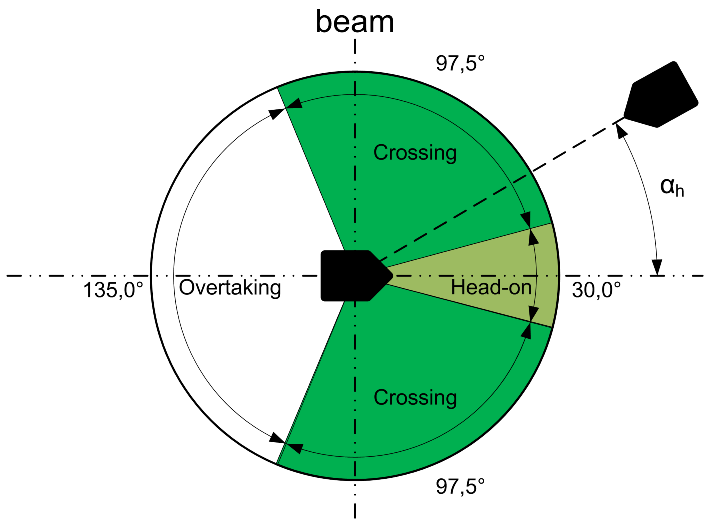


Figure 3.6: Which COLREGS situation our vessel is in can be determined from which sector it is approaching from. The approaching vessel is displayed inside the circle, and our vessel is outside. In this case, our vessel is in the *crossing* sector, and should act accordingly.

$$-\frac{\pi}{2} \leq \alpha_h \leq \alpha_{hpo}, \tag{3.20}$$

where α_h is the angle between the vessels as shown in Figure 3.6. The constant $\alpha_{hpo} \geq 0$ defines how far the penalty sector is extended. As shown in Figure 3.7, selecting a nonzero value for α_{hpo} makes the sector in front of the vessel a penalty sector. This extended sector forces the controlled vessel out of the way of the approaching vessel early, and clearly signals the intent of passing on the correct side.

The situation is over when the two vessels pass each other, or the distance between them becomes larger than a preset constant. Whether the vessels have passed each other can be identified by the following relationship:

$$|\alpha_h| > \frac{\pi}{2}. \tag{3.21}$$

When in a crossing situation, a more elaborate method is required to determine when to apply a penalty. As shown in Figure 3.8, the penalty is determined by which vessel reaches the crossing point first. Given $x_\Delta = x_2 - x_1$ and $y_\Delta = y_2 - y_1$, the times of crossing are calculated as:

$$t_1 = \frac{y_\Delta \cos\psi_2 - x_\Delta \sin\psi_2}{(\cos\psi_2 \sin\psi_1 - \sin\psi_2 \cos\psi_1) \cdot u_1} \tag{3.22}$$

$$t_2 = \frac{y_\Delta \cos\psi_1 - x_\Delta \sin\psi_1}{(\cos\psi_2 \sin\psi_1 - \sin\psi_2 \cos\psi_1) \cdot u_2}, \tag{3.23}$$

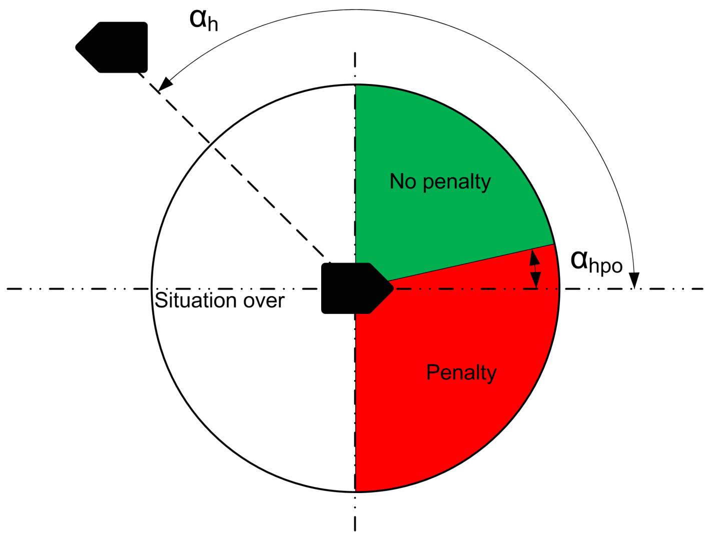


Figure 3.7: When in a head-on situation; if the line between the controlled vessel, shown in the upper left, and the other vessel lies over the penalty sector, a penalty should be applied. For the vessels in the figure, the situation is over.

where ψ_1 and ψ_2 are the headings of the respective vessels, and u_1 and u_2 are their surge speeds. Sideslip can be taken into account by exchanging u_1 and u_2 with the velocities of the vessels, and ψ_1 and ψ_2 with the course angles of the vessels.

If the crossing time of the give-way vessel is *less* than the crossing time of the other vessel, the give-way vessel crosses in front of the other vessel, and a penalty should be applied. If the crossing time is larger than that of the other vessel, the give-way vessel is crossing behind as it should, and no penalty is applied.

In addition to the overlapping penalty-sector for the head-on situation, a more general penalty is applied whenever the controlled vessel is on a collision course with another vessel. These two features encourage the controlled vessel to move out of any dangerous situations, providing the *significant* course alteration requested by COLREGS.

3.1.5 Utility function

The utility function of the Dynamic Window algorithm is responsible for the behaviour of the controlled vehicle. The desired behaviour is ranked as follows, with avoiding collisions being the most important:

1. Avoid collisions
2. Keep safe distance from obstacles

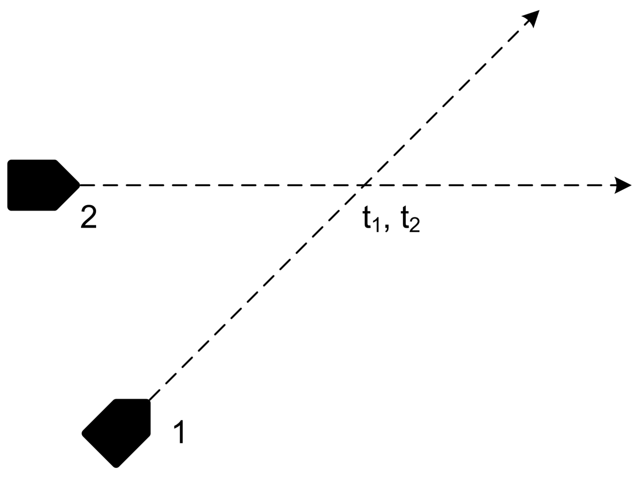


Figure 3.8: When in a crossing situation; the give-way vessel should pass the crossing point after the other vessel. If vessel 1 is to give-way, its time of crossing t_1 should be larger than the other vessel's time of crossing t_2.

3. Follow COLREGS rules
4. Make good progress towards the goal

Collision avoidance and progress towards the goal is more or less provided by the original algorithm, while terms for COLREGS compliance and keeping safe distance will be added to the utility function.

Scaling

The performance of the Dynamic Window algorithm is very dependent on the scaling of the different components in the objective function. With the original objective function (2.29), the distance component had to be given a high priority to prevent the vehicle from colliding. This gave problems when choices leading the vessel away from the goal had a high $\text{dist}(u, r)$ 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* are all scaled to give a value $\in [0, 1]$. For the first two functions, *heading* and *velocity*, a linear scaling is applied. The *dist* function requires a bit more elaborate approach though.

Curved paths and paths with low speeds cover less distance than straight paths and paths with high speeds. The distance of a path therefore has 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.

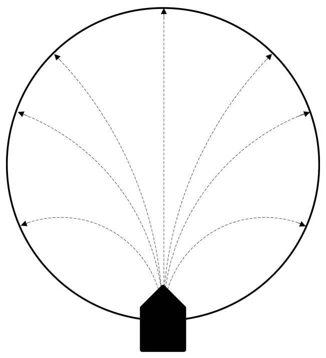


Figure 3.9: Maximum distance is limited by the length of the arcs within the black circle.

To achieve this feature, a circle is defined intersecting the vessel with a center in the vessels forward direction, see Figure 3.9. The maximum distance of travel for a choice (u_i, r_i) is limited by the length of the constant-speed arc defined in Section 2.2.1 that fits within this circle. The distances travelled for each choice (u_i, r_i) is thus limited and scaled by this distance. This scheme is also applied when scaling paths with negative speeds, but then the circle is flipped to the other side of the vessel.

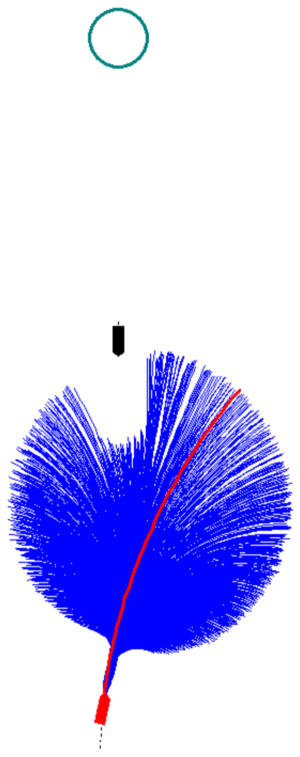
The speed function of the Dynamic Window algorithm is also modified to make the algorithm try to follow a commanded velocity u_{sp}. Before scaling, the speed function is given as:

$$\text{speed}(u, r) = -|u_{sp} - u|. \tag{3.24}$$

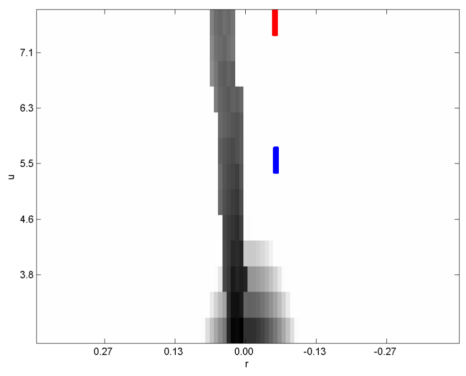
Additional Terms

In addition to scaling the original terms in the objective function of the Dynamic Window algorithm, extra terms have to be added. The first term comes from the COLREGS implementation. This is calculated as a penalty, so before it can be added to the objective function, it has to be negated and scaled.

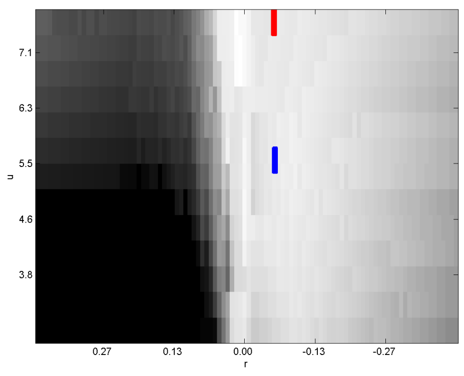
The second term to be added is the term that penalizes paths going into *unsafe* areas. Both of these terms accumulate over time, and gets larger the longer the vessel



(a) Vessel and *dynamic* obstacle

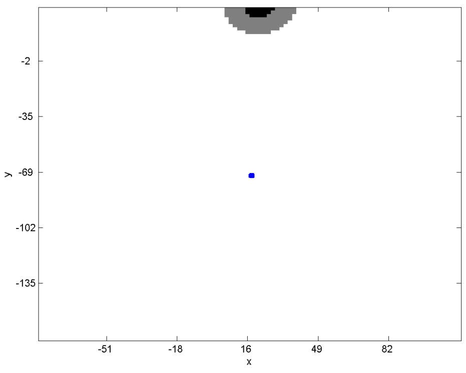


(b) Distance map

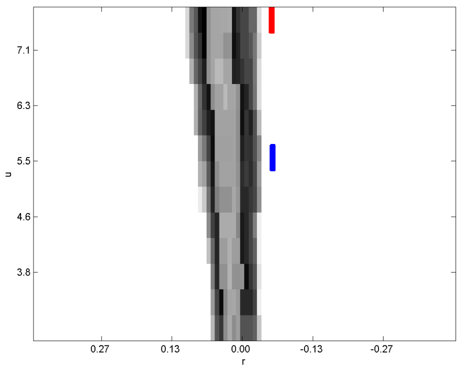


(c) COLREGS map

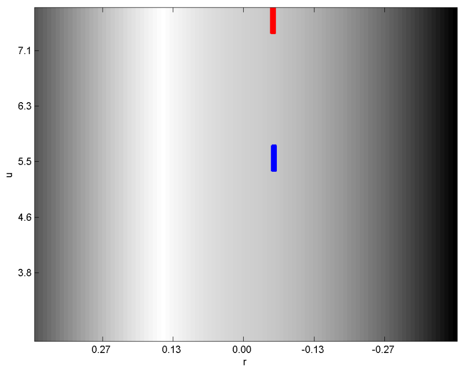
Figure 3.10: In (a) the controlled vessel is approaching another vessel from below. The blue lines show the predicted trajectories for different velocity choices. The motion of the other vessel is predicted, and limits the distance that can be travelled along some of the trajectories before crashing. This is a situation where COLREGS rules must be followed for correct behaviour. The COLREGS map shown in (c) clearly suggests passing on the right side of the other vessel. The distance map is shown in (b). Brighter is better. The blue rectangle in (b) and (c) marks the current velocity of the vehicle, while the commanded velocity is marked by the red rectangle. The red line in (a) shows the chosen trajectory.



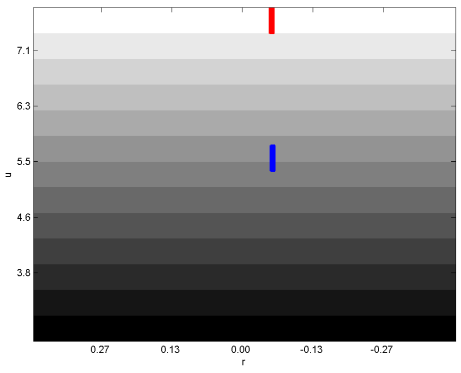
(d) Environment map



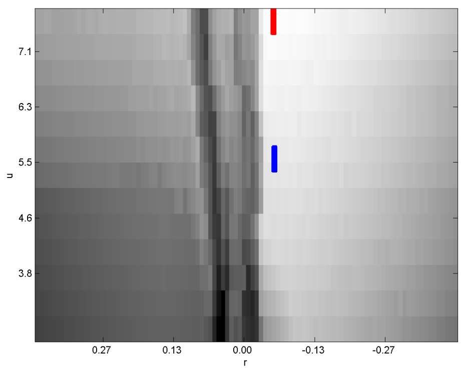
(e) Unsafe map



(f) Heading map



(g) Speed map



(h) Final dynamic window map

Figure 3.10: The map of the environment is shown in (d). It is centered around the current position of the vessel, shown in red. The gray area in the environment map is unsafe, while the black is forbidden. The unsafe-map shown in (e) weighs down trajectories through the unsafe areas. The heading map in (f) and the speed map in (g) try to lead the vessel towards the target at a reasonable velocity. The final dynamic window is shown in (h).

is in violation.

The final objective function thus becomes:

$$G(u,r) = \sigma\left(\alpha \cdot \mathrm{h}_s(u,r) + \beta \cdot \mathrm{d}_s(u,r) + \gamma \cdot \mathrm{v}_s(u,r) + \zeta \cdot \mathrm{c}_s(u,r) + \upsilon \cdot \mathrm{s}_s(u,r)\right), \quad (3.25)$$

where h_s is the scaled *heading* function, d_s is the scaled *distance* function, v_s is the scaled *speed* function, c_s is the scaled *colregs* function and s_s is the scaled *unsafe* function. Figure h shows a vessel controlled by the improved Dynamic Window algorithm.

Filtering

Both the original Dynamic Window algorithm and [38] suggest that some kind of filter should be used on the objective function of the algorithm. A filter may be necessary if the sensor data used in the algorithm is noisy, but this was never a problem during the work on the thesis. A couple of filter configurations were implemented and tested, but as they all slowed down the response of the algorithm to sudden changes in the environment, the filter was abandoned.

When working with noisy and uncertain sensor data, preprocessing such as filtering should be applied to the data before it is given to the Dynamic Window algorithm.

3.2 Global collision avoidance

The global part of the algorithm is responsible for long-term planning. As seen in [38] this is necessary for avoiding local minima in the environment. It also has the potential of improving the performance of the overall system since it takes a larger area into account when planning a path than the local method.

The main component of the global collision avoidance system is the RRT algorithm. It is implemented as described in Section 2.2.2 with the extensions described in this section. The A* algorithm is introduced as a guide for the RRT algorithm, trying to force it along the shortest route through the environment.

The RRT algorithm is, like the Dynamic Window algorithm, implemented in the three-dimensional C-space with prediction as explained in Sections 3.1.2 and 3.1.3. Prediction can be used 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. The flexibility of the algorithm also allows us to implement COLREGS compliance in the same way as for the Dynamic Window algorithm.

3.2.1 Milestone generator

At the heart of the RRT algorithm is the *random state generator*. Represented by the function CANDIDATE_MILESTONE() in Section 2.2.2, it is responsible for generating candidate milestones, or states, which the local planner of the RRT algorithm then tries to reach. In this implementation, the states calculated for each milestone are $[x_c, y_c, u_c]^T$, i.e., the coordinates of the vehicle and its surge speed. When the corresponding states of the vehicle get close enough to the states of the milestone, the milestone is considered reached. The rest of the vehicle states are allowed to have arbitrary values at the milestone.

By limiting the number of milestone states, the number of dimensions in the search-space of the algorithm is reduced from 6, for all the states of the 3-DOF model, to 3. This size-reduction of the search-space provides a significant increase in the performance of the algorithm.

The Halton sequence

The distribution of the candidate milestones, and especially their spatial coordinates, can have a serious impact on the performance of the algorithm. It has been claimed that the distribution given by the *Halton* sequence gives better results than ordinary randomized and biased distributions [47]. The reason for this is probably that the near-uniform distribution allows the algorithm to quickly explore the entire search space before creating major branches impeding path optimality.

The basis for the milestone generator is the Halton sequence. 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 [57]. As seen in Figure 3.11, coordinates created with the Halton sequence are more uniformly distributed than the ordinary randomized coordinates.

The surge speeds at the milestones are selected a bit differently than the spatial coordinates. They are first of all biased towards velocities to promote faster routes through the environment. Secondly, they are chosen from a predefined set of discrete values, to avoid the unnecessary throttling one would experience when using analog velocity values.

A* assistance

When trying to find a route through an environment, the search space for the RRT algorithm can often be very large. Using only the Halton generator, equal attention is paid to all parts of the search space. This is good because it allows the algorithm to find alternative and difficult routes through the environment, but it also has a downside. As the allowed computation time is limited, the uniform distribution often leads to less

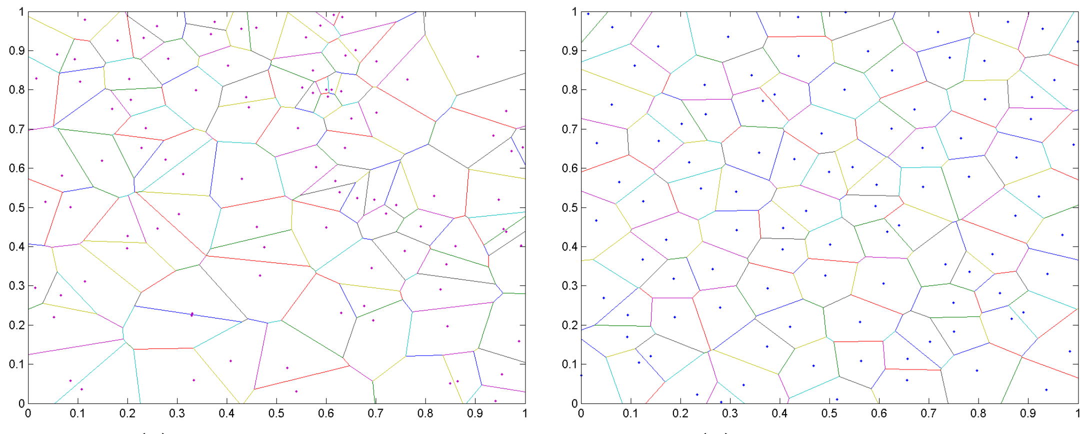

(a) Random distribution **(b)** Halton distribution

Figure 3.11: The figure shows a set of coordinates generated by the Matlab *twister* pseudorandom number generator (a), and by the Halton sequence (b). The Voronoi diagrams[1] for the points are drawn to better visualize the distribution of the points. As seen in the figures, the area of the regions in (b) are more even than those in (a), clearly suggesting that the coordinates generated by the Halton sequence are more uniformly distributed than those generated by the ordinary randomized method in Matlab.

optimal paths since the attention of the algorithm is scattered over the entire search space.

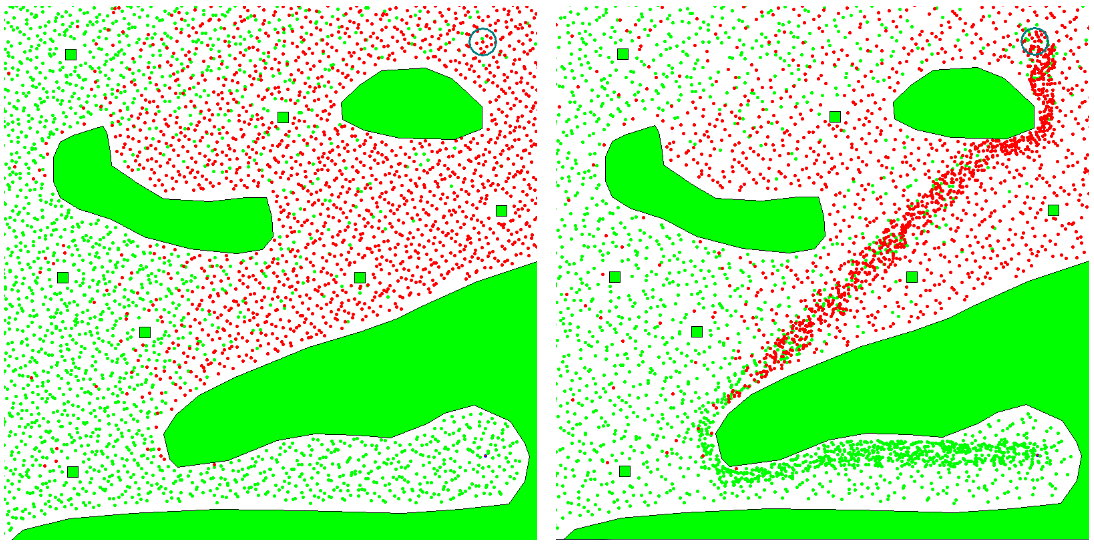
What if we could direct the attention of the algorithm towards the area most likely to hold the most optimal path? The algorithm would then spend more time in this area, and perhaps come up with a better solution to the path planning problem.

As seen in [38], the A* algorithm consistently found more optimal paths through the environment than the RRT algorithm. The idea here is that the optimal path through the environment probably lies around the route generated by the A* algorithm. We need to direct the attention of the RRT algorithm towards this route, and this can easily be accomplished by increasing the density of candidate milestones along the A* route.

Before each run of the RRT algorithm, the A* algorithm is run to find the A* path through the environment. This generates a set of coordinates $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_N$ where $\mathbf{a} \in \mathcal{R}^2$. A milestone taken from the A* route is now generated as follows:

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \mathbf{a}_q + R \cdot \mathbf{s}, \tag{3.26}$$

[1]For a set of points S in a plane, the Voronoi diagram is a partition of the plane associating a region $V(p)$ with each point $p \in S$. The regions are set up so that all points in a region $V(p)$ are closer to p than any other point in S [60].



(a) Without A* assistance **(b)** With A* assistance

Figure 3.12: This figure shows the distribution of candidate milestones when A* assistance is disabled (a) and when it is enabled (b). The generator parameters are chosen as $f = 0.1$ and $R = 50m$, which should result in about 10% of the nodes being placed by the A* generator. The density of milestones is clearly higher along the A* path. The green milestones are the ones reached in the given iteration. The red ones were not reached.

where $q \in \{1, 2, \cdots, N\}$ and $\mathbf{s} \in \{\mathbf{x} \in \mathcal{R}^2 \mid \|\mathbf{x}\|_1 \le R\}$ are randomly chosen. The surge speed at the milestone is chosen purely by using the Halton generator.

This strategy will produce candidate milestones in close vicinity to the A* optimal path, but we also want the rest of the search space to be explored. Every time a candidate milestone is requested by the algorithm, the CANDIDATE_MILESTONE() function has to determine whether to respond with a candidate generated by the Halton sequence or one chosen along the A* route. In this implementation, the choice is done by chance:

$$generator = \begin{cases} Halton, & if\ \mathcal{X} > f \\ A*, & if\ \mathcal{X} \le f, \end{cases} \tag{3.27}$$

where $\mathcal{X} \in [0, 1\rangle$ is randomly chosen. This means that an A* milestone is generated with a probability of f. If for instance $f = 0.1$, an A* milestone will be generated 10% of the time on average. Care has to be taken when choosing a value for f, as too high a value will prohibit exploration of the rest of the search space, possibly resulting in no path being found. Choosing f too small however, reduces the effect of the A* assistance to the algorithm. An example is shown in Figure 3.12.

Feedback and continuity

During the execution of the RRT algorithm, the Halton and A* generators are responsible for creating candidate milestones. The first milestones are special however, and are selected quite differently.

When the algorithm starts, the state of the controlled vessel is $\mathbf{x}_{init}$. The first milestone is given this state, and is thus placed at the current position of the vessel. As the vessel already is at this position, the milestone must be feasible, and it can be added without checking for intersections with the environment.

The first milestone will be the *root* of the tree generated by the RRT algorithm. All other milestones connect to the root either directly or indirectly. The root milestone is what connects the RRT algorithm to the controlled vessel and provides feedback into the algorithm.

The purpose of the next milestones is to try to connect the new path with the old one. Because the execution of the RRT algorithm takes a considerable amount of time, the state of the vessel when a path is complete will not be the same as it was when the calculation started. If the algorithm can make the paths coincide at the point the vessel will be when the new path is put into use, the transition between the two paths will likely be very smooth.

The controlled vessel will probably follow its current path relatively accurately while the RRT algorithm is creating a new path. If the calculation time of the RRT algorithm is fixed to T_{RRT} seconds, the vessel will therefore be located at a time $2T_{RRT}$ along the previous path when the new path is put into use. It will likely be at a time T_{RRT} along the new path when it is put into use.

One solution to making the paths coincide is to use the first $2T_{RRT}$ seconds of the old path as the start of the new path. Doing so would eliminate the feedback to the algorithm however, so another approach is needed. The solution chosen for this implementation is very simple and does not require any modifications to the main algorithm. The first N_h candidate milestones are simply selected from the interval $[0,\, 2T_{RRT}]$ of the old path at constant intervals. This gives the algorithm an opportunity to connect with the old path.

The first part of the RRT will now reconnect with the old path if possible. To make the rest of the tree build off of the reconnected branch, causing the final solution to coincide with the old path, the cost of travelling to these first milestones are reduced by multiplication with a factor K_h.

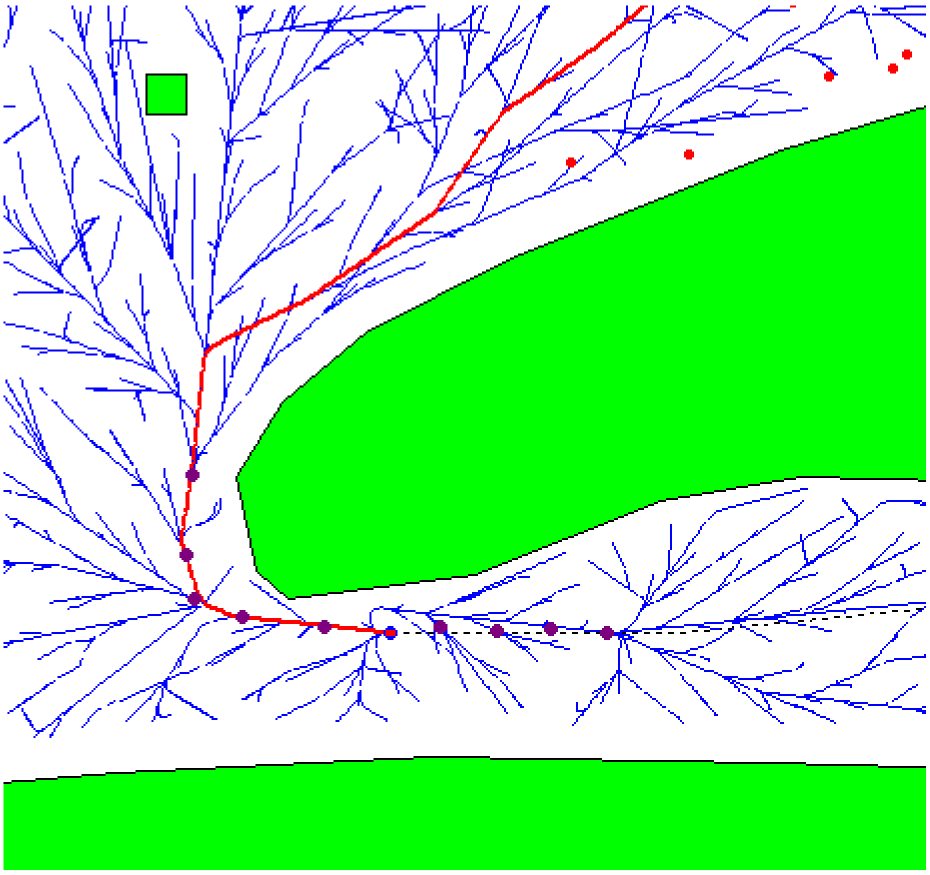


Figure 3.13: The dotted line coming from the lower right shows the path of the vessel up till this point. The purple dots are the candidate waypoints generated from the old path to provide continuity. They successfully guide the new path, which is shown in red, along the route chosen by the previous path, providing a smooth transition between the two paths.

3.2.2 Cost functions

The RRT algorithm uses two cost functions (or metrics). The first is used to determine the cost of travelling along a given path, and the second provides an estimate of the cost of connecting a path to a new milestone.

Cost of path

The cost of travelling along a path is very context-dependent. Fuel efficiency, speed, or even stealth may be the priority, and in most cases, a suitable cost function can be synthesized to create a path with the given properties. In this case there are three priorities:

1. Keep safe distance from obstacles

2. Follow COLREGS rules
3. Maximize path speed and minimize path length

The first two priorities are the same as for the utility function of the local algorithm in Section 3.1.5. We want to keep a safe distance from obstacles if possible and follow the rules of the road. The additional element is added to make the generated paths as fast and short as possible. This is accomplished by weighting a combination of the time used and the distance travelled. Collision avoidance and progress towards the goal are not taken into account as these appear other places in the algorithm.

The cost of a path is given by the following expression:

$$\mathrm{COST}(\mathbf{x}) = \left[\gamma\frac{1}{u_w} + (1-\gamma)\frac{1}{\bar{u}}\right] \cdot \mathrm{DIST}(\mathbf{x}) + \zeta \cdot \mathrm{COLREGS}(\mathbf{x}) + \upsilon \cdot \mathrm{SAFE}(\mathbf{x}), \tag{3.28}$$

where $\mathrm{DIST}(\mathbf{x})$ is the travelled distance to $\mathbf{x}$, $\mathrm{COLREGS}(\mathbf{x})$ is the COLREGS penalty accumulated along the path, and $\mathrm{SAFE}(\mathbf{x})$ is the accumulated penalty for travelling through unsafe areas. The multiplier for the distance parameter tries to balance keeping the path length down while also keeping travelling time to a minimum. The value of $\gamma \in [0, 1]$ determines the weight on path length. Choosing $\gamma = 1$ causes the algorithm to find the shortest path disregarding speed along the path, while setting $\gamma = 0$ results in the path using the least amount of time. $\bar{u}$ is the average velocity of the vessel, and u_w is a constant scaling factor for distance to make it compatible with time.

Cost to connect

The cost to connect is an estimate of the COST() function at a new milestone through a given path. It is used by the NEAREST_NEIGHBOUR() function in the RRT algorithm to figure out which node to try to connect to a new milestone. If this function gives a wrong estimate, we can end up trying to connect new milestones to the wrong node, ending up with a very suboptimal path through the environment.

The candidate milestone consists of the three states $\mathbf{m}_c = [x_c, y_c, u_c]^T$, while the states of the vehicle of a given node is $\mathbf{x} = [x\ y\ \psi\ u\ v\ r]^T$. The cost estimate tries to make sure the selected node is close to the candidate milestone. It also puts a penalty on nodes where the vehicle is facing away from the candidate milestone in an effort to reduce the amount of turning required by the vessel:

$$\mathrm{COST}_E(\mathbf{x}, \mathbf{m}_c) = \mathrm{COST}(\mathbf{x}) + \alpha\left[\gamma\frac{1}{u_w} + (1-\gamma)\frac{1}{\bar{u}}\right]\sqrt{x_\Delta^2 + y_\Delta^2} + \beta\cdot(\psi - \mathrm{atan2}\,(y_\Delta,\ x_\Delta)), \tag{3.29}$$

where $x_\Delta = x_c - x$ and $y_\Delta = y_c - y$. The value of β in the above equation determines how *bendy* the resulting path will be. A high value gives a straighter path, but may make it harder for the algorithm to find a feasible path to the goal.

The function atan2 $(x,\ y)$ computes the *arctangent* of y/x in much the same way as the original tan^{-1}. The difference is that because atan2 takes x and y as separate arguments, it is able to give a result in the range $\langle -\pi,\ \pi]$, as opposed to tan^{-1} which gives a result in the range $\langle -\pi/2,\ \pi/2]$. When calculating the angle of a vector, atan2 thus gives the true angle where the original tan^{-1} can be wrong by a value of π.

3.2.3 The local controller

The local controller used by CAN_CONNECT($\mathbf{x}$, $\mathbf{m}_c$) was designed to be as simple as possible. When trying to connect a state $\mathbf{x}$ with a candidate milestone $\mathbf{m}_c$, the controller simply steers the heading of the vehicle towards the milestone $\mathbf{m}_c$. The yaw setpoint for the vessel is simply calculated as:

$$\psi_{sp} = \text{atan2}\left(y_c - y,\ x_c - x\right), \tag{3.30}$$

where $\mathbf{m}_c = [x_c, y_c, u_c]^T$ is the candidate milestone, and $\mathbf{x} = [x\ y\ \psi\ u\ v\ r]^T$ is the vessel state vector. The surge speed setpoint is set to the surge speed of the target milestone:

$$u_{sp} = u_c. \tag{3.31}$$

3.2.4 Medial nodes

The RRT tree expands by selecting candidate milestones and trying to steer the simulated vessel towards them. This process is expensive, especially for large search spaces where the vessel must be integrated over great distances to reach new candidate milestones. In order to populate the search-space faster, medial nodes are introduced.

Every time a new milestone is reached, a number of medial nodes are added on the path between the parent node and the new milestone. In [22], a single node was added at a random position on every path. In this implementation however, the number of nodes added depend on the distance travelled between the parent node and the new milestone, positioning N_m nodes per meter. The nodes are evenly distributed on the path.

Adding medial nodes reduces the time needed to populate the search-space, and has the potential of generating straighter paths with less overshoot as new candidate milestones can connect directly with the medial nodes.

3.2.5 When the algorithm fails

As previously mentioned, the RRT algorithm is not complete, and may fail in finding a path even though one exists. With no path that leads to the goal, the question is what to do.

In the case of failure, an attempt is made to figure out which of the explored paths are best to follow even though they don't lead to the goal. This is done by calculating a score for each of the paths. The score emphasizes two properties of the paths: how close to a goal they lead the vessel, and how costly they are. Given a goal $\mathbf{g} = (x_g, y_g)$, the score function is given as:

$$s(\mathbf{x}, \mathbf{g}) = \sqrt{(x_g - x)^2 + (y_g - y)^2} + \omega \cdot \text{COST}(\mathbf{x}) \tag{3.32}$$

The path with the lowest score is considered the best choice.

3.3 Glued together

The local and the global algorithms of the controller have now been described. What remains is how to join these to create the final hybrid controller.

3.3.1 Path tracker

As shown in Figure 2.1, the global algorithm provides input to the local algorithm. Here we run into our first problem. The output of the RRT algorithm is a path, while the Dynamic Window algorithm wants a heading to travel in. This is solved by adding an intermediate component which takes a path from the RRT algorithm and tracks it, providing the Dynamic Window algorithm with *heading suggestions* as the vessel travels along the path. This concept is visualized in Figure 3.14.

It may be easiest to think of the system as the path tracker generating heading commands for the vessel, and the Dynamic Window controller modifying the generated commands if necessary to avoid collisions and dangerous situations. In open waters with no other objects in the vicinity, the commands of the path tracker will be in near-direct control of the vessel.

The path tracker makes sure that the path from the global algorithm is followed. The paths generated in this system consist of waypoints, each described by its spatial coordinates and the surge speed which the vessel should keep while approaching the waypoint. A basic algorithm for following a path can be decomposed into two different parts, namely *line following* and *waypoint switching.*

Line following

Line following is when the vessel is travelling between two waypoints, following the line between them. See Figure 3.15 for an illustration. One of the simplest ways to accomplish this is to steer the vessel directly towards the second waypoint. Given two

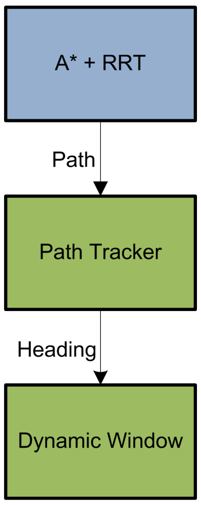


Figure 3.14: The path tracker generates the heading commands required to make the vessel follow the path given by the RRT algorithm. It is here shown as a separate component.

waypoints $\mathbf{w}_k = (x_k, y_k)$ and $\mathbf{w}_{k+1} = (x_{k+1}, y_{k+1})$, where the vessel is travelling from $\mathbf{w}_k$ to $\mathbf{w}_{k+1}$, the heading-command for the vessel with coordinates $\mathbf{p}(t) = (x(t), y(t))$ is then calculated as follows:

$$\psi_{sp}(t) = \operatorname{atan2}\left(y_{k+1} - y(t),\ x_{k+1} - x(t)\right), \tag{3.33}$$

which is the same as the local controller for the RRT algorithm, for which it is good enough because external noise such as wind and sea currents are not taken into account in its internal world model. In the real world however, the controller may result in the vessel deviating from the line between the two waypoints.

In some applications it is desired to follow the path as closely as possible. The lateral deviation from the path, also known as the *cross-track error* must then be minimized. The cross-track error is calculated as [10]:

$$e(t) = (y(t) - y_k)\cos\alpha_k - (x(t) - x_k)\sin\alpha_k, \tag{3.34}$$

where $\alpha_k = \operatorname{atan2}\left(y_{k+1} - y_k,\ x_{k+1} - x_k\right)$ is the orientation of the line segment between the two waypoints.

One way of reducing the cross-track error is to use the *Line-of-sight* (LOS) algorithm for generating heading commands [20]. Instead of steering the vessel towards the next waypoint, the vessel is now steered towards a point on the line between the two waypoints. The point is located at a fixed distance L from the vessel, i.e., at the one of the intersections between a circle with radius L centered at the vessel and the line. When travelling from $\mathbf{w}_k$ to $\mathbf{w}_{k+1}$, the intersection closest to $\mathbf{w}_{k+1}$ must be chosen to make

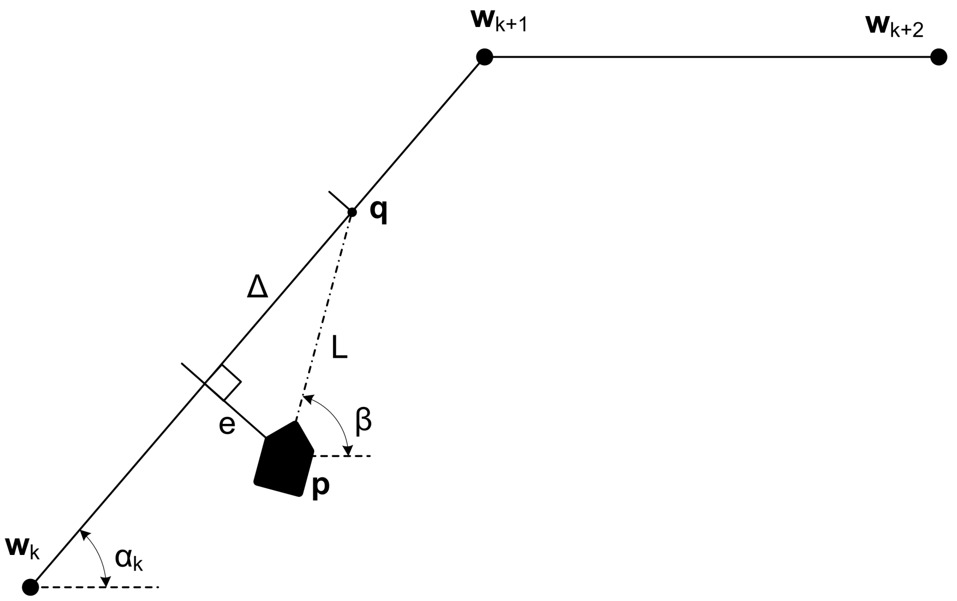


Figure 3.15: This figure shows the most important variables to take into account when trying to make a vessel follow the line between two waypoints. Here, e is the cross-track error, i.e., the shortest distance from the vessel to the line. The primary goal is to minimize the cross-track error to the line currently being followed. Also, $\mathbf{p}$ is the position of the vessel, while $\mathbf{q}$ is the point which the vessel velocity (ideally) aims at in an attempt to minimize e.

sure the vessel travels in the right direction.

A slightly modified version of the LOS algorithm, which also works with a crosstrack error $e > L$, is presented here. It gives the following steering command:

$$\psi_{sp} = \alpha_k - \operatorname{sgn}(e(t)) \sin^{-1} \left(\min(\frac{|e(t)|}{L}, 1) \right). \tag{3.35}$$

This method is effective at reducing the cross-track error. If the error gets larger than L, the vessel will actually steer directly towards the line in an effort to get back on track. A variation of the algorithm, aims to hold Δ, shown in Figure 3.15, constant instead of L. This is achieved by applying the following steering command to the vessel [10]:

$$\psi_{sp} = \alpha_k + \operatorname{atan2}\left(-e(t),\ \Delta\right). \tag{3.36}$$

This scheme is a bit more gentle than LOS when the vessel deviates from the line, as it never tries to steer directly towards the line. It always approaches the line at an angle. Consequently, this is the algorithm used in the developed control system.

While these approaches are good, and manage to significantly reduce the cross-track error, they cannot completely eliminate it. Imagine a constant wind blowing perpendicular to the line that the vessel is trying to follow. The methods reduce the cross-track error to a manageable value, but will not completely eliminate it. The problem is that while the vessel is travelling along the line, it generates no force to independently oppose the wind force, and is pushed away from the line.

One way to solve this problem is to use (3.35) or (3.36) to steer the actual velocity of the vessel instead of its heading. Lower-level controllers will then be responsible for making the vessel travel in the requested direction. By controlling the course of the vessel, convergence will be achieved even in the presence of environmental disturbances, given that they are within the capabilities of the vessel and control system.

Waypoint switching

With line following taken care of, the next issue is what to do when the vessel gets to the end of the line at the upcoming waypoint. The answer is simple: switch to the next waypoint and start following the line between the new waypoint and the previous target. In practice the procedure is a bit more involved however. How do we for instance know when the vessel has reached the end of the line?

One way is to define a *Circle of Acceptance* [20] around the target waypoint as shown in Figure 3.16. The circle is given a radius R_a, and when the vessel gets within the circle, the controller switches to the next waypoint. The switching criterion thus becomes:

$$(x(t) - x_{k+1})^2 + (y(t) - y_{k+1})^2 \leq R_a^2. \tag{3.37}$$

This approach has the advantage that the vessel starts turning towards the next waypoint before actually getting to the target waypoint, which can reduce overshoots of the path.

The *Acceptance Circle* method makes sure the vessel visits every waypoint along a given path. For some applications this may be desirable, but in others the waypoints are only there to make up a path, and the goal is to follow the path, not visit the waypoints. If the vessel misses a waypoint when under the Acceptance Circle method, it will turn back in an attempt to reach the missed waypoint. When following the path is most important, this behaviour is very undesirable.

Given the measures in Figure 3.17, the switching criterion in (3.37) can be rewritten as:

$$s^2 + e^2 \leq R_a^2, \tag{3.38}$$

where s is the distance to the next waypoint projected onto the current line. As seen in the equation, the cross-track error is a part of the switching criterion. Cross-

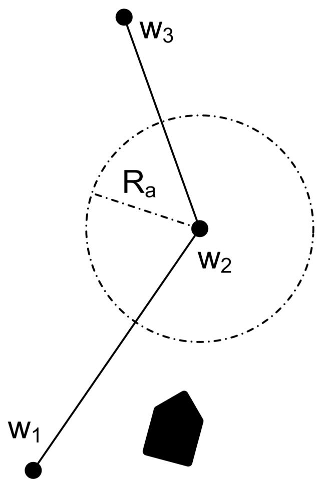


Figure 3.16: When using the *Acceptance Circle* method, a circle of acceptance is defined around the next waypoint, in this case $\mathbf{w}_2$. When the vessel gets to the circle, the subsequent waypoint, i.e., $\mathbf{w}_3$, takes over as the next waypoint.

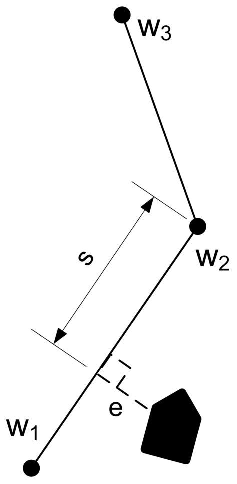


Figure 3.17: In this waypoint switching scheme, the switching occurs when the distance to the next waypoint along the path, i.e., s, becomes small enough.

track error is handled by the line following algorithm, and should (because of the issues discussed above) not be a part of the switching criterion. By removing the cross-track error, the new criterion becomes:

$$s \leq R_a. \tag{3.39}$$

When using this new criterion, waypoint switching will be based solely on the progress of the vessel along the path. The distance to the next waypoint along the current line is calculated as:

$$s(t) = (x(t) - x_k)\cos\alpha_k - (y(t) - y_k)\sin\alpha_k, \tag{3.40}$$

where R_a determines how early the switching should occur. This is the switching criterion used in the developed controller.

3.3.2 Threads

The final issue of joining the global and local controllers together still remains. The local algorithm is fast and executes in a fraction of a second. This allows it to be run in real-time, giving good control bandwidth. The global algorithm on the other hand, is pretty slow. The quality of the generated path is also heavily dependent on the time the algorithm is allowed to run.

When implementing the control algorithms on a computer these properties present a challenge. The local algorithm must be run in real-time and the global algorithm must be allowed enough time to get a good result. In many control applications, the entire controller is implemented in a single thread, executing the control algorithms from the upper layer and down at each iteration. This approach is not possible here since the long running-time of the global algorithm would slow down the entire controller, and neutralize the advantage of having a high-bandwidth local algorithm.

The solution is to implement the local and global algorithms in different threads, allowing them to run in parallel. This scheme allows the local algorithm to run at a consistently high frequency. For every time the global algorithm runs, the local algorithm runs N times, as seen in Figure 3.18. When the global algorithm is finished, the new path is passed to the local algorithm, which follows this path until the global algorithm is done with another path.

When a new path has been delivered to the local algorithm, a search through the waypoints of the path is done to find the waypoint closest to the vessel. The path tracker then starts tracking the path at this waypoint, to ensure a smooth transition to the new path.

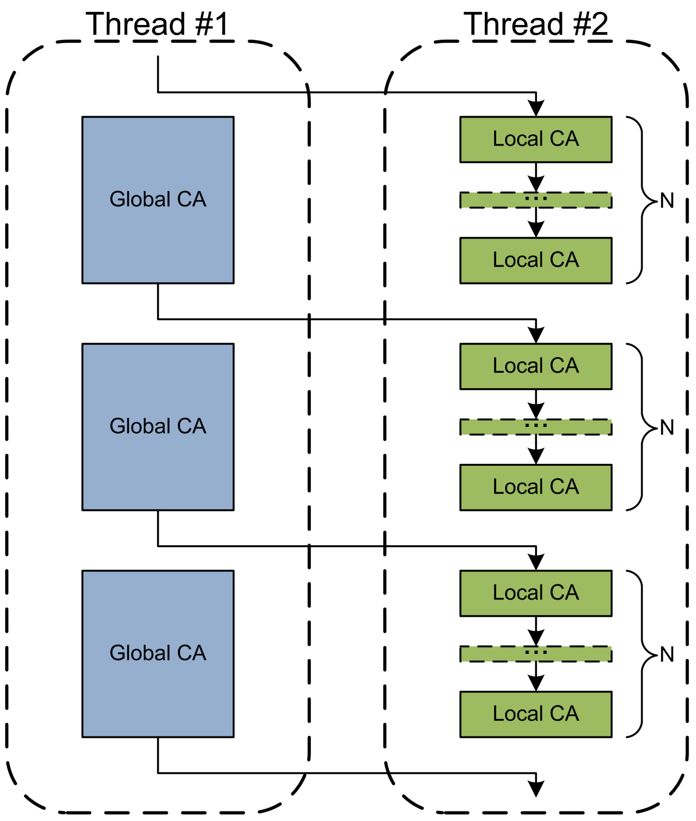


Figure 3.18: The global and local controllers each have their own thread of execution. This scheme allows the global algorithm to run in the background, calculating a new path, while the local controller runs at a constant high frequency giving good control of the vessel. The arrows show when control data is passed between the threads.

Chapter 4

Simulation results

In [38], several simulations were performed in Matlab to quantitatively determine the properties of the presented collision avoidance algorithms. Simulation is an important tool as it allows for repeatable testing under controlled conditions. Factors of uncertainty like noise, component failure, etc. are entirely under control in the simulator, putting full focus on the properties of the controllers.

The simulations presented in this chapter are meant to highlight the additional features and changes in the developed controller, as well as show of some of its strengths. The vessel model used in the simulations use the parameters from Table 5.5. Parameters for the low-level controllers of the vessel are given in Table 4.1.

Parameter	Value	Description
$K_{p,u}$	0.1	Proportional gain for speed controller
$K_{p,r}$	5	Proportional gain for yaw rate controller
$K_{p,\psi}$	5	Proportional gain for heading controller
$T_{d,\psi}$	1	Derivative time for heading controller

Table 4.1: Parameters for the low-level controllers of the vessel defined in Section 2.1.4. The parameters were selected by trial-and-error, and provide relatively good performance.

The performance of the collision avoidance algorithms is very dependent on tuning. It is not difficult to find parameters that give acceptable performance, but they can definitely be improved by further tuning. The parameters used for the Dynamic Window algorithm in the following simulations are given in Table 4.2 and the parameters for the RRT algorithm are given in Table 4.3.

The RRT algorithm is given a constant amount of time to find the best path through the environment, which means that path quality depends heavily on the amount of processing power available to the algorithm. Specifications for the machine running the

Parameter	**Value**	**Description**
α	1	Heading importance
β	3	Distance importance
γ	1	Speed importance
ζ	2.5	COLREGS importance
υ	3	Safety importance
M	13	Number of translational velocities in window
N	100	Number of angular velocities in window
R	50 m	Radius of circle limiting trajectories
$T_{a,r}$	5	Number of yaw rate choices in DW
$T_{a,u}$	1	Number of surge speed choices in DW
R_a	40 m	Path tracker switch distance
Δ	60 m	Path tracker look-ahead
Map-res	0.5 $m/cell$	Map resolution
Pred-horz	30 s	Prediction horizon
T_r	0.5 s	Timestep for heading-map
Col-cone penalty	0	Penalty for being on a collision course
Head-on penalty	1	Penalty for doing wrong in a head-on situation
Crossing penalty	1	Penalty for doing wrong in a crossing situation

Table 4.2: Parameters for the Dynamic Window algorithm. As seen in the table, distance, COLREGS and safety are given the highest importance to avoid collisions and dangerous situations. Goal achievement is secondary to keeping the vessel safe. As for COLREGS, the penalty for being on a collision course with another vessel had to be removed as it prevented the vessel from making manoeuvres where it temporarily entered the collision cone. The look-ahead distance Δ was chosen larger than the switch distance R_a to give a smooth convergence to the path while still tracking a line segment close to the vessel.

simulations are given in Table 4.4.

Since the vessel in the simulations is unaffected by waves, currents and other sources of randomness, simulations only including the local Dynamic Window controller will have the same outcome every time. The RRT controller however, introduces randomness by itself, so simulations featuring this controller are run a number of times to show the range of paths developed by the controller.

As well as showing the path of the controlled vessel, most of the simulation figures in this chapter display key values from the simulation. The reported values are *energy usage* (E), *distance travelled* (S), *time taken* (T_w) and *computation time* (T_c). For figures containing multiple simulations, the reported values are an average of the results from all the simulations.

Parameter	**Value**	**Description**
α	2	Distance penalty in NEAREST_NEIGHBOUR()
β	20	Angular penalty in NEAREST_NEIGHBOUR()
γ	0.5	Weight on time versus distance
u_w	10	Distance scaling factor
ζ	10	COLREGS cost
υ	50	Safety cost
ω	0.1	Cost multiplier when algorithm fails
f	0.1	Chance of waypoint being from A*
R	50	Spread of A* waypoints from A* path
N_h	10	Number of nodes for continuity
K_h	0.25	Reduces cost for continued path
T_{RRT}	40 s	Allowed execution time
N_m	0.025 nodes / m	Medial node density
Map-radius	400 m / 2000 m	Radius of world map
Map-res	0.5 / 0.1	Map resolution
Pred-horz	500 s	Prediction horizon
Col-cone penalty	0.1	Penalty for being on a collision course
Head-on penalty	1	Penalty for doing wrong in a head-on situation
Crossing penalty	1	Penalty for doing wrong in a crossing situation

Table 4.3: Parameters for the RRT algorithm. The RRT algorithm runs in cycles of 40 seconds. This enables it to find higher quality paths through the environment, at the cost of a long update time.

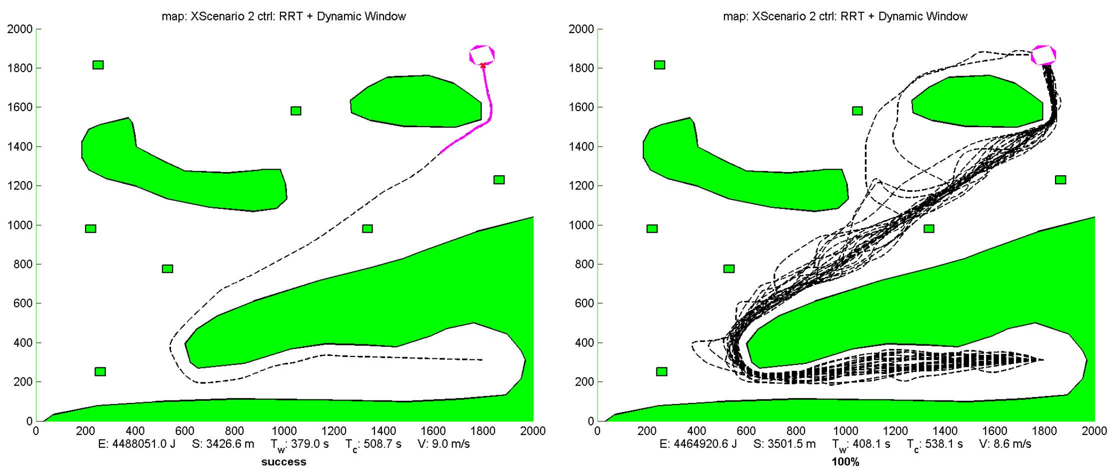
Unit	**Type**
Processor	Intel Pentium 4 @ 3.00 GHz
Memory	2048MB DDR2
Operating System	Windows XP Pro, SP2
Matlab Version	R2007a

Table 4.4: Specifications of the computer which the simulations were run on.

4.1 Basic collision avoidance

The first scenario has already been used for a couple of visualizations earlier in the thesis, and shows the capabilities of the collision avoidance system in a static environment using the full control system. The environment is relatively large at $2km \cdot 2km$, and so is the search-area of the RRT algorithm. The vessel starts out in the lower right, and the goal is in the upper right of the map.

The first simulations, shown in Figure 4.1, use the full controller with all features enabled. The local, reactive part of the collision avoidance algorithm is relatively unnecessary in this scenario, as the environment is completely static. In cases where the vessel is not able to do exact path tracking, it has some use however, as it avoids obstacles when the vessel is off-path.



(a) Single simulation **(b)** Multiple simulations

Figure 4.1: This scenario shows basic collision avoidance by the entire hybrid controller. Here (a) shows a typical run through the map, while (b) shows 25 different runs through the map.

As seen in the figure, the controller has no problem navigating the static environment. Most of the paths taken by the controlled vehicle are relatively optimal, taking the shortest route from the start to the goal. Some of the paths are less optimal though, but this is expected. Unless given infinite processing power, the RRT will always provide suboptimal paths.

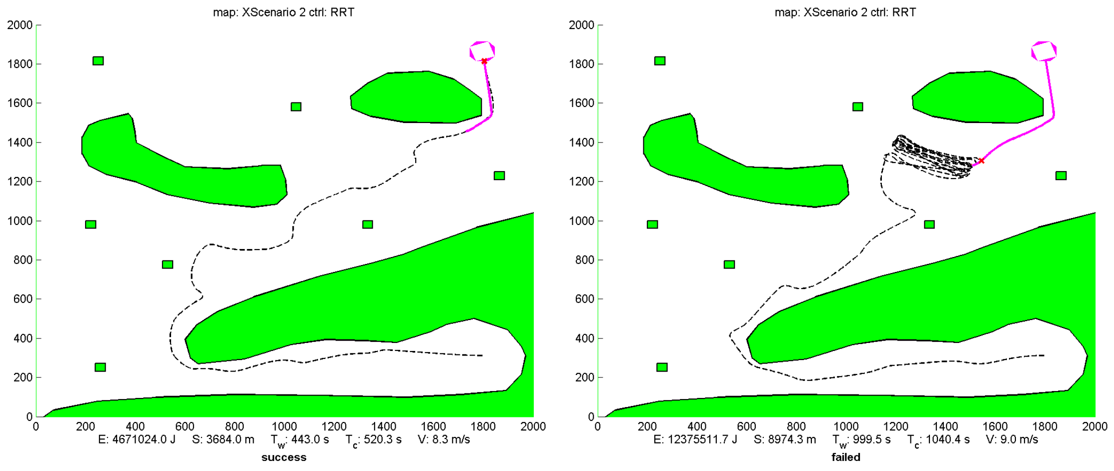
4.2 The RRT modifications

The RRT algorithm has been extended in a number of ways in this thesis. The extensions were discussed in Section 3.2, but their effect on the performance of the RRT algorithm has not yet been determined. To be able to determine their usefulness, the scenario from the previous section has been run a number of times for different configurations of the controller. The Dynamic Window algorithm was not included in the simulations so full focus could be put on the performance of the RRT algorithm.

As some of the extensions were meant for increasing the efficiency of the RRT algorithm, simulations were also run with a harder constraint on computation time. In these simulations, the RRT algorithm was only allowed a tenth of the normal computation time, i.e., 4 seconds for each iteration. By doing this, the suitability of the RRT algorithm for less powerful computing platforms could also be determined. The results are given in tables 4.5 and 4.6.

4.2.1 Continuity

Continuity was discussed in Section 3.2.1. The idea is to make a path generated by the RRT algorithm coincide with the beginning of the previous path, avoiding discontinuity in the path tracked by the vessel when a new path is put into use. As seen in Figure 4.2, where this extension has been disabled, its absence is devastating for the performance of the system.



(a) Poor performance **(b)** Limit cycle

Figure 4.2: As seen in (a), the absence of the continuity extension may result in the vessel spending a great deal of time travelling from an old RRT path to a new one. The worst case scenario is shown in (b) where the path-switching results in a minimum that the vessel is unable to get out of.

As well as improving the performance of the controller, striving for coinciding paths also reduces the possibility of limit cycles caused by the path-switching. As seen in Table 4.5, 4 of the 25 simulations performed resulted in the vessel ending up in a local minimum, preventing it from reaching the goal.

4.2.2 A* assistance

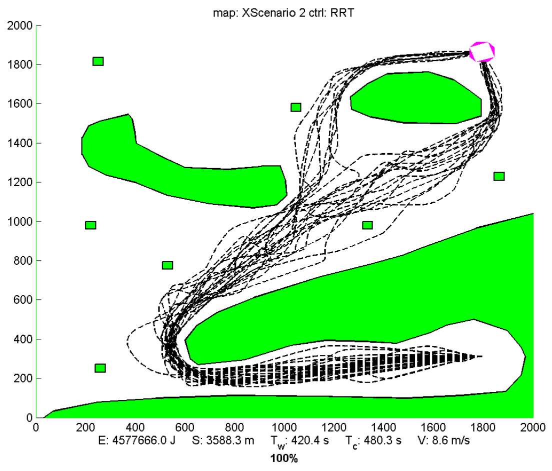
The purpose of A* assistance is to concentrate the attention of the RRT algorithm towards places in the environment that are most likely to contain the shortest path. This extension was discussed in Section 3.2.1, and as shown in Figure 4.3 it has a very positive effect on the performance of the algorithm.

4.2.3 Medial nodes

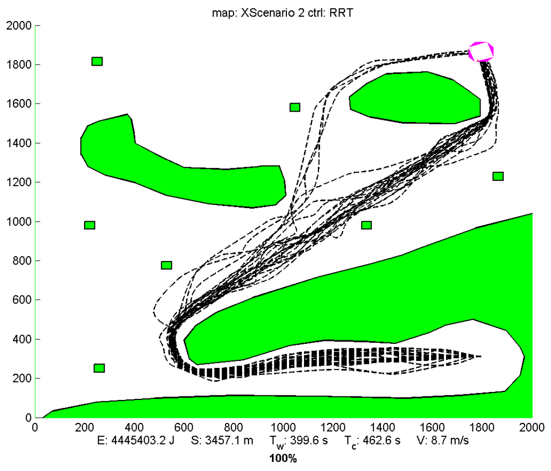
Adding medial nodes had relatively little positive effect on performance. As seen in Table 4.5, not placing medial nodes even results in a small performance gain. The results from the simulations with limited computation time in Table 4.6 are a bit mixed however. Looking at distance travelled, the controller with the regular settings for medial nodes comes out on top of the controllers with no and less medial nodes per meter.

The reason for the poor results probably lies in the NEAREST_NEIGHBOUR() function in the RRT algorithm. The more nodes there are in the RRT tree, the longer this function takes to execute, probably becoming a bottleneck in the algorithm. The value of a medial node is lower than a regular milestone in terms of exploring the environment. For the medial nodes to be useful, the number of medial nodes added must be such that they don't negate their own value by slowing the further execution of the algorithm, hindering additional milestones to be placed on the map.

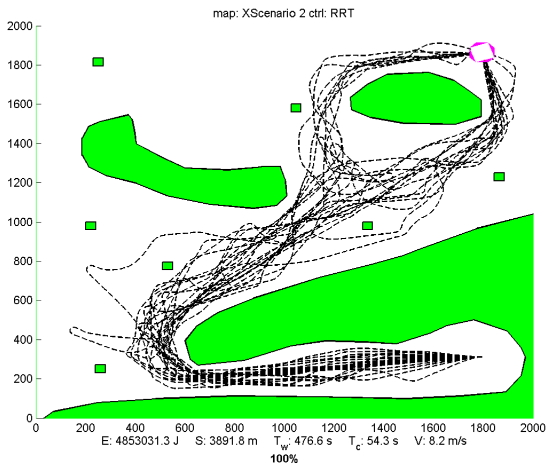
Whether or not this extension can be made useful is probably a question of correct tuning. By optimizing the NEAREST_NEIGHBOUR() function, the negative effect of the medial nodes can probably also be reduced, possibly giving a better result.



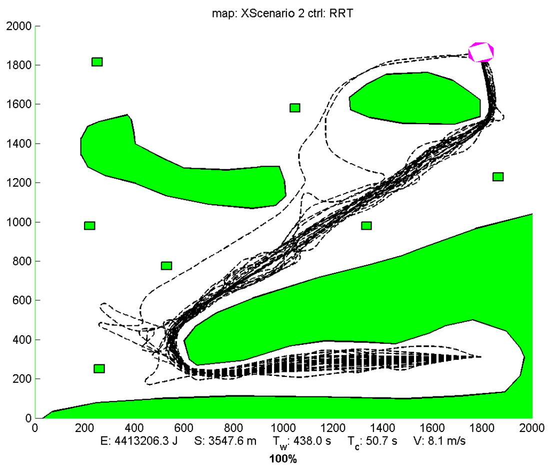
(a) No A*



(b) 10% A*



(c) Less CPU, no A*



(d) Less CPU, 50% A*

Figure 4.3: Figures (a) and (b) shows the RRT algorithm without and with A* assistance. The latter clearly has a higher concentration of paths around the optimum. This property is even more clear in figures (c) and (d), where less processing time was available to the algorithm. In (d), 50% of the milestones were placed by the A* generator, resulting in very good paths with very little processing power.

	Success	Distance	Avg. Vel.	Energy	Time	CPU Time
RRT w/all	100%	3457.0 m	8.7 m/s	**4.45** MJ	399.6 s	462.6 s
RRT w/o A* $f = 0$	100%	3588.3 m	8.6 m/s	4.58 MJ	420.4 s	480.3 s
RRT w/o medial $N_m = 0$	100%	**3453.0** m	**9.0** m/s	4.53 MJ	**384.9** s	445.0 s
RRT w/o continuity $N_h = 0$	84%	3604.1 m	8.7 m/s	4.67 MJ	415.6 s	478.4 s

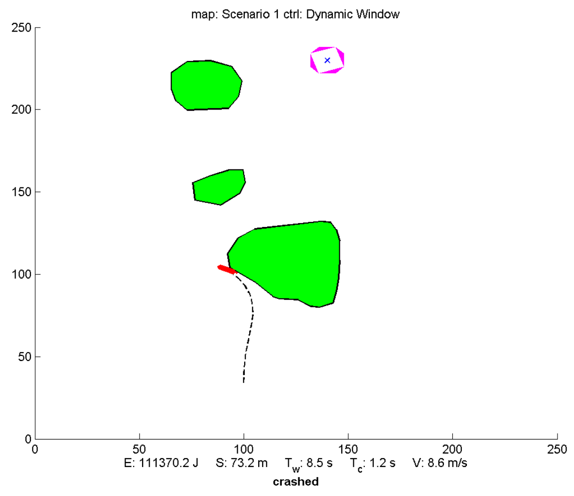
Table 4.5: This table shows the results of running the RRT algorithm with and without its modifications. The scenario was run 25 times for each controller, and the results averaged. The RRT algorithm had 40 seconds of calculation time for each iteration. The average CPU times are higher than the time required to travel through the environment because the first RRT iteration is not included in the world time. The last iteration is also finished even though the vessel finishes the scenario halfway through the computation. As seen in the results, the highest performance is actually gained by not placing any medial nodes. It is also worth to mention that only the runs where the vessel got to the goal have been included in the averaged results.

	Success	Distance	Avg. Vel.	Energy	Time	CPU Time
RRT w/all	100%	3647.8 m	8.0 m/s	4.47 MJ	457.8 s	52.4 s
RRT w/o A* $f = 0$	100%	3891.8 m	8.2 m/s	4.85 MJ	476.6 s	54.3 s
RRT w/50% A* $f = 0.5$	100%	**3547.6** m	8.1 m/s	**4.41** MJ	438.0 s	50.7 s
RRT w/o medial $N_m = 0$	100%	3692.5 m	**8.6** m/s	4.73 MJ	**429.0** s	49.5 s
RRT w/less medial $N_m = 0.01$	100%	3659.2 m	8.0 m/s	4.52 MJ	456.5 s	52.2 s
RRT w/more medial $N_m = 0.05$	100%	3778.5 m	7.8 m/s	4.60 MJ	484.1 s	55.0 s
RRT w/o continuity $N_h = 0$	64%	3786.9 m	8.3 m/s	4.83 MJ	459.5 s	52.1 s

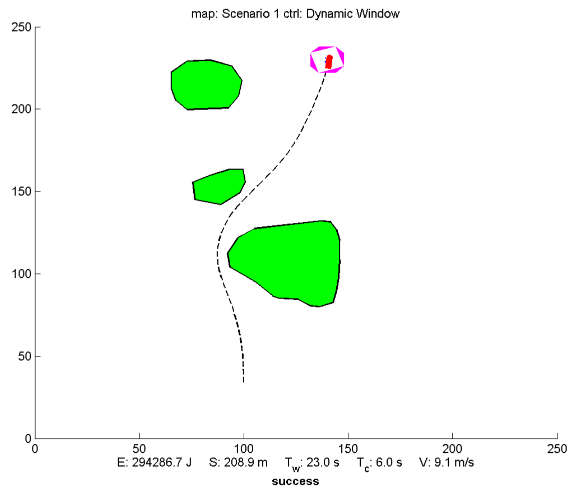
Table 4.6: In this table, results are shown from simulations with and without the RRT modifications when less computation time was available to the algorithm. The results are consistent with what was found for the case with no restrictions on computation time, except for the medial nodes extension. The results here are a bit more diffuse, but the possible gains are nowhere near what is gained by the A* and continuity extensions. Results are also shown for a simulation where 50% of the nodes are placed by the A* algorithm, compared to the normal 10%. Here this addition results in superior performance relative to the other configurations.

4.3 The Dynamic Window modifications

The original Dynamic Window algorithm does not take accelerations and lateral speeds into account. As shown in Figure 4.4, this can lead to gross mispredictions of the future behaviour of the controlled vessel. The obstacles in the scenario are given a safety-distance of only 2 metres, so the controllers cannot rely on the safety-zone to save them from mispredictions.



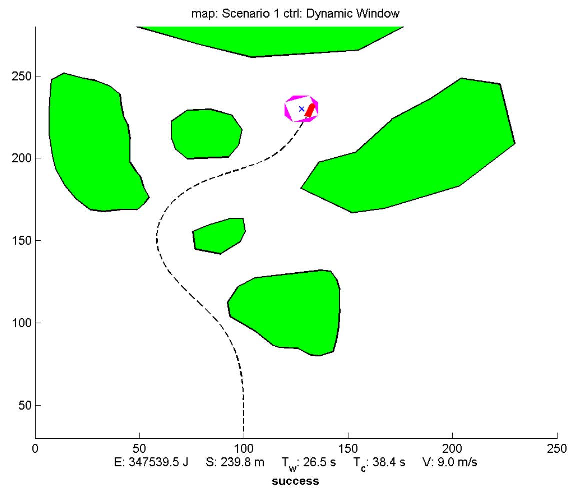
(a) Regular Dynamic Window



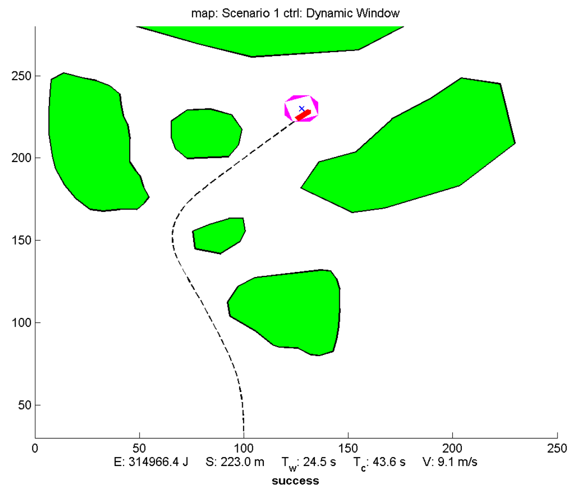
(b) Improved Dynamic Window

Figure 4.4: The original Dynamic Window algorithm is unable to detect the sideways motion of the vessel, causing it to crash into the first obstacle as shown in (a). The improved Dynamic Window algorithm in (b) predicts the sideways motion of the vessel and steers well clear of the obstacles.

In scenario shown in Figure 4.5, the obstacles have been given a larger safety-distance of 5 metres. Both the controllers are now able to guide the vessel to the goal, but as seen in the figure the path generated by the improved Dynamic Window algorithm is a lot smoother than what the original algorithm is capable of.



(a) Regular Dynamic Window



(b) Improved Dynamic Window

Figure 4.5: As in the previous scenario, the original Dynamic Window algorithm is unable to see the sideways motion of the vessel, and is thus not able to pre-compensate for it. Being able to predict this motion, the improved Dynamic Window algorithm compensates for it earlier, resulting at a straighter and safer path.

4.4 COLREGS-compliant collision avoidance

Misunderstandings at sea can be costly, so COLREGS compliance is important for any autonomous vessel operating alongside human-operated vessels. It also improves predictability when dealing with other autonomous vessels. Some COLREGS rules have been incorporated in both the global and local collision avoidance algorithms, providing both long-term COLREGS-compliant path planning and dynamic COLREGS-compliant collision avoidance.

The scenarios in this section show situations where the correct behaviour is determined by COLREGS rules. The initial states of the vessels in the scenarios are set up in such a way that a non-compliant system would choose wrongly. Following the rules costs something to the system.

The global and local parts of the algorithm are run separately to show the ability of both layers to follow *the rules of the road*. As can be seen in the following sections, the system follows the implemented rules correctly.

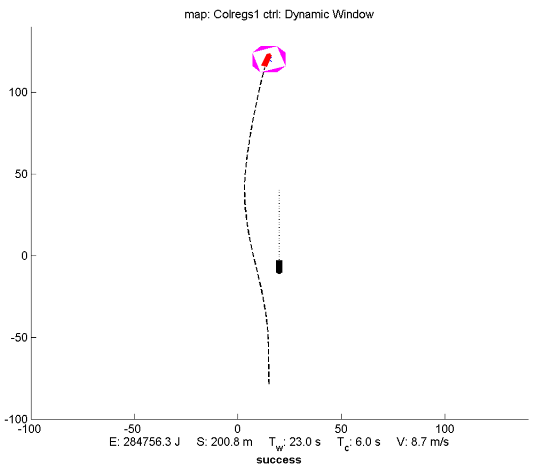
4.4.1 Head-on

When in a Head-on situation, the controlled vessel should pass with the other vessel on its port side. In the following case, it means passing to the right of the other vessel. The controlled vessel approaches the goal from the bottom of the map, while the other vessel travels downward. As seen in figures 4.6 and 4.7, the controlled vessel starts out a little to the left, making a non-compliant manoeuvre easier than a compliant.

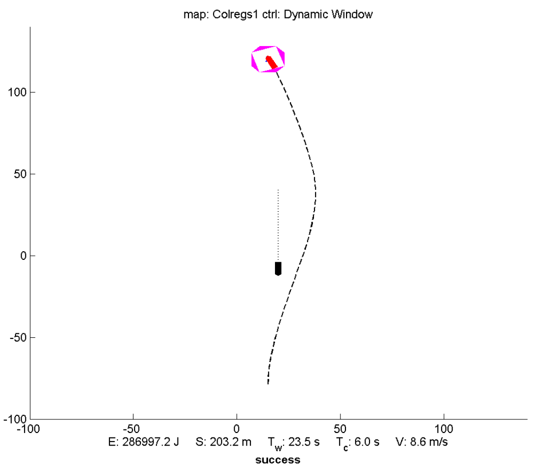
4.4.2 Crossing from left

When in a crossing situation, the correct action depends on the context of the situation. In this case we will assume that the controlled vessel is the give-way vessel, and has to cross behind the other vessel. The results are shown in figures 4.8 and 4.9.

In a crossing situation where the controlled vessel has to give way, it basically has two choices. It can either slow down and let the other vessel pass before continuing, or it can continue at the same velocity and steer around the back of the other vessel. The choice selected by the algorithms depends on the situation and the algorithm parameters. For the Dynamic Window algorithm, the importance of *heading* over *speed* is the dominant factor. If the target heading points directly towards the crossing vessel and it is more important to keep the heading, the vessel will slow down. Else it will keep its speed and travel around. Choices in-between are of course also possible, where the vessel slows down a bit, but still steers around the crossing vessel.

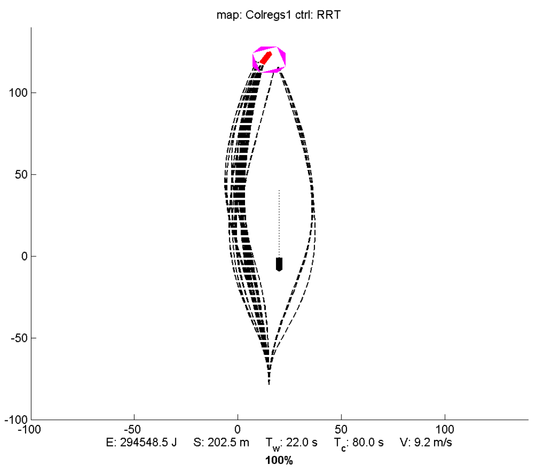


(a) Without COLREGS

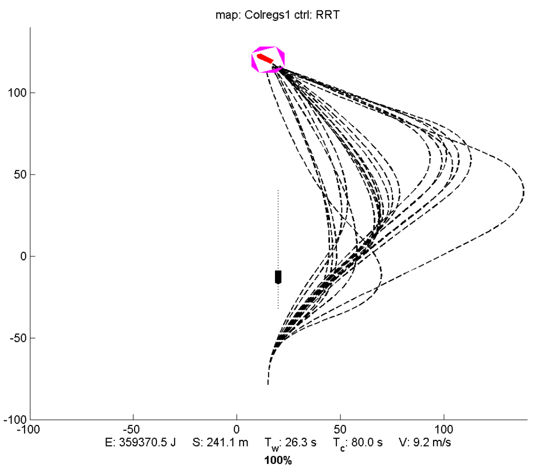


(b) With COLREGS

Figure 4.6: When COLREGS is disabled, the Dynamic Window algorithm chooses the easier non-compliant route as expected. Enabling COLREGS gives the algorithm a large penalty for violating COLREGS, making the compliant route the best option.

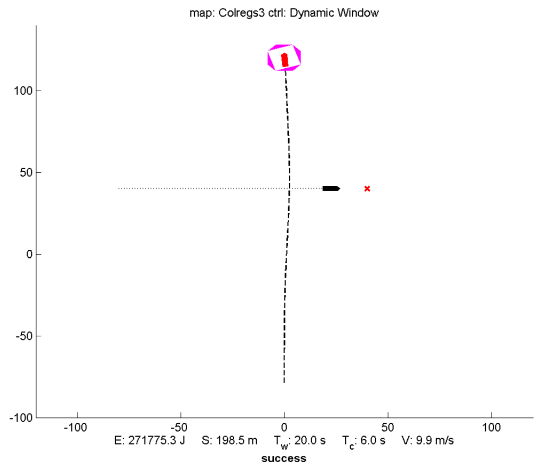


(a) Without COLREGS

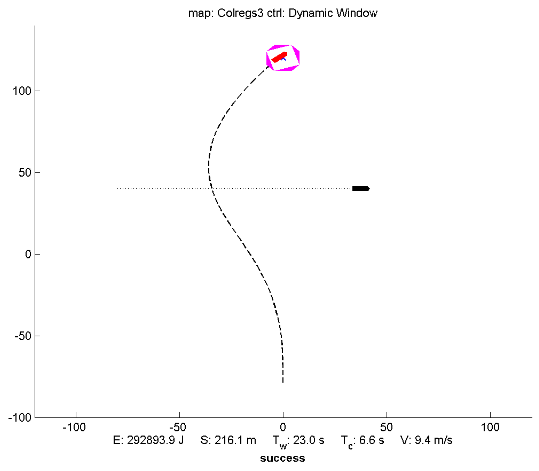


(b) With COLREGS

Figure 4.7: With COLREGS disabled, RRT is free to pass on any side of the approaching vessel, as seen from the traces in (a). In (b), COLREGS is enabled and RRT passes on the correct side.

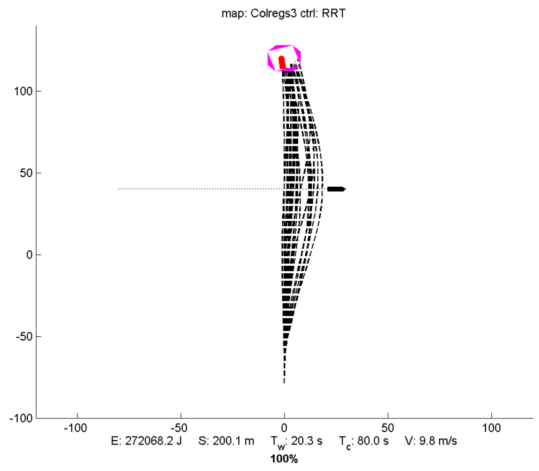


(a) Without COLREGS

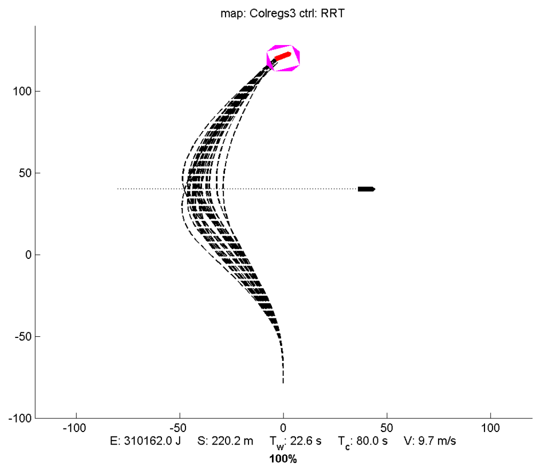


(b) With COLREGS

Figure 4.8: In (a), the controlled vessel travels at maximum speed, and passes in front of the other vessel. The result is manoeuvre that is both dangerous and non-compliant. With COLREGS enabled, the Dynamic Window algorithm performs correctly, passing behind the other vessel as seen in (b).



(a) Without COLREGS

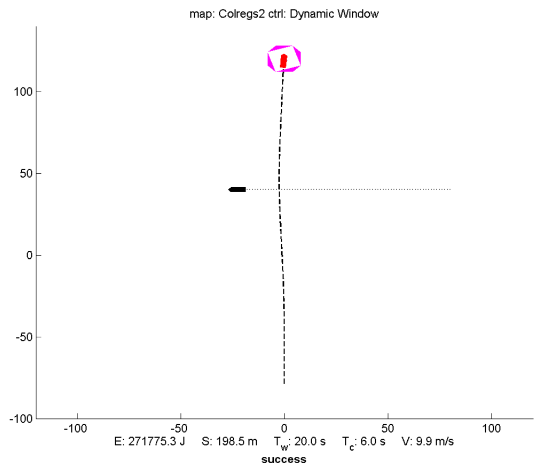


(b) With COLREGS

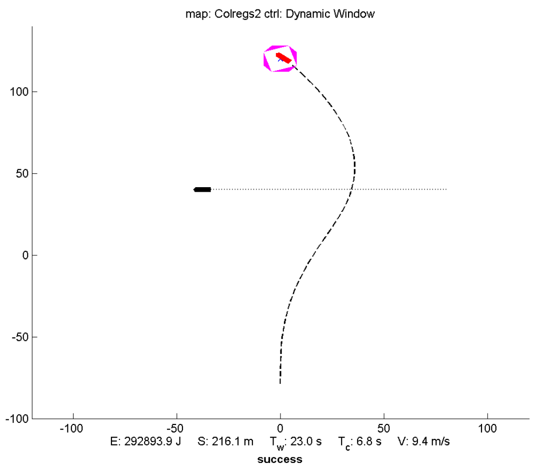
Figure 4.9: The RRT algorithm also passes behind the other vessel when COLREGS is enabled. Passing in front of the vessel is the fastest option though, and without COLREGS the RRT algorithm consistently passes in front of the other vessel.

4.4.3 Crossing from right

The situation where the other vessel is crossing from the right is included for completeness. The controlled vessel still has to pass behind the crossing vessel. The results are shown in figures 4.10 and 4.11.

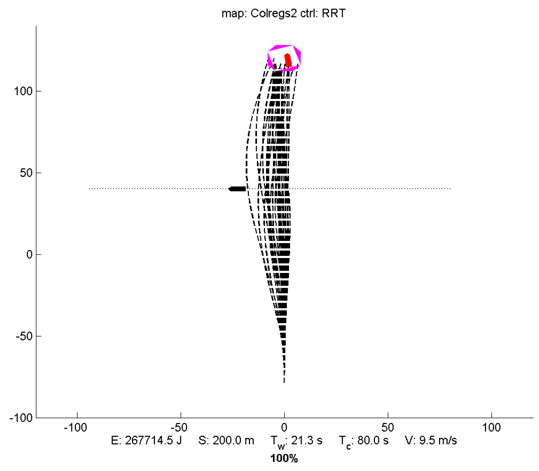


(a) Without COLREGS

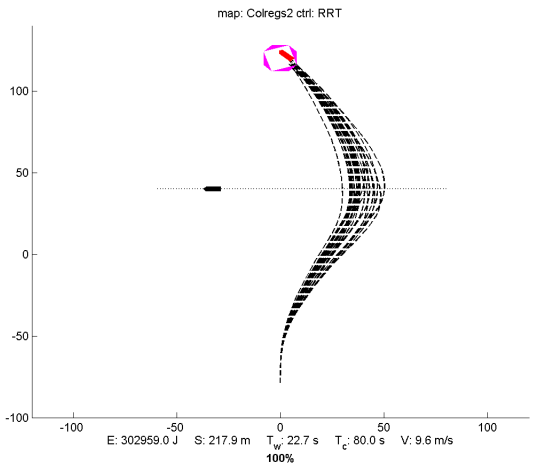


(b) With COLREGS

Figure 4.10: The Dynamic Window performs as expected, passing on the correct side with COLREGS enabled.



(a) Without COLREGS



(b) With COLREGS

Figure 4.11: The RRT algorithm also performs flawlessly with COLREGS enabled.

4.5 The advantage of the hybrid approach

The final scenario is a modified and tougher version of one of the scenarios from [38]. It features five hostile vessels doing their best to collide with the controlled vessel. As seen in the image series of Figure 4.12, the local algorithm makes the vessel deviate from the planned path to avoid the other vessels.

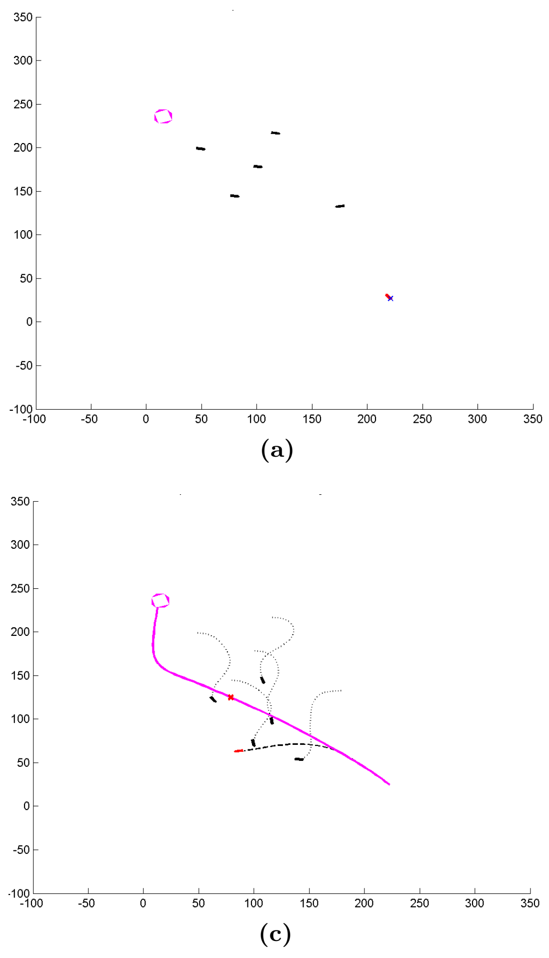


Figure 4.12: The initial configuration is shown in (a). The controlled vessel is in the lower right, and the others are hostiles. The goal is in the upper left. In (b), the RRT has generated a path, but it cannot predict the movements of the hostile vessels, so in (c) the Dynamic Window algorithm forces the vessel off-path to avoid them. The last image shows the controlled vessel back on the path, having outmanoeuvered the hostile vessels.

The simulation shown in Figure 4.12 was run with COLREGS compliance disabled. This was done for two reasons: First of all, it allowed the RRT to plan a path closer to the hostile vessels, and second it made sure the manoeuvres of the Dynamic Window algorithm were not limited by having to follow the rules.

Interestingly, this scenario is most successfully run with COLREGS enabled, using only the RRT controller. With COLREGS enabled, the RRT controller plans a path around the hostile vessels, taking the controlled vessel far away from them in an effort to minimize the COLREGS penalty, see Figure 4.13. Even though the vessel then follows the generated path in *open loop*, not paying attention to the changing surroundings, it manages to get to the goal in all the simulations because it has a higher top speed than the hostile vessels.

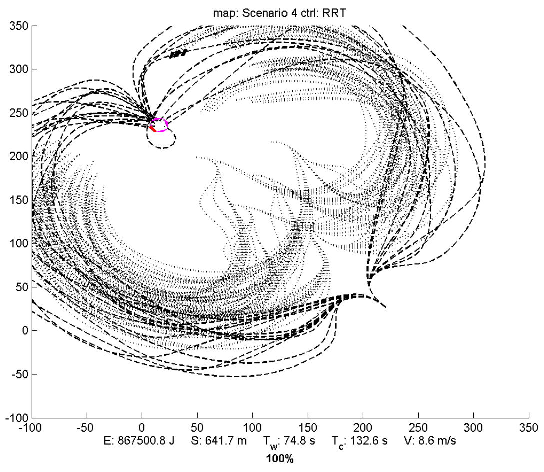


Figure 4.13:

The excellent performance shown by the RRT algorithm in Figure 4.13 is partly because of the way the scenario is set up. If the hostiles were arranged in another way, the RRT could have planned a path taking the controlled vessel closer to the hostiles, which would have resulted in a significantly lower success-rate.

In a real-world scenario, the RRT will generally not be able to cope with a rapidly-changing environment, and requires the assistance of the Dynamic Window algorithm as shown in Figure 4.12.

Chapter 5

Full-scale implementation

Simulation is an excellent tool for determining the performance of control algorithms, but since simulations introduce a lot of simplifications and inaccuracies, simulation results have to be verified by performing full-scale experiments.

One of the main goals of this thesis was to test the developed controller on a real vessel. The vessel would need computer-controllable actuators, as well as an extensive set of on-board sensors. The Trondheim-based company *Maritime Robotics* [3] early announced their interest in the project, and made their vessels available for implementation of the control system.

5.1 The Viknes 830

The target for the implementation was a vessel of type *Viknes 830*, as shown in Figure 5.1 [5]. The Viknes is a relatively small vessel with good manoeuvrability. Some technical specifications are listed in Table 5.1.

Parameter	**Value**	
Length	8.52	m
Width	2.97	m
Draught	0.82	m
Weight	3.300	kg
Power	137	kW
Top speed	10.5	$\frac{m}{s}$
Max rudder	15	deg

Table 5.1: Some of the basic Viknes 830 [5] specifications.

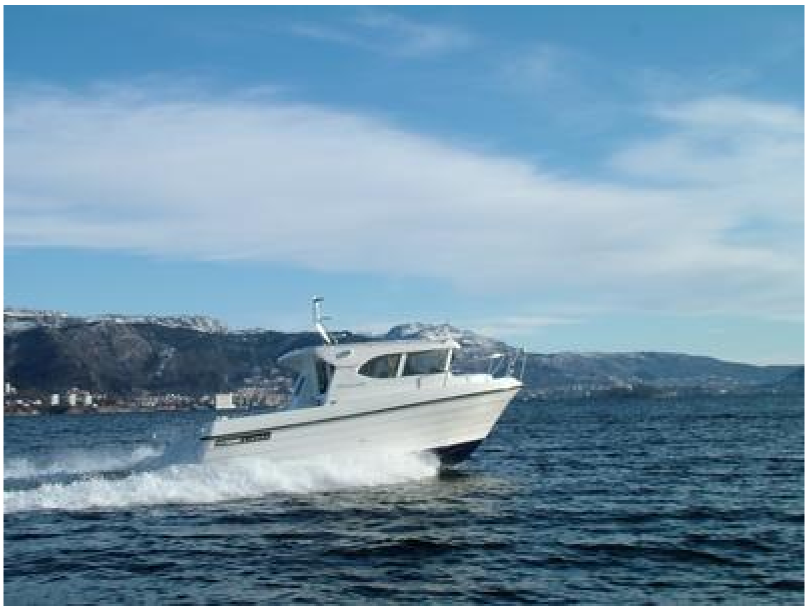


Figure 5.1: The *Viknes 830* is the testing platform for the collision avoidance system. (Image courtesy of Viknes [5])

5.1.1 Sensors

To be able to do a good job controlling the vessel, a controller needs accurate information about the state of the vessel and the vessel's surroundings. During the Matlab simulations, the controller worked with perfect information, meaning that the information received was complete and 100% accurate. In a real-world scenario this is seldom the case however, and one of the limiting factors for controller performance is the sensor outfit of the vessel.

GPS

The *Global Positioning System* [18, 20, 56], named *NAVSTAR GPS*, was developed by the United States Department of Defense. It is based on a set of *at least* 24 satellites which periodically transmit their internal time, as well as data needed to calculate their spatial coordinates. If a GPS receiver gets signals from at least 4 satellites it can calculate an estimate of its own position (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 position, the linear speeds of the vehicle (u, v, w) can be calculated. The third value w denotes the speed along the z-axis, and is not included in our 3 DOF consideration. 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.

The Viknes is fitted with a total of 3 GPS receivers. In addition to possibly giving a better position estimate, this outfit gives the system the ability to determine the true heading ψ of the vessel. When using GPS for this purpose, it is often called a *GPS compass*.

The GPS receiver transmits data every second or so, which is fast enough for controlling vessel position and velocity, but the heading measurement is a bit slow. Trying to deduce yaw rate from the heading measurement by calculating its derivative gives poor results.

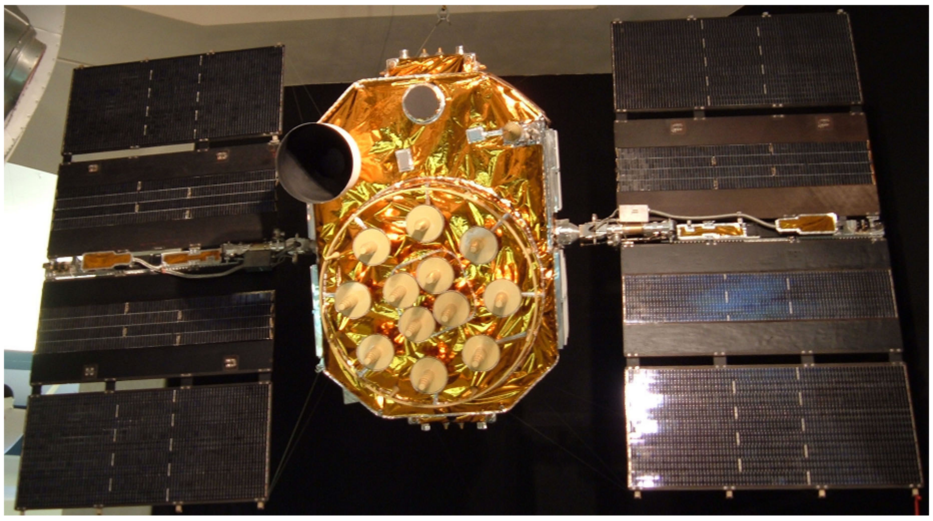


Figure 5.2: This unlaunched GPS satellite is on public display at the San Diego Aerospace Museum.

IMU

An *Inertial Measurement Unit* (IMU) [20, 58] is a component able to measure linear accelerations and angular velocities by relying 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 rates about the same axes (p, q, r).

The IMU mounted in the Viknes also includes a three-axis magnetometer. The magnetometer senses the Earth's magnetic field giving, the direction of the vessel. In contrast to the GPS compass which gives the direction of *true north*, the magnetometer gives the direction of the magnetic north like a traditional compass. The location of the magnetic north varies, and generally deviates a great deal from the *true north*, which has to be compensated for. The advantage of using measurements from the IMU (or a compass) over the GPS compass is that the measurement-frequency can be much higher, giving tighter control of the vessel.

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. The problem with these calculated velocities and coordinates is that they tend to drift, which is due to factors such as noise, quantization errors and rounding errors. Hence, 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, 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 is used to realign the coordinate and velocity estimates of the IMU to prevent drifting.

Fusing the estimates from the IMU with the measurements from a GPS unit requires some kind of filter [18]. The Extended Kalman Filter [51] is commonly used for this task. No Kalman filter has been developed for the Viknes however, so the IMU will not be used for estimation. Position and linear speeds are taken from the GPS, while angles, angular rates and linear accelerations are taken from the IMU. For the collision avoidance application this setup will be sufficient.

AIS

AIS is short for *Automatic Identification System.* This system allows vessels to exchange information required to perform effective collision avoidance [4, 41, 53].

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 within range receives it. This information gives all AIS-enabled vessels an accurate view of the situation; more accurate than that acquired by using radar only. The transmission of identification information and handling characteristics makes AIS especially powerful, as it gives the control system the ability to better predict the future behaviour of the other vessels. The transmitted ship draught will for example allow trajectories through too shallow waters to be excluded.

A downside of using the AIS system is that only vessels employing AIS are visible through the system. With some exceptions, only vessels exceeding 300 gross tonnes are required to be fitted with an AIS system [34]. Smaller vessels, hostile vessels, and other dynamic obstructions such as icebergs and kayaks, will not be detected.

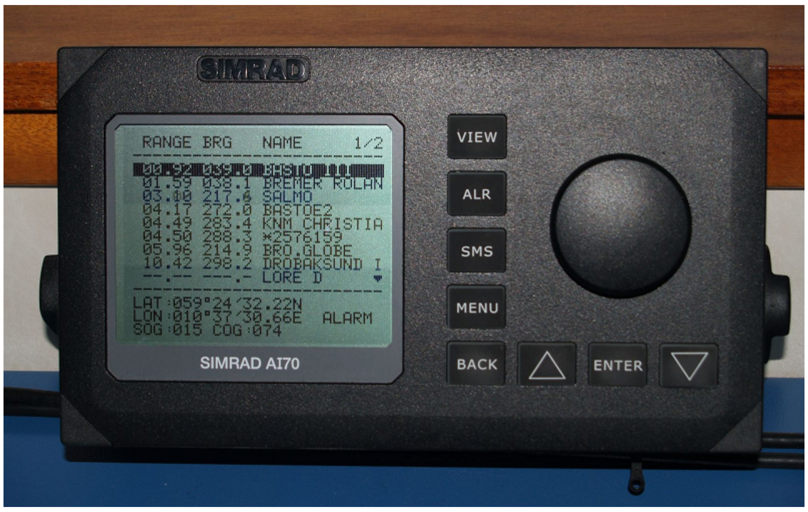

```
RANGE BRG   NAME       1/2
--------------------------
00.92 039.0 BASTO III
01.59 038.1 BREMER ROLAN
03.10 217.6 SALMO
04.17 272.0 BASTOE2
04.49 283.4 KNM CHRISTIA
04.50 288.3 *2576159
05.96 214.9 BRO.GLOBE
10.42 298.2 DROBAKSUND I
--.-- ---.- LORE D
--------------------------
LAT :059°24'32.22N
LON :010°37'30.66E   ALARM
SOG :015 COG :074
```

Figure 5.3: A text-only AIS display, listing nearby vessels with range, bearing and name.

Another downside is that collision avoidance using AIS will only work if the transmitted information is correct. A faulty speed sensor on a vessel can for instance fool other vessels to believe that the vessel is at a standstill, while it in reality is travelling at a high speed. This will make the prediction of the vessel's motion wrong, and can have a detrimental effect on the performance of the collision avoidance system. In a hypothetical situation, an enemy may also gain control of a vessel by faking AIS messages to fool the vessel's collision avoidance system.

Because of these issues, a vessel needs to have a redundant sensor system for detecting and observing other vessels and objects in the environment. In addition to AIS, employing a radar system should be a requirement. One way of incorporating radar data into the collision avoidance system is to use raw radar data directly, together with some measure of uncertainty. This is done using ultrasonic sensors in [8].

The use of radar data has not been implemented here though, and AIS will be the Viknes' primary means of detecting other vessels in this thesis.

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

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

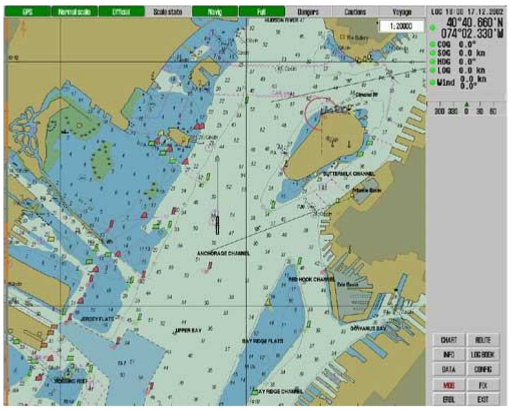


Figure 5.4: ECDIS is short for Electronic Chart Display and Information System, and represents a specific form of computer-based navigation information system that complies with IMO regulations [55]. The image shows an ECDIS display.

5.1.2 Vessel capabilities

For the application of collision avoidance, the capabilities of the Viknes vessel and its sensor outfit are good enough. The lack of a good observer fusing estimates from the IMU with measurements from the GPS should not be a problem. The state estimate calculated without such an estimator ought to be accurate enough.

The AIS system on the vessel allows the control system to detect and avoid larger vessels in the environment. For algorithm verification, this property is all that is needed, but the vessel will be unable to detect smaller vessels and other objects without AIS transceivers.

To make the system more capable in a real-life situation, the control system should also process data from the onboard radar. More sensors, like laser range finders and computer vision should also be added to enhance the vessel's perception abilities towards the surrounding environment.

The Viknes is an *under-actuated* vessel, but so are most vessels when travelling at higher speeds. The controller developed in this algorithm should be fully capable of controlling the Viknes.

5.1.3 Model identification

As explained in Section 2.1.4 it is often important to have a good mathematical system model when designing a controller. In this case it is important for two reasons:

The controllers need an internal model . Both the Dynamic Window and the RRT algorithm require knowledge about the vessel model. The Dynamic Window algorithm can get away with less information about the vessel model, but the RRT algorithm on the other hand requires a full model for integration when exploring different paths through the environment.

A model is needed for simulation . Both the simulations in Matlab, and the ones in the *Hardware-in-the-Loop* system (defined in Section 5.3), require a model of the controlled vessel. The more accurate this model is, the closer the simulations will be to the real world.

Model identification is the act of finding a good system model, in this case a model for the Viknes vessel. Using the vessel model defined in Section 2.1.4 as a basis, what remained was to find good values for the model parameters.

Mass and inertia

As seen in Table 5.2, vessel mass m and moment of inertia I_z were found by summing up the different parts of the vessel. The exact *center of gravity* (CG) was not known, so an average has been calculated.

Part	**Mass**	**Avg. dist. from CG**	**Moment of inertia**
Hull	2780 kg	2 m	11120 $kg \cdot m^2$
Engine / gear	420 kg	2 m	1680 $kg \cdot m^2$
Equipment	70 kg	3.2 m	716 $kg \cdot m^2$
Diesel tanks	300 kg	3.5 m	3675 $kg \cdot m^2$
Water tanks	120 kg	2.6 m	811.2 $kg \cdot m^2$
Septik	20 kg	2 m	80 $kg \cdot m^2$
Targa	90 kg	4 m	1440 $kg \cdot m^2$
2 people	80 kg	1 m	180 $kg \cdot m^2$
Total mass:	3980 kg	**Total moment of inertia:**	19703 $kg \cdot m^2$

Table 5.2: The parts of the vessel are summed up to calculate an estimate of the vessel mass. Combined with the average distance of the parts from the center of gravity (COG) of the vessel, the moment of inertia is also estimated.

The added mass of the vessel is more difficult to determine. If a model of the vessel's hull is available, computer programs can normally calculate an estimate by employing *strip theory*. This involves dividing the submerged part of the vessel into a number of strips and calculating the added mass for each strip. The values are then summed up, giving an estimate of the added mass of the vessel [20].

Due to time limitations, a good estimate of the Viknes' added mass was not found. The added mass is therefore set to zero, omitting added mass and added coriolis:

$$\boldsymbol{M}_A = 0 \tag{5.1}$$
$$\boldsymbol{C}_A = 0. \tag{5.2}$$

Disregarding added mass may seem coarse, as it will make the vessel model easier to accelerate than its physical counterpart, but as the other vessel parameters are calculated with the assumption of zero added mass, this should be evened out.

Damping

The damping parameters were identified axis by axis. Some work had already been put into finding the parameters for the Nomoto model (2.3) of the vessel. These parameters depend on the speed of the vessel, but a decent estimate, shown in Table 5.3, had been found.

Parameter	**Value**
K	0.5
T	4

Table 5.3: An estimate of the parameters for the Nomoto model (2.3) for the Viknes 830.

The Nomoto model (2.3) gives the following differential equation for the heading of the vessel:

$$T\dot{r} + r = K\delta. \tag{5.3}$$

This model is too simple however, as the yaw dynamics of the vessel is only relatively linear for small rudder deflections δ. For larger values of δ, the model will be too inaccurate. This inaccuracy can be improved by adding second and third order damping terms to the equation, making it more in-line with the equations of the full vessel model in (2.5):

$$T\dot{r} + n_3 r^3 + n_2 r|r| + n_1 r + n_0 = K\delta. \tag{5.4}$$

By setting $\dot{r} = 0$ in (5.4), the steady-state equation is found:

$$n_3 r^3 + n_2 r|r| + n_1 r + n_0 = K\delta. \tag{5.5}$$

This equation shows the relationship between δ and r after all transients have died out, i.e., the yaw rate which the vessel will end up with after the rudder has been held at a constant value for some time. The parameters (n_3, n_2, n_1, n_0) can now be found for the Viknes experimentally. The steady-state yaw rate for the Viknes was found for a set of different rudder deflections. A third-order polynomial curve-fit was then performed to calculate the optimal parameters for the recorded values. The parameter values are shown in Table 5.4, and with the parameters inserted, the steady state function (5.5) is plotted in Figure 5.5 along with the source data.

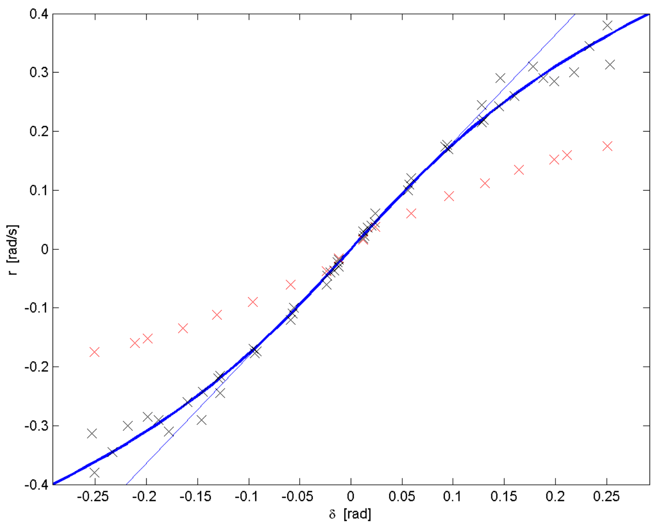


Figure 5.5: The relation between steady-state yaw rate and rudder angle was found for engine speeds giving 5, 8, 12 and 16 knots steady-state forward speed. As seen in the figure, the measurements marked by red crosses deviate significantly from the others. They were obtained at 5 knots, and because of the large deviation they were not taken into account when fitting (5.5) to the data. The measurements at the rest of the speeds, shown in black, coincide relatively well, and the fitted function shown in thick blue matches the data relatively well. The thin line shows an example of a linear approximation matching the data well for small rudder deflections. Matching in this area is important, as these are the deflections that are used most often. At larger deflections, the polynomial function provides a much better fit.

To relate the extended Nomoto model in (5.4) with the original vessel model equations (2.5), the moment of inertia of the vessel is multiplied into the equation, and the equation is rearranged. Neglecting added inertia and cross-coupling terms, this results in the

Parameter	Value
n_3	$1.3089 \cdot K$
n_2	$1.1374 \cdot K$
n_1	$0.5202 \cdot K$
n_0	$0 \cdot K$

Table 5.4: The following values were found when fitting (5.5) to the experimental data. Since (5.5) is underdetermined, the parameters for the equation depend on K.

following equation:

$$I_z\dot{r} + \frac{I_z n_3}{T}r^3 + \frac{I_z n_2}{T}r^2 + \frac{I_z n_1}{T}r = \frac{I_z K}{T}\delta. \tag{5.6}$$

The rudder bias is not included as it was found to be zero. In the case that the bias of the rudder is non-zero, this effect can be compensated for by calibration.

From (5.6) and (2.5), the following equalities can be found, which give us the values for yaw damping and the maximum lateral force generated by the rudder:

$$N_r = -\frac{I_z n_1}{T} \tag{5.7}$$

$$N_{|r|r} = -\frac{I_z n_2}{T} \tag{5.8}$$

$$N_{rrr} = -\frac{I_z n_3}{T} \tag{5.9}$$

$$F_{y,max} = \frac{I_z K}{T}\delta_{max}, \tag{5.10}$$

where δ_{max} is the maximum rudder deflection in radians as given in Table 5.1. This approximation is by no means an accurate estimate, but it will suffice.

The damping parameters in the surge and sway directions of the vessel were found in a rather different way. For the surge damping, the Viknes was run at a constant velocity for a short period of time before the main engines were shut down. For a series of different initial velocities, the velocity response was recorded as the vessel slowed down and came to a stop. The response of the vessel model was then made to match the response of the Viknes as best as possible by manipulating X_u and $X_{|u|u}$.

The lateral damping parameters could have been found by towing the vessel sideways at different speeds while measuring the required forces. Due to time constraints however, the parameters were not measured. The parameters for lateral damping presented in Table 5.5 were found by looking at the behaviour of the simulated vessel for multiple parameter values, and selecting a set of values that made the vessel movements seemingly realistic.

The cross-damping parameters N_v and Y_r were not identified, and are approximated to zero. At higher speeds, the effect of the cross-damping terms on the lateral motion of the vessel will most likely be dominated by coriolis. Since the vessel model is intended for higher speeds, neglecting cross-damping should not give large errors in the lateral movement of the vessel. The yaw rate is affected however, but the low-level controllers of the vessel should be able to cope with this effect.

Actuators

For the force generated by the vessel engine F_x, the stationary velocity was recorded for the engine at different RPMs. This data was then matched against engine charts for the Viknes giving the effect of the engine at the different RPMs to calculate an estimate of the force generated by the engine. Fortunately, the force ascended relatively linearly with the engine RPM, maxing out at $F_{x,max} = 13.1\,kN$ at 3200 RPM.

Although the control system is able to make the vessel travel backwards, reverse force is mostly used for decelerating the vessel. Lacking good measurements, the maximum decelerating force $F_{x,min}$ is estimated to approximately half the forward force. The reduction in power efficiency comes from a combination of several issues, including the propeller being less effective when run backwards and the water flowing the wrong way past the propeller.

The final parameters

The final vessel model parameters are summarized in Table 5.5. For surge speeds larger than 8 knots, they should provide a sufficient estimate for simulation and prediction.

Several rough estimates were made in order to find the parameters however. The neglection of added mass is somewhat compensated for in the computation of the damping and force parameters, but error in the lateral damping and cross-damping parameters will result in a model that deviates somewhat from the physical vessel.

In simulations, the modelling errors will not be problematic, as both the controllers and the vessel model in the simulator uses the same parameters. They do however give a certain amount of uncertainty to the results obtained by simulation, as the simulated vessel does not behave exactly as its physical counterpart.

The importance of full-scale experiments was mentioned at the beginning of this chapter, and this is clearly seen here. To remove the uncertainty associated with the simulations, verification is needed. This is why full-scale experiments are important.

Parameter	Value	
m	3980	kg
I_z	19703	$\frac{kg}{m^2}$
X_u	-50	
$X_{\|u\|u}$	-135	
X_{uuu}	0	
Y_v	-200	
$Y_{\|v\|v}$	-2000	
Y_{vvv}	0	
N_r	-1281	
$N_{\|r\|r}$	0	
N_{rrr}	-3224	
N_v	0	
Y_r	0	
$F_{x,max}$	13100	N
$F_{x,min}$	-6550	N
$F_{y,max}$	645	N
l_r	4	m

Table 5.5: Parameters for the vessel's equation of motion. The parameters approximate the parameters of the *Viknes 830* [5].

5.2 Control system

The control system for the physical vessel is built on the *Control Design Platform* (CDP) framework provided by *Industrial Control Design* (ICD) [15], a Norwegian company located in Ålesund. As its name suggests, CDP is a platform for control systems. It allows a control system to be decomposed into several smaller components, making development more manageable. CDP also acts as a layer between the operating system of the control computer and the software, allowing the control system to be run on a multitude of different operating systems. The framework is written in C++, but includes functionality for implementing control algorithms in Matlab.

A brief overview of the control system is shown in Figure 5.6. An observer is responsible for fusing sensor data to provide a good estimate of the vessel state, as well as creating an image of the surrounding environment. This data is passed on to the different controllers, and together with operator input, provides the input to the control system.

The control system is divided into several layers. The RRT algorithm assisted by A* is responsible for creating a path through the environment using all available information. The path tracker calculates the heading and velocity setpoints required to follow the path. These are only suggestions though, as the Dynamic Window algorithm in the

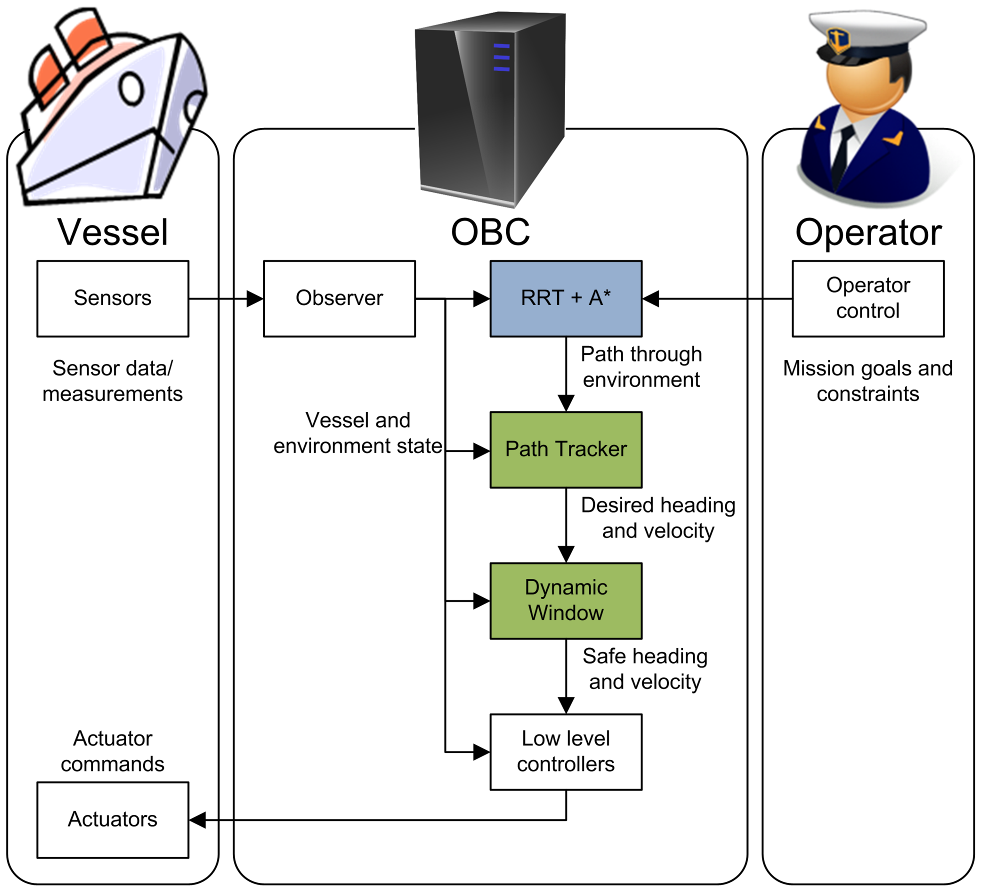


Figure 5.6: The vessel control system consists of several self-contained components which cooperate in an effort to control the vessel according to the operator's goals.

next layer freely manipulates the setpoints to avoid dangerous situations. The final layer consists of the low-level controllers of the vessel, which attempt to track the setpoints given by the Dynamic Window algorithm.

5.2.1 Low-level controllers

When moving to a real vessel, the low-level controllers defined in Section 2.1.4 no longer suffice. They rely on explicit knowledge of the vessel model which is no longer available. The model defined by the equations in (2.5) most likely is too simple, and its parameters too inaccurate for feedback linearization.

A more robust method for control design is *sliding mode*, but in this implementation, simple PID regulators are used for all low-level controllers. Implemented with gain-scheduling, these perform a good job in controlling the vessel.

Gain-scheduling is required since the response of the vessel to certain inputs change in magnitude depending on the state of the vessel. A small rudder deflection will for instance have more effect on the vessel at higher than lower speeds. A gain-scheduling controller compensates for this effect by changing the controller gain according to the system state, making sure the system is stable and has a good response for a large range of different states.

Both the heading and yaw rate controllers on the Viknes are gain-scheduled with respect to the vessel's surge speed u. They both generate desired rudder deflections δ_{sp} for the *rudder controller* which is responsible of positioning the rudder correctly. The PID controller for heading and yaw rate is given by:

$$\delta_{sp}(t) = \underbrace{K_{p,h}(u)e_h(t) + K_{i,h}(u)\int_0^t e_h(\tau)d\tau + K_{d,h}(u)\frac{d}{dt}e_h(t)}_{PID\ FB}, \tag{5.11}$$

where $e_h(t)$ is the error in the controlled value. When controlling the heading of the vessel the error is defined as $e_h(t) \triangleq \Gamma(\psi_{sp}(t) - \psi(t))$, and when controlling yaw rate the error is $e_h(t) \triangleq r_{sp}(t) - r(t)$. The function $\Gamma : \mathcal{R} \rightarrow \langle -\pi, \pi]$ maps the heading error to the interval $\langle -\pi, \pi]$, making sure that the proportional term of the controller always causes the vessel turn the correct way in order to reduce the heading error. For the yaw rate controller this is not required. The controller gains $K_{p,h}(u)$, $K_{i,h}(u)$ and $K_{d,h}(u)$ are gain-scheduled to make sure the controller performs well for all feasible surge velocities.

Gain scheduling is not needed in the speed controller of the vessel, as the gain-change here is not as drastic as for the heading and yaw rate modes. The PID controller for speed controls the engine throttle ω:

$$\omega(t) = \underbrace{K_{p,s}e_s(t) + K_{i,s}\int_0^t e_s(\tau)d\tau + K_{d,s}\frac{d}{dt}e_s(t)}_{PID\ FB}, \tag{5.12}$$

where $e_s(t) \triangleq u_{sp}(t) - u(t)$ is the error in the speed of the vessel, i.e., the difference between desired and actual speed. As opposed to the controllers defined in Section 2.1.4, these low-level controllers only contain feedback terms; they do not try to compensate for the actions of the vessel before they can be measured.

The outputs of the low-level controllers can be related to the equations of motion defined in Section 2.1.4 by estimating a linear relationship between the actuator inputs and their generated forces. We also neglect the dynamics of the rudder, giving $\delta = \delta_{sp}$, as well as the dynamics of the engine. Since the dynamics of these actuators are generally much faster than the vessel dynamics, this approximation is not too crude. The forces acting on the vessel under these assumptions become:

$$F_x = K_1\omega \tag{5.13}$$
$$F_y = K_2\delta, \tag{5.14}$$

where K_1 and K_2 are some constants. As infinitely large forces are not feasible, the values of ω and δ are limited. This may cause the integrator in the PID controllers to *wind up*, resulting in bad behaviour. This is why integral terms were not used in the controllers in Section 2.1.4. Many techniques exist to solve this problem though [46].

5.3 HIL system

The simulations done in Section 4 give a good indication of the performance of the vessel control algorithms. The vessel control system consists of more than just algorithms however, both in terms of software and hardware. These aspects include failure handling routines, I/O interfaces, operator interfaces, etc., and result in a total system that is relatively complex and requires extensive testing and verification before being put into use.

This is where *Hardware-in-the-Loop* (HIL) simulations come into play. A HIL simulator employs a real-time simulator as a replacement for the physical vessel. The simulator simulates all signals from the vessel to the control system, optionally adding effects as sensor dynamics and noise. All commands output by the control system are received and applied to an internal vessel model, simulating the motion of the vessel. During execution, the control system should not experience any qualitative differences from being used with the physical vessel [28, 43].

The strength of the HIL simulator is the broad range of scenarios that can be tested without the need of sea trials. As both the hardware and the software making up the control system is included in the simulation, various failure scenarios can be simulated, including power outages, sensor noise, different network conditions, etc. These features enable the verification of failure detection and handling of a number of failure modes. Realistic operational scenarios can also be simulated to verify the functionality of the control system under different operating conditions.

By using a HIL simulator, extensive testing can be done early in the development process. While it cannot replace the costly full-scale sea trials completely, the need for such trials is reduced since basic problems can be found and corrected at an earlier stage. Another benefit with HIL simulation is that it allows the testing of scenarios that are too expensive or risky to test in full-scale. The control system can thus be tested both deeper and more thorougly at a lower cost with a HIL simulator than with conventional testing [28].

5.3.1 HIL for the Viknes

An overview of the HIL setup for the Viknes is shown in Figure 5.7. The physical vessel has now been replaced by a simulation computer. Communication between the control computer, also known as the *on-board computer* (OBC) and the simulator is done through the normal hardware I/O interface of the OBC (analog, digital, serial, etc.) to include these interfaces in the simulation.

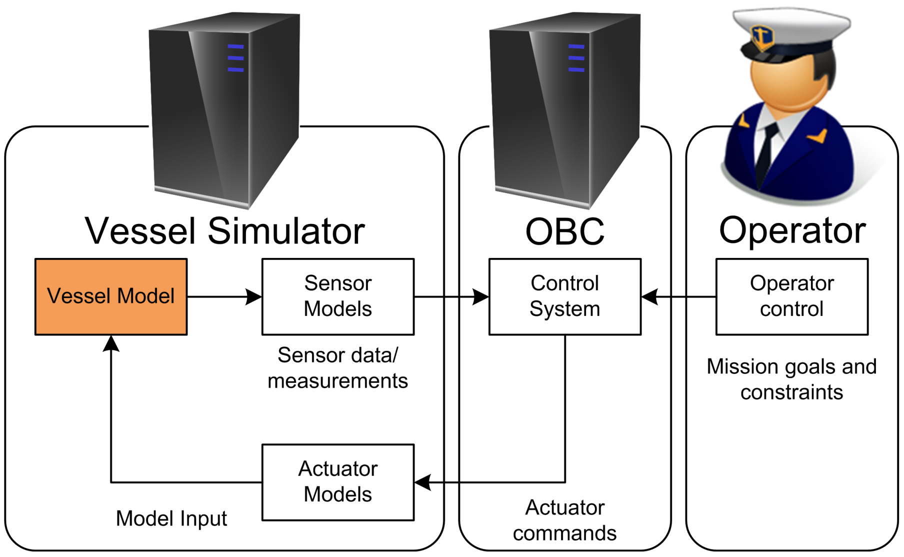


Figure 5.7: The HIL simulator replaces the physical vessel with a software-simulated vessel. The vessel simulator contains a model of the vessel, its sensors and its actuators, and emulates the vessel for interaction with the *on-board computer* (OBC). Dealing with a HIL simulator or dealing with the physical vessel should be qualitatively the same from the OBC's viewpoint. This feature allows extensive testing of the OBC to be performed on land in a controlled environment.

The HIL system allows most of the aspects of the control computer to be tested and verified, but also represents a great development platform. The simulator contains a mathematical model of the vessel as well as its sensors and actuators, allowing control algorithms to be tested in real-time with very little effort. During development this proved very valuable, as most bugs in the control system could be found without having to test the system on the physical vessel.

The vessel model in the HIL system is based on the model presented in Section 2.1.4 with the parameters given for the Matlab simulations in Chapter 4, which should make the results from the HIL system somewhat comparable with the results from the Matlab simulations. What separate them is the different low-level controllers, the communication time delay in the control system and simulator, and a more realistic sensor model. The control system no longer has perfect knowledge, but relies on data from the available sensors.

Chapter 6

Full-scale experiments

One of the main goals of the work leading up to this thesis was to test the developed control system on a real vessel in a set of full-scale experiments. To straighten out all quirks in the control system and its algorithms, extensive testing was performed on a HIL simulator (see Section 5.3) prior to running the system on the real vessel. This was nonetheless an iterative process, as not all errors were detected during HIL testing.

An interesting question is how well the HIL simulator is able to emulate the physical vessel. In an attempt to answer this question, the scenarios presented in this chapter have been run on both the HIL simulator and in the real world. As another vessel is part of the scenarios, controlled by the simulator in the first case and a human in the second, the scenarios will not be completely identical, but hopefully identical enough for qualitative comparisons.

The first scenario uses purely reactive control to show the vessel's ability to follow the commands given by the Dynamic Window algorithm. The second scenario uses the full control system. Both of the scenarios feature another vessel approaching the controlled vessel, to verify that the controllers will choose a COLREGS-compliant course of action.

One of the main differences when moving from the Matlab simulations to the HIL simulator and the real vessel is, from the control algorithm's point of view, the lack of perfect knowledge about the vessel state and the environment. The GPS only transmits position coordinates every second or so, and other vessels are tracked solely by messages from the AIS system. Such aspects introduce time delays into the system, as it takes some time before the response of a control input is measured. Additional time delay comes from actuator dynamics and the time which signals take to propagate through the control system.

The introduced time delays were handled at multiple levels. The low-level controllers had to be tuned less aggressively than their Matlab counterparts to avoid oscillations and instability. The parameter T_r of the Dynamic Window algorithm, determining the settling time to the desired heading, had to be increased to $T_r = 1$ for the same reason, and finally the parameters of the path tracker were increased to $\Delta = 90m$ and $R_a = 60m$. Ideally, Δ and R_a should be functions of the surge speed of the vessel, increasing monotonically with the speed, but the given values resulted in good performance at the higher velocities used in the following experiments.

Because the throttle-controller hardware for the Viknes was delayed, and could not be delivered in time for the experiments, both the HIL simulations and full-scale experiments were performed at constant speed of 6 m/s.

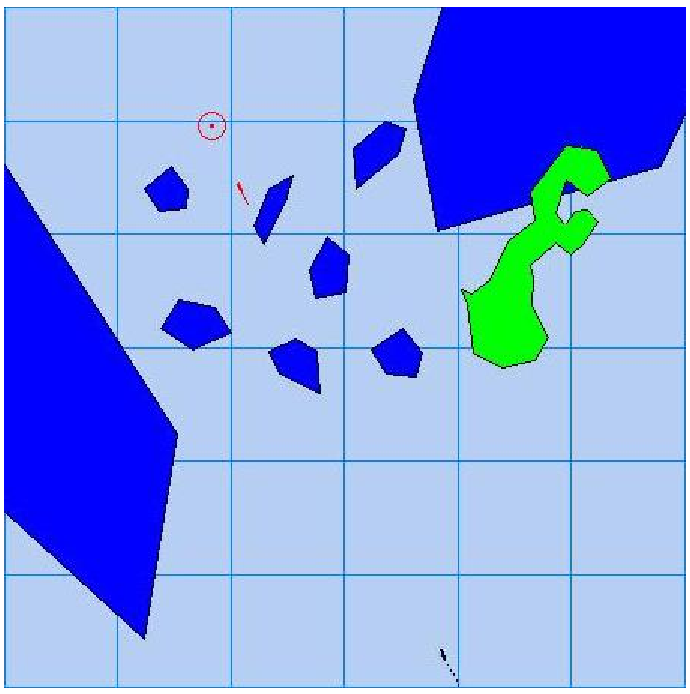
The environment is based on the Trondheimsfjord just north of Trondheim, Norway. The green polygons represent land while the dark blue polygons represent *forbidden areas*, attempting to make the environment a bit more interesting. The grid spacing in the figures is 250 metres.

6.1 HIL simulations

In the HIL simulations, the other vessel is controlled by the simulator. It travels in a straight line, making good prediction possible by both the Dynamic Window and RRT algorithms.

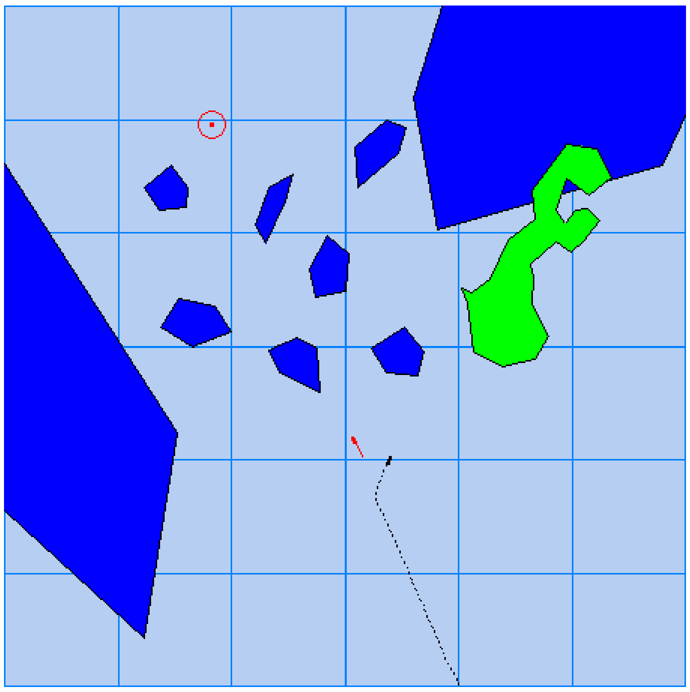
6.1.1 Reactive scenario

In the first scenario, the vessel is controlled solely by the Dynamic Window algorithm. It starts out in the lower right of the map, travelling towards the circular target in the upper left. The controlled vessel encounters the other vessel in a head-on situation along the way, opening up for a COLREGS-compliant manoeuvre. The results are shown with comments in Figure 6.1.

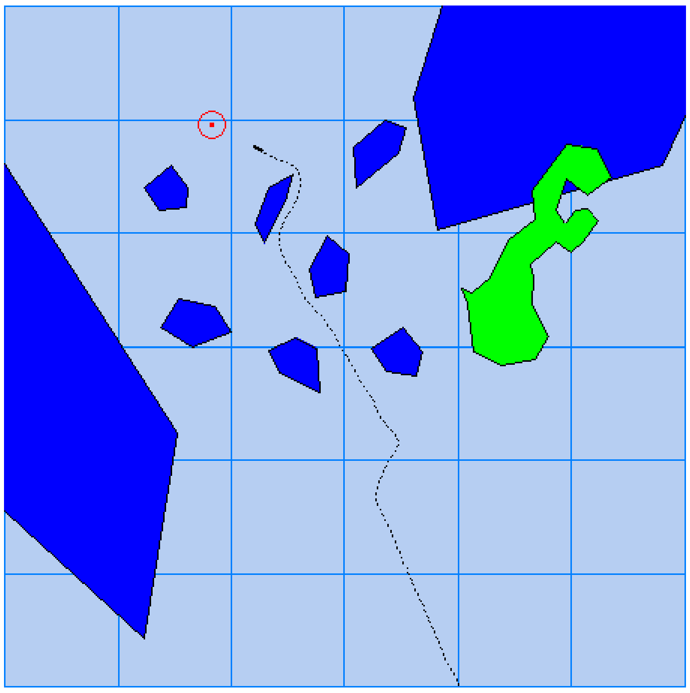


(a) Starting out

Figure 6.1: The vessel starts out in the lower right, and has to get to the target in the upper left while avoiding the other vessel and the blue forbidden zones.



(b) COLREGS manoeuvre



(c) Final trace

Figure 6.1: As seen in (b), the controlled vessel passes the other vessel on the correct side. It also manages to steer clear of the forbidden zones, resulting in a successful simulation.

6.1.2 Hybrid scenario

In the second scenario, the full hybrid controller is put to work. The controlled vessel starts out in a local minimum in the lower center of the map, and has to reach the target near the top of the map. The fastest way is in-between the two forbidden zones, where it meets the other vessel which is travelling downwards. For the experiment to be a success, the controlled vessel has to pass the other vessel in a COLREGS compliant-fashion, and get to the goal. The results are shown in Figure 6.2.

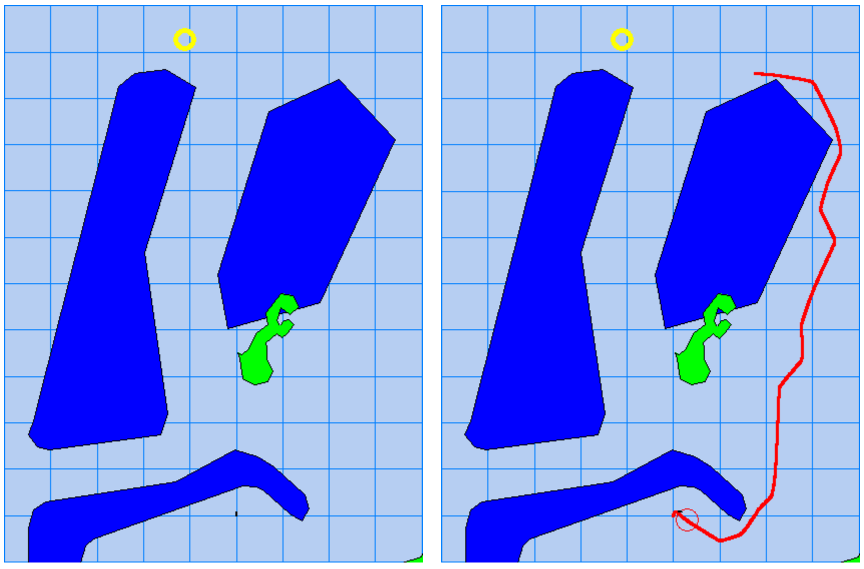
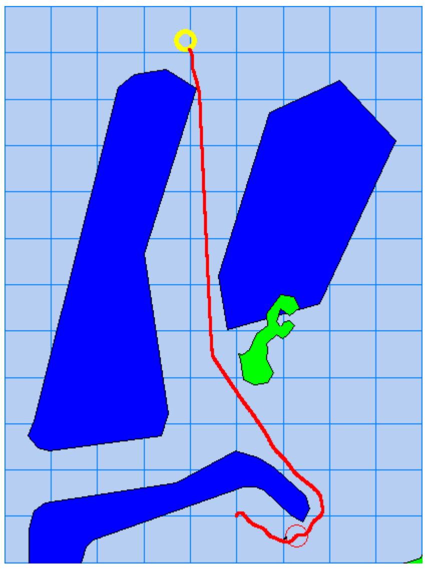
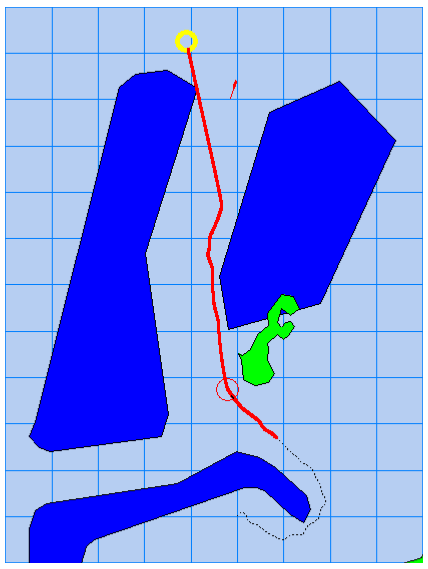
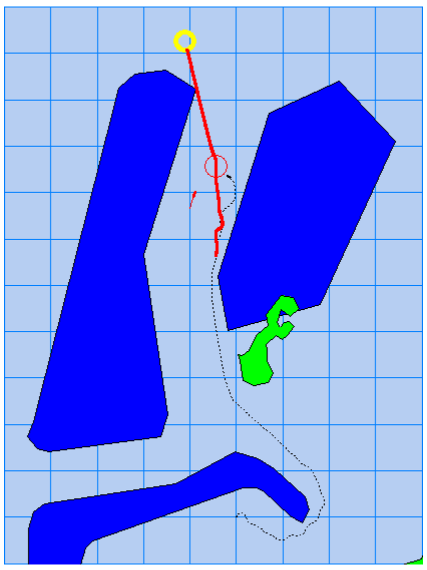

(a) Starting out **(b)** Initial RRT path

Figure 6.2: Figure (a) shows the initial state. The controlled vessel is the barely visible black dot in the lower part of the map, and the other vessel is not yet visible. The yellow circle in the upper left is the target. As seen in (b), the RRT algorithm actually fails in finding a solution to the problem in the first iteration, but the chosen route still provides good progress towards the goal. The first part of the route will also act as a guide for the second iteration. The circle shown on the RRT path in (b) shows the next waypoint along the path.


(c) Improved RRT path


(d) COLREGS compliant path


(e) Deviating from path

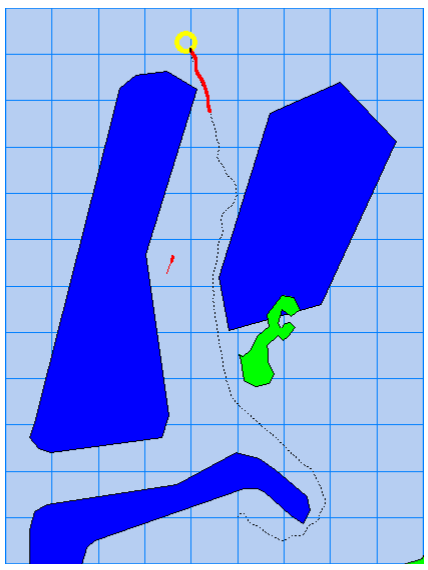

(f) Final trace

Figure 6.2: In (c), the second iteration of the RRT algorithm has generated a valid path to the goal with good help from the first iteration. The other vessel has entered the area from the top in (d), and the RRT algorithm has generated a path leading the controlled vessel clear of the approaching vessel. The RRT algorithm leads the controlled vessel a bit close to the other vessel, causing the Dynamic Window algorithm to make the vessel deviate from the path as shown in (e). The controlled vessel reaches the goal in (f), ending the simulation with success.

6.2 Sea trials

The sea trials were done with the other vessel being controlled by a human, trying as best to steer it as in the HIL simulations. As explained in Section 5.1.1, the controlled vessel relies solely on AIS for information about the state of the other vessel. In the final experiments, the AIS data transmitted from the other vessel contained invalid heading and yaw rate data, resulting in a misrepresentation of the vessel state. The position and speed data was valid however, and the decision was made to go through with the experiments.

The invalid heading and yaw rate data resulted in gross mispredictions of the movements of the other vessel because the predictions in the system are based on integration of the full vessel model (2.5). As seen in Figure 6.3, the other vessel appears to be travelling near-sideways, and the large damping in this direction results in a prediction where the velocity of the other vessel is reduced very quickly.

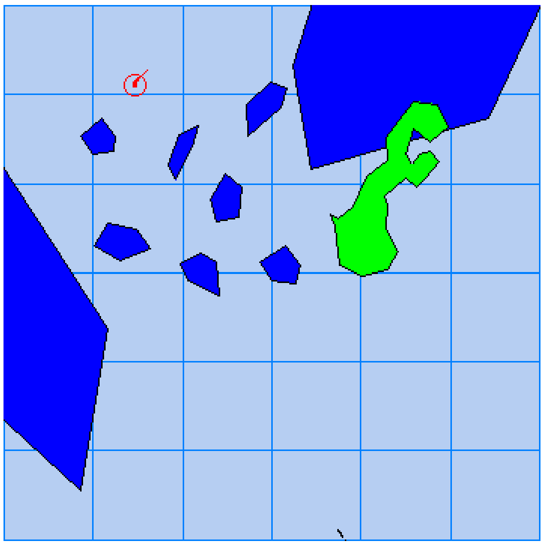
Despite the prediction errors, the control system still manages to keep the controlled vessel clear of collision. This achievement is a sign of the robustness of the collision avoidance system.

6.2.1 Reactive scenario

Like in the reactive scenario for the HIL simulator, the reactive scenario features control by the Dynamic Window algorithm alone.

As seen in Figure 6.3, the controlled vessel completed the scenario, but not as gracefully as on the HIL simulator. As it could not predict the movements of the other vessel correctly, it took a lot more time before it started executing evasive manoeuvres to avoid the other vessel. It eventually saw the vessel, but as it travelled relatively fast at 8 m/s, giving a relative velocity of 14 m/s, only a limited amount of manoeuvring was possible before the vessels passed each other. The margins were relatively small.

Moving through the area with the forbidden zones, the control system had a bit of trouble steering clear of the first zone it encountered, brushing up against the side of it. The main reason for this was the yaw rate controller, which was tuned too conservatively. This resulted in a significant delay from the time a command was issued by the Dynamic Window algorithm to when it was put into action. The second zone was no match for the controller.


(a) Starting out

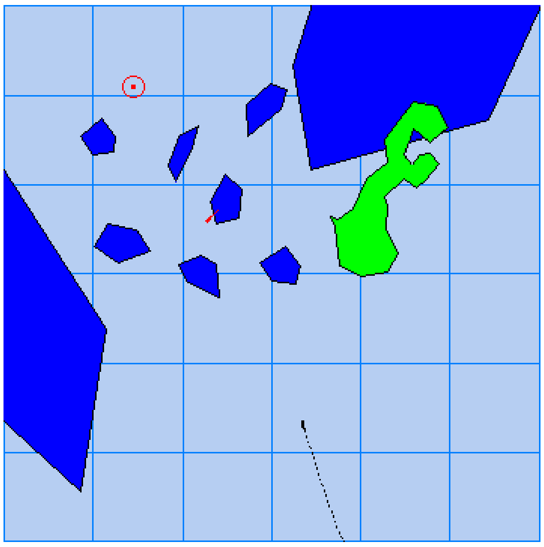
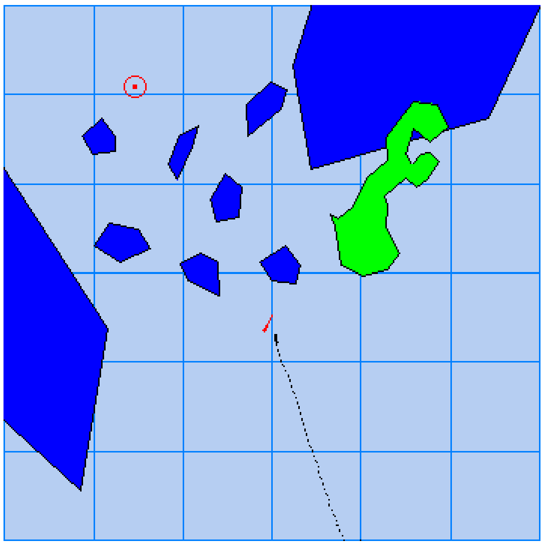

(b) Approaching

Figure 6.3: The initial conditions are shown in (a), with the controlled vessel in the lower part of the map, and the other vessel within the goal area. In (b), the vessels are approaching one another. Looking closely at the other vessel, you can clearly see the error in the received heading data: The vessel seems to be travelling sideways towards the controlled vessel.


(c) Evasive manoeuvre

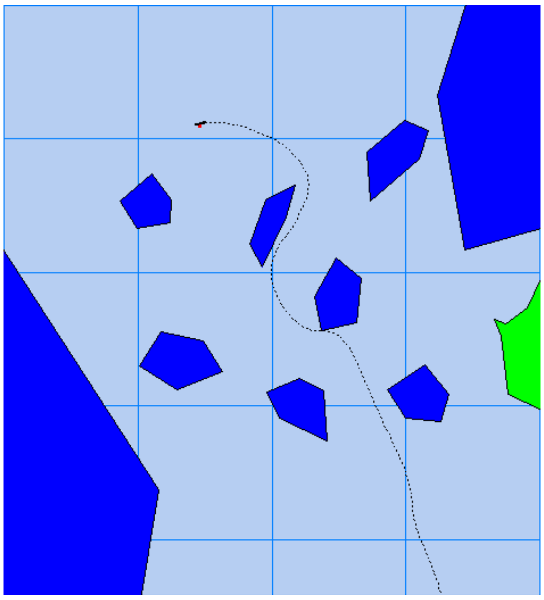
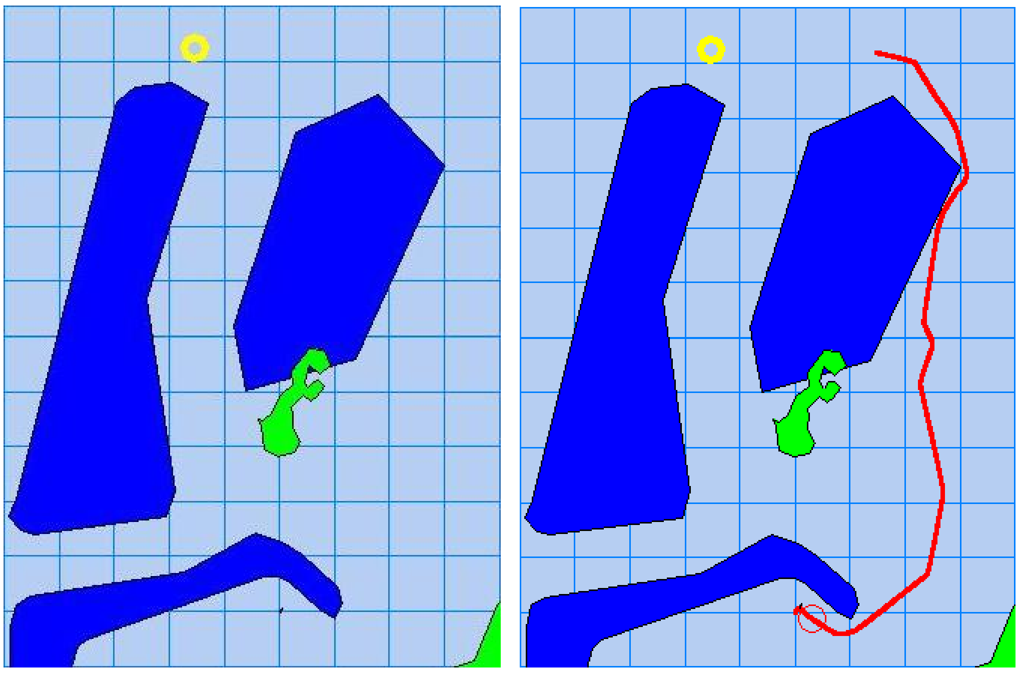

(d) Final trace

Figure 6.3: As the prediction capabilities are severely reduced, the controlled vessels reaction to the other vessel comes relatively late. In (c) the controlled vessel has just started an evasive manoeuvre. The final trace is shown in (d).

6.2.2 Hybrid scenario

The hybrid scenario was the chance to test the full control system on the physical vessel, i.e., the ultimate test, containing both a local minimum and a potential COLREGS situation.



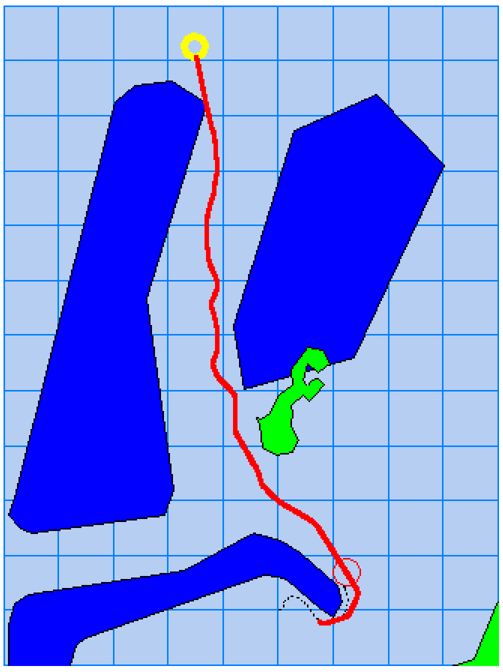
(a) Starting out

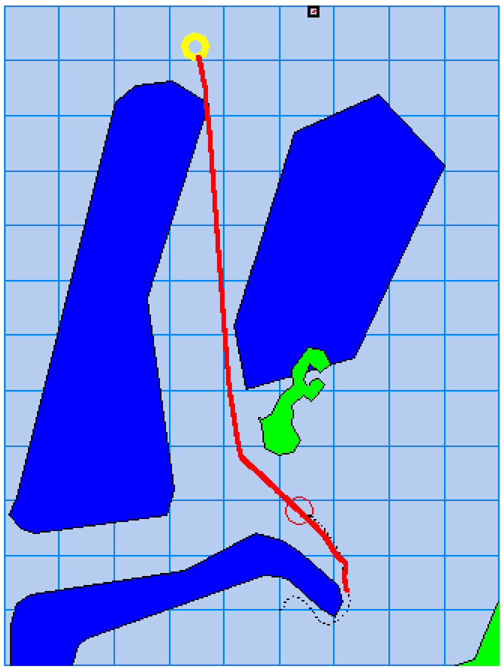
(b) Initial RRT path

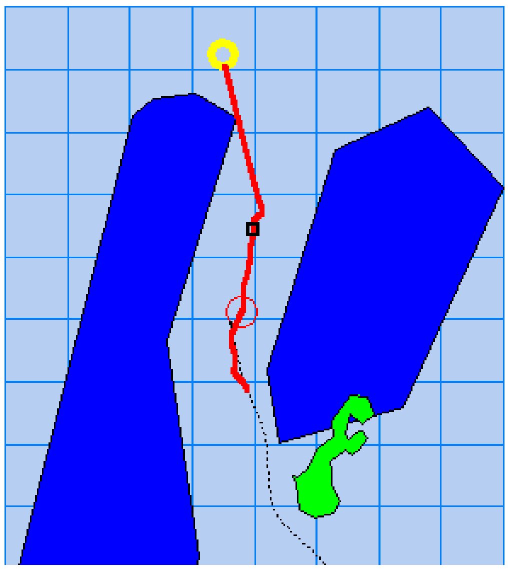
Figure 6.4: The initial condition is shown in (a), with the first RRT path having been generated in (b). The other vessel has not yet entered the map.

The results from the scenario are shown in Figure 6.4. Except from the prediction problems with the other vessel, the performance of both the RRT and Dynamic Window algorithm was excellent. Identically to the HIL simulation, the first RRT iteration was unable to find the goal, but still came up with a reasonable path. The next iteration improved on the first, and all later iterations found valid paths.

Because of the prediction problems, the passing of the other vessel was not as smooth as in the HIL simulation. The RRT algorithm managed to create a path steering clear of the other vessel however, but only by a small margin. In this scenario, the other vessel travelled at a low 3 m/s, so when approaching, the Dynamic Window algorithm had enough time to steer clear by taking the controlled vessel away from the planned path to pass the other vessel at a safe distance.


(c) Improved RRT path


(d) Other vessel enters


(e) COLREGS compliant path

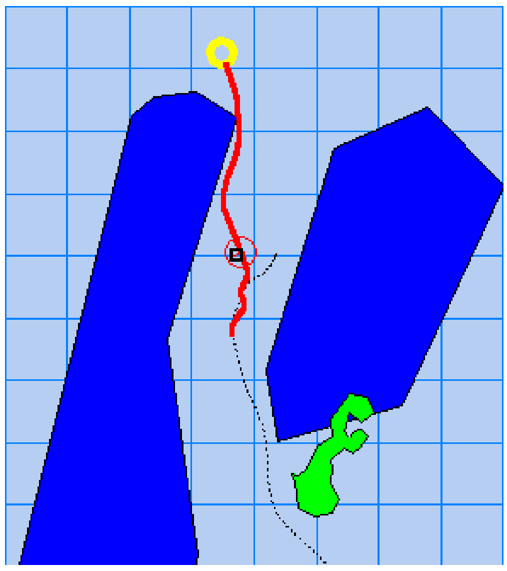
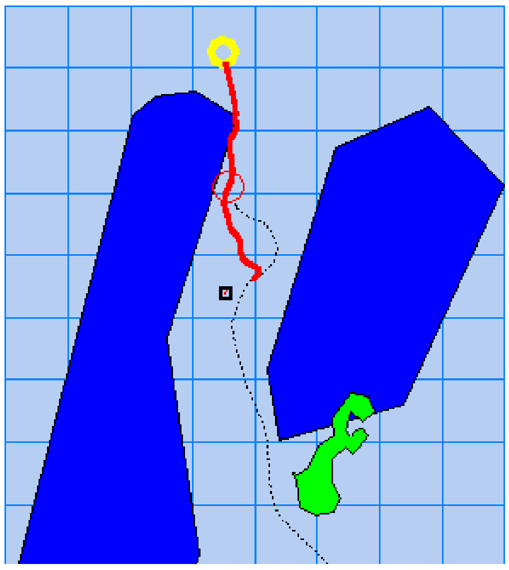
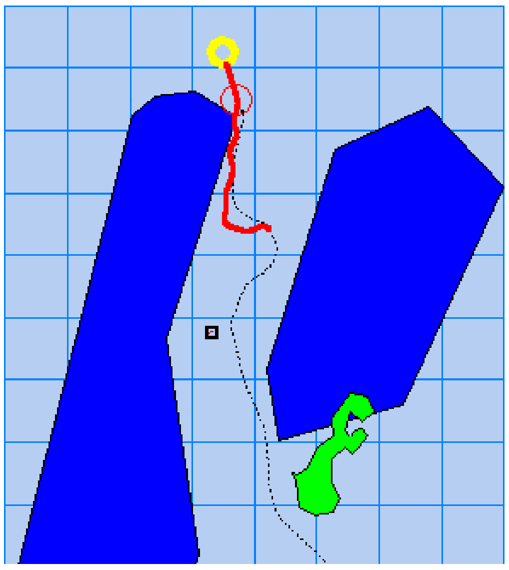

(f) Deviating from path

Figure 6.4: Figure (c) shows the first path reaching the goal. The other vessel enters the top of the map as a tiny red dot in (d). A black rectangle has been placed around the vessel to make it easier to see. Both in (e) and (f), the RRT algorithm has generated a path keeping clear of the vessel, but because of the mispredictions, the generated path takes the vessels very close. This is corrected by the Dynamic Window algorithm in (f), by steering the controlled vessel off-path to keep a safe distance to the passing vessel.


(g) Back to the path


(h) Approaching goal

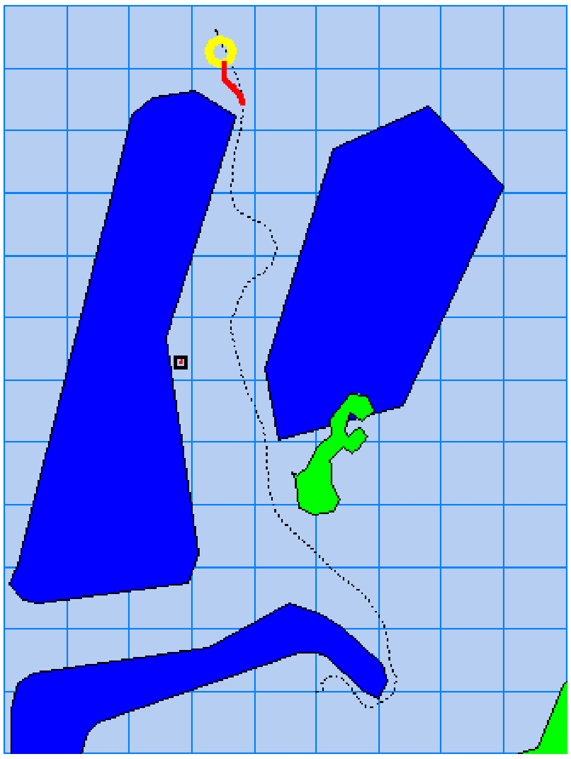

(i) Final trace

Figure 6.4: The vessel returns to the path and continues towards the goal in (g) and (h). The final path trace is shown in (i).

Chapter 7

Control system analysis

When designing a control system, simulations and live testing are often the primary tools for determining the properties of the system. Tests show how the system behaves, how it is influenced by external influences such as noise, whether it reaches its goals, and so on. Performing tests would seem to be everything needed to create a control system. It is not that easy however.

The problem with simulations and live testing is that they only represent a subset of the possible situations that the control system may be exposed to in the real world. If the system performs flawlessly during testing, this is a good indication of how the system will perform under other scenarios, but there is no way of knowing whether a specific scenario will cause trouble.

For discrete systems it is sometimes possible to simulate and test all possible scenarios, but for general systems, the number of scenarios is too large and often infinite. Another approach is required, and this is where control system analysis comes into play.

7.1 Introduction

When analysing control systems, three properties are of primary concern: *stability*, *transient behaviour* and *robustness*. Stability says something about what happens to the system as time approaches infinity. In [30], stability is defined as follows: Given an autonomous system with state vector $\mathbf{x}$, and a derivative given by a function $\mathbf{f}$:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \tag{7.1}$$

with an equilibrium point at $\mathbf{x}_g$, i.e., $\mathbf{f}(\mathbf{x}_g) = 0$. This system is said to be *stable* if for each $\epsilon > 0$ there is a $\delta(\epsilon) > 0$ such that:

$$\|\mathbf{x}_g - \mathbf{x}(0)\| < \delta \implies \|\mathbf{x}_g - \mathbf{x}(t)\| < \epsilon \ ,\forall\, t \geq 0. \tag{7.2}$$

This relation means that if the system is stable and starts out at $\mathbf{x}(0)$ at some distance from the goal $\mathbf{x}_g$, the distance from the goal will be bounded by a function of the initial distance for all times. A stable system will not move uncontrollably away from the goal, but it cannot be guaranteed that the system will ever reach the goal either.

A stronger type of stability is *asymptotic stability.* A system is asymptotically stable if it is stable and a δ can be chosen such that:

$$\|\mathbf{x}_g - \mathbf{x}(0)\| < \delta \implies \lim_{t\to\infty} \|\mathbf{x}_g - \mathbf{x}(t)\| = 0. \tag{7.3}$$

An asymptotically stable system *will* reach its equilibrium point, but there is no guarantee for how much time it will take, or how the system will reach it. This is the problem with the definitions of stability. Stability and asymptotic stability can be proven for many systems, at least locally, but to be able to evaluate the system's approach to the goal, their *transient behaviour* has to be analysed.

A transient can be defined as the path that the system takes from an initial state $\mathbf{x}(0)$ to a goal state $\mathbf{x}_g$. For linear systems, there exist many tools for analysing and managing transient responses. For more general nonlinear systems however, there are very few. *Exponential stability* is one such measure. It tries to relate the transient of a nonlinear system to that of a linear system to give a bound on the transient. A system is exponentially stable if there exist positive constants c, k, and λ such that:

$$\|\mathbf{x}_g - \mathbf{x}(t)\| \leq k\|\mathbf{x}(0)\|e^{-\lambda t},\ \forall\, \|\mathbf{x}(0)\| < c. \tag{7.4}$$

Robustness is the ability of a system to perform well even in the presence of modelling errors, sensor and actuator inaccuracies, noise, and other parameters that are unknown at design time. The level of robustness indicates how large inaccuracies the system can handle before performance and/or stability is severely degraded.

For general nonlinear systems, there are very few tools for determining and ensuring robustness. The problem with these systems is that there are so many different variations and levels of complexity, that even though there exist tools, these are normally targeted towards specific classes of systems.

7.2 Analysis

As low-level controller design is not the purpose of this thesis, the low-level controllers are assumed to be *locally uniformly asymptotically stable*, meaning that they will drive the system to their setpoints given the right circumstances. For the PID controllers (5.2.1) used in this implementation, these properties can be achieved by correct tuning of the regulator parameters $K_{p,h}(u)$, $K_{i,h}(u)$, $K_{d,h}(u)$, $K_{p,s}$, $K_{i,s}$ and $K_{d,s}$.

For a collision avoidance system, stability can be defined in much the same way as in Section 7.1. Asymptotic stability of the control system implies convergence to a given goal state in an infinite amount of time. To be able to reach the goal, the controlled vessel must avoid colliding with other vessels along its route, as a collision will prevent the vessel from continuing towards the goal. Asymptotic stability thus also means that the vessel manages to avoid collisions.

In accordance with the definitions, a stable collision avoidance system is able to avoid collisions, but is not necessarily able to make the controlled vessel converge to the goal state.

7.2.1 The simplest case

Given a good vessel model and an environment with only static obstacles, the paths generated by the RRT algorithm will be feasible for all time, and no action is required by the dynamic window algorithm to avoid unforeseen events. In this case, asymptotic stability of the algorithm depends on two factors: the RRTs ability to generate a path to the goal, and the path tracker's ability to follow the generated path.

Tracking the path

The generated path consists of a chain of straight lines. When tracking a line using (3.36), the yaw rate command to the vessel is given by the following equation:

$$\dot{\psi}_{sp} = \frac{u \cdot e}{(e^2 + \Delta^2)^{\frac{3}{2}}}, \tag{7.5}$$

which, given a constant Δ, is maximized for $e = \frac{\Delta}{\sqrt{2}}$. If the corresponding $\dot{\psi}_{sp}$ is within the capabilities of the vessel, it will have no problem tracking and converging to the line.

When moving to a path consisting of several straight-line segments, the heading command ψ_{sp} will no longer be continuous. Every time the controller switches to a new line, there will be a step in the heading command; a discontinuity. The size of the step depends on the difference in angle between the new and the previous line, giving a larger step for a greater difference in angle.

The step in the heading command can result in the path tracker issuing commands which the vessel is not able to perform, especially for paths with very short lines and great variations in line angles. Path trackers able to cope with this is possible to synthesise, for instance by actively adjusting Δ, but in our case it is not necessary. The RRT algorithm uses the dynamics of the vessel when generating paths, so we can be sure that the path is possible to follow by our vessel given a correctly tuned path tracker.

Dynamic Window

The Dynamic Window algorithm lies as a layer between the path tracker and the low-level controllers. When the commands given by the path tracker are safe, the Dynamic Window algorithm should optimally pass them directly through, but as the controller uses yaw rate to control the vessel, the command is modified to some extent.

When no obstructions are in the vicinity of the vessel, the generated yaw rate command is found by maximizing the *heading objective* of the algorithm, which is the same as assigning $\hat{\psi} = \psi_{sp}$ in (2.30). The heading command ψ_{sp} is here given by the path tracker. Substitution results the following equation:

$$r|r| + 2\dot{r}_{max}T_r r + 2\dot{r}_{max}(\psi - \psi_{sp}) = 0, \tag{7.6}$$

which when solved for r results in the following control law:

$$r = \operatorname{sgn}(\psi_{sp} - \psi)\left(T_r\dot{r}_{max} + \sqrt{T_r^2\dot{r}_{max}^2 + 2\dot{r}_{max}|\psi_{sp} - \psi|}\right). \tag{7.7}$$

Assume that the vessel and its low-level controllers are able to track the r generated by the control law directly. If we also assume that the lateral speed of the vessel is negligible, the cross-track error when following a straight line can be given as:

$$\dot{e} = u\sin(\psi - \alpha_k), \tag{7.8}$$

where α_k is the orientation angle of the line. Setting $\tilde{\psi} = \psi_{sp} - \psi$, the Lyapunov function:

$$V = \frac{1}{2}e^2 + \frac{L}{2}\tilde{\psi}^2, \tag{7.9}$$

where $L > 0$ is some constant, can be used to show stability for the system under the given control law. The time derivative of the Lyapunov function is:

$$\dot{V} = eu\sin(\psi - \alpha_k) - \tilde{\psi}\frac{\Delta L}{\Delta^2 + e^2}u\sin(\psi - \alpha_k) - |\tilde{\psi}|K(\tilde{\psi})L, \tag{7.10}$$

where

$$K(\tilde{\psi}) = T_r\dot{r}_{max} + \sqrt{T_r^2\dot{r}_{max}^2 + 2\dot{r}_{max}|\tilde{\psi}|} \;\geq\; 2T_r\dot{r}_{max} \;\; \forall \;\; \tilde{\psi}. \tag{7.11}$$

The Dynamic Window path tracker is asymptotically stable in all areas where $\dot{V} < 0$. When the vessel travels towards or parallel to the line, i.e., $e\sin(\psi - \alpha_k) \leq 0$, this is fulfilled as long as $u < K(\tilde{\psi})\Delta$. For the vessel and controllers used in this thesis, this relation puts an upper limit on the vessel's surge speed of $u_{max} \sim 35\,m/s$.

When heading away from the line, i.e., $e\sin(\psi - \alpha_k) > 0$, we have asymptotic stability given that:

$$L > \frac{eu}{|\tilde{\psi}|\left(2T_r\dot{r}_{max} - \frac{u}{\Delta}\right)}. \tag{7.12}$$

Since the requirement is only necessary when vessel is heading away from the line, the minimum value of $|\tilde{\psi}|$ is limited by $|\psi_{sp} - \alpha_k|$, resulting in a stricter requirement for L:

$$L > g(e), \quad g(e) = \frac{eu}{\operatorname{atan2}(e,\ \Delta)\left(2T_r\dot{r}_{max} - \frac{u}{\Delta}\right)}. \tag{7.13}$$

If (7.13) holds, then (7.12) will also hold. One may ask what happens with the right hand of (7.13) as $|\tilde{\psi}| \to 0$, i.e., when $e \to 0$. The limit can be found using L'Hôpital's rule:

$$\lim_{e \to 0} g(e) = \frac{u\Delta}{\left(2T_r\dot{r}_{max} - \frac{u}{\Delta}\right)}. \tag{7.14}$$

As seen by plotting $g(e)$, this limit is the lowest point of $g(e)$, so by choosing a $L > g(e_{max})$, this L will be valid for all $|e| < e_{max}$. By choosing L arbitrarily large, the system is thus asymptotically stable for arbitrarily large cross-track errors e. As the choice of L is based on e, the analysis does not prove global asymptotic stability for the system, but this is of no practical concern, as the system will be asymptotically stable for all finite values of e given the right value of L. Since the system's convergence is also time-independent, the classification identified for the system is *uniformly locally asymptotically stable*, or ULAS.

When moving to series of connected lines, a piecewise linear path, this result no longer holds. The system will be ULAS along each of the lines, but this property says nothing about the tracking of the entire path. Having said this, exact tracking of the path is really not that important. The Dynamic Window algorithm will track the path as best it can, but deviating from the path will not increase the hazard of collision or violation of COLREGS, as the Dynamic Window algorithm also does its best to avoid these situations.

The generated path

Given that the control system is able to follow a generated path, the next question is whether the RRT algorithm is able to generate a path that leads the vessel to the goal.

It can in fact be proven that the RRT algorithm is probabilistically complete [29] given the right technical circumstances, that is, the probability of finding a path through the environment approaches 1 as time approaches infinity if a feasible path exists in the search space of the algorithm. Hence, given enough processing time, the RRT algorithm will find a path through an environment if one exists. There are still two practical problems however.

The first problem is that due to real-time demands, the RRT is only given a limited amount of time to find a path. At the end of the search, the algorithm may not have found a feasible path through the environment. This problem is solved by selecting the path that provides the best progress towards the goal at the lowest cost. This solution makes the RRT algorithm very robust, as it does not rely on being able to find a path in every iteration; it can still guide the vessel towards the goal.

The second problem can occur when the search space of the controller is too small to include any feasible paths to the goal. In this case, the RRT algorithm is reduced to a local algorithm, and as the local algorithms, it may be trapped in a local minimum in the environment. This situation may also occur as a result of limited processing time to find a path out of a local minimum. In these cases, convergence to the goal cannot be guaranteed. The path generated by the RRT provides a stable system however, as it will always do its best to reduce the distance to the goal, even though it may not be able to get all the way.

As previously mentioned, a bonus with the RRT algorithm is that it will always generate a path that our vessel is able to follow given an accurate vessel model. By choosing a vessel model that is less manoeuvrable than the actual vessel, an additional level of robustness is added to the system, as the generated paths will be more conservative. Having a very accurate model is thus not necessary, as long as the controlled vessel is equally or more manoeuvrable than the model.

Given an environment without considerable local minima, and enough processing time, the RRT algorithm will be able to guide the vessel to the goal. The rate of convergence provided by a path generated by the RRT algorithm is difficult to determine, but it will improve with the amount of time allowed for execution.

7.2.2 It gets worse

As seen in the previous section, global convergence to the goal cannot be guaranteed even in an environment consisting only of static obstacles. When dynamic obstacles such as other vessels are added to the equation, the situation gets worse.

The behaviour of the RRT algorithm is altered by the presence of dynamic obstacles, but its properties remain the same. Given that there exists a path through the environment which does not violate safety zones or COLREGS, the RRT is guaranteed to find a path given enough processing time and a large enough search area. The RRT may also find paths that violate safety and COLREGS, but these will only be used if there are no better alternatives.

The performance of the RRT algorithm in the presence of dynamic obstacles should be consistent with the performance expected for a static environment. The line-tracking abilities of the system can no longer be guaranteed however, because the Dynamic Window algorithm may deviate an arbitrary distance from the line to avoid dangerous situations, which may lead the vessel into local minima, making it unable to return to the path. This behaviour does not represent a major problem however, as the RRT algorithm on the next run will generate a path leading it out of the minimum. As shown in Figure 7.1, the Dynamic Window algorithm can also enter limit cycles given the right circumstances.

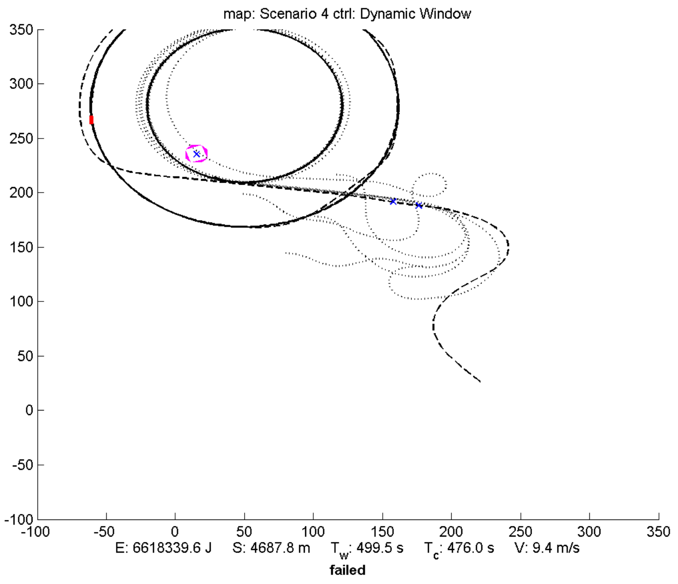


Figure 7.1: This simulation shows the vessel entering a limit cycle when trying to reach the goal in the upper left while avoiding five hostile vessels. The outer circle shows the path that the controlled vessel converges to, while the hostile vessels converge to the inner circle. The controlled vessel was in this case guided solely by the Dynamic Window algorithm. The RRT algorithm could have been able to get the vessel out of the limit cycle.

With correct estimates of the vessel dynamics, the Dynamic Window algorithm should be able to eliminate the possibility of collisions with static obstacles. When dynamic obstacles enter the equation this is no longer true however. An example is when another vessel deliberately tries to hit the controlled vessel. If the hostile vessel is more manoeuvrable than the controlled vessel, collision avoidance is generally impossible!

Another issue is the discontinuity happening when the RRT algorithm has generated a new path. If the new path does not coincide with the old path at the current location of the vessel, the vessel will have to manoeuvre to the new path. This discontinuity can give poor convergence rates towards the goal and in the worst case result in a limit cycle, preventing further progress towards the goal. Some thought has been put into the algorithm to make the possibility of the paths coinciding high (see Section 3.2.1) however, so this effect should not be a big problem.

7.2.3 Conclusion

As a system gets more complex, it also gets harder to analyse. For a simple linear system such as $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$ there are lots of tools for determining performance [12, 45]. In our case however, we have a nonlinear system with discrete events causing severe discontinuities, which makes determining stability properties nontrivial and very difficult.

The conclusion is that the system is likely to behave well, will likely have a good rate of convergence, will likely be stable, and it will likely avoid collisions. These properties are confirmed by the simulations and experiments in chapters 4 and 6.

Chapter 8

Discussion

Nearing the end of the thesis, it is time for an assessment of the results: What was achieved, how well, and what can be done to improve the developed collision avoidance system?

A number of extensions and modifications were made to base algorithms of the controller. Using the A* algorithm as a guide for the RRT algorithm turned out to give a great performance gain. In areas where the A* path is feasible and good for the vessel, the extension is able to improve the paths a great deal. This is especially important in applications with low processing power, or when the search space of the algorithm is very large.

The continuity extension also delivered good results. Without it, the switch occurring when a new RRT path has been generated almost makes the controller unusable. An additional bonus with the extension is that it tends to give the RRT algorithm a *hot-start*. By attempting to re-use parts of the old path, this area is explored at almost no cost, giving the RRT a good start.

The medial nodes on the other hand didn't have any noticeable positive effect, and even reduced the performance of the algorithm in some instances. This was a bit surprising, but is probably a result of the NEAREST_NEIGHBOUR() function in the RRT. This function runs in O(N) time, meaning that it gets slower the more nodes there are in the tree. Optimizing this function could yield a better result for the extension. One possible optimization includes partitioning the search-space to reduce the number of nodes to check when looking for the one that is closest to the candidate milestone.

The Dynamic Window algorithm also received some attention. The modifications done to this algorithm was more a necessity than a performance improvement though, as the original Dynamic Window had serious problems trying to predict the motions of a surface vessel. In [38], this was not observed because the vessel model had artificially high lateral damping coefficients. When lowering the lateral damping to more realistic levels, the Dynamic Window algorithm could no longer keep the vessel from colliding with objects in the environment. The improved Dynamic Window algorithm incorporates acceleration and lateral velocities into the original Dynamic Window algorithm. The quantitative improvement is difficult to determine, but examples of the difference are given in Section 4.3.

A big point has been made of the importance of COLREGS compatibility. The system developed to give COLREGS-compliant manoeuvres was included in both the local and global part of the algorithm to improve the chance of coinciding decisions in both layers. As seen in both the Matlab and HIL simulations, the vessel successfully followed the rules where necessary. These scenarios were constructed to give a certain reaction though, so they give no guarantee for whether the COLREGS system will work in all situations. This is generally a problem for the algorithmic approach used for the COLREGS system and the rest of the controller. Verification and guaranteeing performance is very difficult, and the only tool one really has to improve the confidence in a system is empirical testing.

Performance in the case of sensor failure is also an issue. How does the performance of the system degrade when certain information is wrong or unavailable? An answer to one such case was given in the full-scale experiments with the Viknes vessel. When parts of the AIS data received from the approaching vessel was wrong, the prediction of the controller was severely degraded. The COLREGS performance also suffered, and whether the vessel passed on the correct side in both cases because of COLREGS or because of *luck* is not known.

What is known is that a collision avoidance system is no better than the information about its environment. If it cannot see an obstacle, it cannot evade it. Because of the importance of complete and correct data, the sensor outfit should contain multiple sensors of different types, maximizing the probability of acquiring the necessary information to stay safe from obstacles in the environment.

One may ask how good the vessel model used in this thesis is, and how well it matches the dynamics of the actual Viknes vessel. This evaluation should in turn be an indicator of how valid the different simulations are. Such an assessment is both correct and wrong. If the goal of the simulations were to imitate the response of the Viknes to different environments as closely as possible, a relatively exact model would have been required. To verify the collision avoidance system on a more general basis however, the accuracy of the model would be of lesser concern, as long as the basic handling characteristics of the vessel were covered by the model.

The way the controller is built, with the Dynamic Window algorithm tracking a path generated by the RRT algorithm, also reduces the need for an exact model in the RRT algorithm, as long as it is less manoeuvrable than the target vessel. On the local side, the original Dynamic Window algorithm requires very little model information. The improved Dynamic Window algorithm requires a bit more however, relying on the damping parameters and acceleration capabilities of the vessel. Good prediction in this algorithm is important, and in a simulation this is not a problem, since the parameters of the simulated vessel is known. In a full-scale implementation however, they are seldom known accurately. This is compensated for by safety-regions defined around all obstacles, allowing some misprediction without resulting in dangerous situations.

In one of the full-scale experiments, the controlled vessel briefly grazed one of the illegal zones (virtual obstacles) in the environment. This was the result of tuning issues. The low-level controllers on the Viknes were a bit conservatively tuned, and the Dynamic Window algorithm expected the vessel to turn faster than it actually did. By correctly matching the maximum angular acceleration estimate of the vessel $\dot{r}_{max}(r)$ to the Viknes with these low-level controller settings, this could have been avoided.

For a collision avoidance system to be adopted and used commercially, it would have to show both predictable and correct behaviour in all situations. COLREGS compliance is one means of achieving predictable and correct behaviour, but only for a limited amount of cases, and how can we be sure that the COLREGS system itself is predictable? Guaranteeing performance is, as already mentioned, very difficult, because of the complexity of the control system.

Performance for physical systems is generally given in terms of statistically determined properties such as the *mean time between failure* (MTBF), which can be determined by a mixture of Monte Carlo simulations and extensive real-world testing. If a collision avoidance system is determined to have a sufficiently high statistical performance, it can be considered safe.

Relying on statistics may seem crude, but as we know all to well, humans are not perfect either. A collision avoidance system is designed to assist or replace human decision-making, and if it can provide a significantly higher statistical performance than the collision avoidance performed by the average human, the human should be replaced. Being able to show this level of performance may also help remove the legal barriers concerning autonomous vehicles.

Chapter 9

Conclusion

A COLREGS-compliant collision avoidance system for an USV has been developed. Under simulations, the system behaved well, avoiding static obstacles with ease, and performing COLREGS-compliant manoeuvres to avoid dynamic obstacles.

Good performance was also seen in the full-scale experiments. Because of sensor problems, where an approaching vessel was reporting false state data, the margins during the experiments were reduced from what was seen in the simulations, but this can be seen as a test of robustness, as the collision avoidance system still behaved well and managed to avoid collisions.

A collision avoidance system stands and falls on the information made available to it, and the available sensor suite on the controlled vehicle should reflect this fact. The limiting factors to the performance of a collision avoidance system are the sensors, processing power and manoeuvrability of the vessel; in that order, and all need to be maximized for the optimal performance.

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] Maritime robotics. http://www.maritimerobotics.com/.

[4] SPAWAR. http://www.spawar.navy.mil/robots/surface/usv. Accessed 2007.12.10.

[5] Viknes. http://www.viknes.no/. Accessed 2007.12.09.

[6] 5G-Marine-Systems. http://www.5gmarine.com/. Accessed 2008.06.01.

[7] M. R. Benjamin, J. J. Leonard, J. A. Curcio, and P. M. Newman. A method for protocol-based collision avoidance between autonomous marine surface craft. *Journal of Field Robotics*, pages 333–346, 2006.

[8] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. *IEEE*, 1990.

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

[10] M. Breivik and T. I. Fossen. Guidance laws for planar motion control. *submitted to the 47th IEEE CDC, Cancun, Mexico*, 2008.

[11] A. Chakravarthy and D. Ghose. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man. and Cybernetics - Part A: Systems and Humans*, 1998.

[12] C. Chen. *Linear System Theory and Design.* Oxford University Press, Inc, 1999.

[13] J. 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.

[14] Space Daily. FAA clears global hawk for routine operation in US national airspace. `http://www.spacedaily.com/news/uav-03zl.html`. Accessed 2008.05.29, 2003.

[15] Industrial Control Design. `http://www.icd.no/`. Accessed 2008.05.22.

[16] O. Egeland and J. T. Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics AS, Trondheim, Norway, 2003.

[17] Ø. Engelhardtsen. 3D UAV collision avoidance. Master's thesis, Norwegian University of Science and Technology (NTNU), 2007.

[18] J. A. Farrell and M. Barth. *The Global Positioning System & Inertial Navigation*. McGraw-Hill Professional, 1999.

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

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

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

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

[23] 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.

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

[25] Northrop Grumman. Integrated systems. `http://www.is.northropgrumman.com/`. Accessed 2008.05.14.

[26] T. Holzhüter. Operating experience with a high precision track controller for commercial ships. *Control Engineering Practice*, 4:343–350, 1996.

[27] T. Holzhüter. LQG approach for the high precision track control of ships. *IEE Proceedings Control Theory and Applications*, 144:121–127, 1997.

[28] T. A. Johansen, A. J. Sørensen, O. J. Nordahl, O. Mo, and T. I. Fossen. Experiences from hardware-in-the-loop (HIL) testing of dynamic positioning and power management systems. *OSV Singapore*, 2007.

[29] J. J. Kuffner Jr. and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. *Proceedings of the ICRA'00*, pages 995–1001, 2000.

[30] H. K. Khalil. *Nonlinear Systems.* Prentice Hall, 2001.

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

[32] 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.

[33] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. *Proceedings of the ICRA'91*, pages 1398–1404, 1991.

[34] Kystverket. Automatisk Identifikasjons System - AIS. `http://www.kystverket.no/?did=9140988`. Accessed 2007.12.11.

[35] J. Larson, M. Bruch, and J. Ebken. Autonomous navigation and obstacle avoidance for unmanned surface vehicles. *Proceedings of the International Society for Optical Engineering*, 2006.

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

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

[38] Ø. Loe. Collision avoidance concepts for marine surface craft. Project report, Norwegian University of Science and Technology (NTNU), 2007.

[39] 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.

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

[41] B. Pillich and G. Büttgenbach. ECDIS - the intelligent heart of the hazard and collision avoidance system. *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, pages 1116–1119, 2001.

[42] 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.

[43] R. Skjetne and O. Egeland. Hardware-in-the-loop simulation for testing of DP vessels. *OSV Singapore*, 2005.

[44] R. Skjetne, Ø. N. Smogeli, and T. I. Fossen. A nonlinear ship manoeuvering model: Identification and adaptive control with experiments for a model ship. *Modeling, identification and control*, 25:3–27, 2004.

[45] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control.* Wiley, 2005.

[46] K. J. Åström and T. Hägglund. *Advanced PID Control.* ISA - The Instrumentation, Systems, and Automation Society, 2005.

[47] C. S. Tan, R. Sutton, and J. Chudley. An incremental stochastic motion planning technique for autonomous underwater vehicles. *Proceedings of IFAC CAMS'04*, pages 483–488, 2004.

[48] S. Trudslev. Path finding in C#. `http://www.codeproject.com/KB/recipes/csharppathfind.aspx`. Accessed 2007.12.10.

[49] Defense Update. Interceptor - unmanned surface vessel unveiled. `http://www.defense-update.com/products/i/interceptor.htm`. Accessed 2008.05.14.

[50] I. Škrjanc and G. 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.

[51] G. Welch and G. Bishop. An introduction to the kalman filter. `http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html`. Accessed 2008.05.29, 2004.

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

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

[54] Wikipedia. DARPA Grand Challenge — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/DARPA_Grand_Challenge`. Accessed 2007.12.16.

[55] Wikipedia. Electronic chart display and information system — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Electronic_Chart_Display_and_Information_System`. Accessed 2008.06.02.

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

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

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

[59] Wikipedia. International regulations for preventing collisions at sea — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/International_Regulations_for_Preventing_Collisions_at_Sea`. Accessed 2008.05.31.

[60] Wikipedia. Voronoi diagram — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Voronoi_diagram`. Accessed 2008.05.28.