Guidance System for Autonomous Surface Vehicles

Thomas Stenersen

Master of Science in Cybernetics and Robotics

Submission date:	June 2015
Supervisor:	Kristin Ytterstad Pettersen, ITK
Co-supervisor:	Øystein Engelhardtsen, DNV GL

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Assuming available sensor data, design a guidance system for an Autonomous Surface Vehicle (ASV) capable of safe navigation in various environments. The system shall adhere to the International International Regulations for Avoiding Collisions at Sea (COLREGs).

1. Outline a path planning and collision avoidance strategy
2. Simulate the system
3. Implement the system in ROS

To my fiancée, for her patience during the countless hours I spent working on this thesis. . .

Abstract

There has been a rapid growth in autonomous technology in several fields the past decade. One of these is autonomous navigation, where the DARPA Grand Challenges has been a major research catalyst. Today, the Centre of Excellence "Centre for Autonomous Marine Operations and Systems (AMOS)" in cooperation with the maritime industry seeks to develop autonomous technology for marine applications.

A robust collision avoidance system is crucial for an Autonomous Surface Vessel (ASV). In order to operate at sea, near other traffic, it also needs to adhere to the "rules of the road", the International Regulations for Avoiding Collisions at Sea (COLREGs). In this thesis, a COLREGs compliant guidance, navigation and control (GNC) system has been developed with the Velocity Obstacle (VO) method as the basis for collision avoidance. To validate the performance of the GNC system, a general nonlinear 3-DOF surface vessel simulator has been implemented. It features simple models for slow and fast-varying disturbances (*e.g.* waves, wind and current). The system has been implemented in the Robotic Operating System (ROS) framework.

Three main scenarios has been designed to test the guidance system in all main COLREGs scenarios: overtaking, head-on and crossing. Each scenario requires multiple COLREGs compliant maneuvers. Several special cases are also examined and discussed.

The VO method performs very well in the simulated scenarios and its avoidance maneuvers complies with all main COLREGs requirements. A thorough discussion highlights both advantages and disadvantages with method and suggests actions to mitigate less ideal behavior experienced in some special cases. The thesis also features a rich discussion on further development and improvements to the system.

Samandrag

Det siste århundret har vi sett ein enorm vekst innan autonom teknologi i ulike felt. Eit av desse felta er autonom navigasjon, der DARPAs «Grand Challenge»-ar har vore ein viktig forskingskatalysator. I dag ynskjer senter for framifrå forsking, «Senter for Autonome Marine Operasjonar og System (AMOS)» i samarbeid med den maritime industrien å utvikle autonom teknologi for marine applikasjonar.

Eit robust kollisjonsomgåingssystem er ein svært viktig del av eit autonomt overflateskip. For å nyttast til sjøs, nær anna trafikk, må systemet òg lyde dei internasjonale reglane for kollisjonsomgåing til sjøs (COLREGs). I denne avhandlinga er eit styrings-, navigasjons- og kontrollsystem som lyder COLREGs utvikla, med «Velocity Obstacle (VO)»-algoritmen som basis for kollisjonsomgåing. For å kunne validere funksjonaliteten til systemet har ein generell simulator for fartøy med tre fridomsgradar òg vorte utvikla. Denne inneheld enkle modellar for sakte- og hurtigvarierande forstyrringar (t.d. bølgjer, vind og straum). Systemet er implementert i «Robot Operating System (ROS)»-rammeverket.

Tre hovudscenario har vorte designa for å teste styringssystemet i dei viktigaste COLREGs-situasjonane: overtaking, front-mot-front og kryssing. Kvart scenario krev fleire COLREGs-samsvarande manøvrar. I tillegg, er fleire spesialtilfelle undersøkt og diskutert.

VO-metoden yter særs bra i dei simulerte scenaria og alle omgåingsmanøvra samsvarar med COLREGs-reglementet. Ein djuptgåande diskusjon belyser både fordelar og ulemper med metoden og foreslår tiltak for å minske mindre ideell oppførsel erfart i einskilde tilfelle. Avhandlinga inneheld òg ein rik diskusjon kring vidare utvikling og forbetringar av systemet.

Preface

This thesis is written as a compulsory part of the Master's degree in Engineering Cybernetics at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The thesis has been written in cooperation with DNV GL.

Acknowledgements

I would like to thank my supervisors Kristin Y. Pettersen (NTNU) and Øystein Engelhardtsen (DNV GL). I would also like to thank Bjørn-Olav, Stine and Sveinung whom I shared office space with the past year. A special thanks goes out to Bjørn-Olav for all the interesting discussions regarding collision avoidance strategies.

Thomas Stenersen,
Trondheim, June 2015

Contents

List of Figures

List of Tables

List of Acronyms

AIS Automatic Identification System

AMCL Adaptive Monte-Carlo Localization

AMN Autonomous Maritime Navigation

ASV Autonomous Surface Vehicle

COLREGs International Regulations for Avoiding Collisions at Sea

CPA Closest Point of Approach

DARPA Defense Advanced Research Projects Agency

DW Dynamic Window

EKF Extended Kalman Filter

ENU East, North, Up

GNC Guidance, Navigation and Control

GPS Global Positioning System

HIL Hardware-in-the-Loop

ILOS Integral Line of Sight

IMO International Maritime Organization

IMU Inertial Measurement Unit

LOS	Line of Sight
LQG	Linear Quadratic Gaussian
MVFF	Modified Virtual Force Field
NED	North, East, Down
POA	Projected Obstacle Area
REP	ROS Enhancement Proposal
ROS	Robotic Operating System
RRT	Rapidly-exploring Random Tree
UKF	Unscented Kalman Filter
USV	Unmanned Surface Vehicle
VFF	Virtual Force Field
VFH	Vector Field Histogram
VO	Velocity Obstacle
WMR	Wheeled Mobile Robot

Notation

$\boldsymbol{\eta}$	Position and orientation vector
$\boldsymbol{\nu}$	Body-fixed linear and angular velocity vector
$\boldsymbol{\tau}$	Vector of generalized forces
$\mathbf{M}$	Mass matrix
$\mathbf{C}(\boldsymbol{\nu})$	Coriolis matrix
$\mathbf{D}(\boldsymbol{\nu})$	Damping matrix
ω_0	Dominating wave frequency
λ	Wave damping coefficient
σ	Wave intensity coefficient
u_d	Surge speed set-point
ψ_d	Heading set-point
K_p	Proportional gain
K_i	Integral gain
K_d	Derivative gain
χ_p	Path-tangential angle
χ_r	Velocity-path relative angle
Δ	Lookahead distance
$e(t)$	Cross-track error
$s(t)$	Along-track error
R_a	Radius of acceptance
$\mathcal{C}$	Configuration space

$\mathcal{CO}$	Part of configuration space *with* obstacles
$\mathcal{C}_{\text{free}}$	Part of configuration space *without* obstacles
t_{CPA}	Time to closest point of approach
d_{CPA}	Distance to closest point of approach
$VO_{A\|B}$	The Velocity Obstacle of B in the velocity space of A
$\mathbf{p}_{AB}$	The position of B relative to A
$\mathbf{v}_{AB}$	The velocity of B relative to A
$\mathcal{A} \oplus \mathcal{B}$	The Minkowski sum of the sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{V}_1$	Set of velocities left of the VO cone
$\mathcal{V}_2$	Set of velocities right of the VO cone
$\mathcal{V}_3$	Set of velocities below of the VO cone

Chapter 1

Introduction

1.1 Motivation

The past century, automation has revolutionized numerous industries by making tasks more safe and cost-effective. Not only has it forever changed pre-existing industries, but also paved the way for several new. One example is the furniture producer Ekornes, that managed to keep their production line in Norway despite high labor cost, by automating much of their production – without any layoffs (Klingenberg, 2014).

In recent years, the intelligence of such systems has increased immensely. A transition from *automated* to *autonomous* systems has taken place. Where the automated system is specialized in one task, the autonomous system is a self-governing system capable of completing loosely defined goals with intelligent reasoning. Autonomy is found in a multitude of fields, from automated journalistic robots (Narrative Science, 2014), used by many of the largest newspapers today, to self-driving cars (Fig. 1.1a) (Google Inc., 2014) and general-purpose mobile robots (Marder-Eppstein, Berger, Foote, Gerkey, & Konolige, 2010).

Today, both MARINTEK through the MUNIN project (Dragland, 2014) and DNV GL (Flæten, 2014) envisions the use of Autonomous Surface Vehicles (ASVs) for transport at sea (Fig. 1.1b). The onshore transportation networks are reaching their capacity, increasing the demand for alternatives. Due to the harsh landscape, expanding road or rail networks in Norway is both difficult and expensive. One promising solution is therefore "short sea shipping" (Adams, 2015). Normally, the cost of maintenance and crew makes this solution less viable. To

(a) Google's self-driving car. (b) DNV GL's Revolt

Figure 1.1: Two autonomous vehicles (images courtesy of Google and DNV GL).

overcome this, DNV GL suggest using an unmanned, fully electrical powered vessel with a lower cruising speed of 6 knots. This design requires low maintenance and is very cost-effective. The concept vessel is named “ReVolt”, and this thesis is a part of a prototype that is currently being developed by DNV GL.

In ASV development, path planning and collision avoidance remains largely unsolved challenges. Previous studies have to a great extent been limited to Wheeled Mobile Robot (WMR) platforms. Although path planning and collision avoidance algorithms for mobile robots have similar difficulties as their maritime counterparts, there are several important factors that differs. Firstly, WMRs are for the most part either holonomic or have first-order nonholonomic constraints, where ocean vehicles for the most part are underactuated with second-order nonholonomic constraints. Secondly, when operating at sea, one must adhere to the International Regulations for Avoiding Collisions at Sea (COLREGs).

In this thesis, a complete COLREGs compliant Guidance, Navigation and Control (GNC) system, with the Velocity Obstacle (VO) method for collision avoidance, is implemented and suggested as a versatile and robust basis for an ASV control system architecture. The VO method is an intuitive method for collision avoidance in the presence of moving obstacles, and imitates in many ways the mindset of an experienced helmsman. The system has been developed using the Robotic Operating System (ROS) framework.

1.2 Assumptions

The following assumptions has been made:

- All other vessels are motor-driven. For example, situations involving interaction with sailboats, where the motor-driven vessel will always be the give-way vessel, are not considered.

- Sufficient sensor data is always available (no dead-reckoning) and the system is observable based on this sensor data.
- Information regarding other vessels is available, *e.g.*, through sensors such as radar, stereo-cameras, etc. This information is enough to determine pose and velocity of the other vessels.

1.3 Contribution

The main contributions of this thesis are:

- A thorough review of the existing literature on the subject.
- An in depth analysis of the Velocity Obstacle (VO) algorithm as a collision avoidance strategy for an ASV operating under COLREGs requirements.
- An implementation of the algorithm together with the necessary framework ready for full-scale experiments.
- Algorithm performance validation through extensive simulations together with a rich discussion and recommendations for future development and improvements.

1.4 Thesis Structure

Chapter 2 provides a review of the most prominent research on guidance systems for maritime applications in recent years together with research from the mobile robotics field. Chapter 3 presents the necessary theoretical background and Chapter 4 explains the implementation of the system. In Chapter 5, the results from a selected set of simulated scenarios are given. A discussion of the challenges and possible extensions to the system is given in Chapter 6 and 7. Finally, a conclusion is given in Chapter 8.

There are three chapters in the Appendix. First, a general introduction to ROS, secondly system implementation details and finally a collection of source code examples.

Chapter 2

Literature Review

Since the latter part of the 20$^{\text{th}}$ century, *planning algorithms* has been an increasingly popular research area. Despite decades of research, no unified approach or solution to the planning problem has been found. For a comprehensive collection of general planning algorithms, the reader is referred to LaValle (2006).

This chapter reviews several of the main contributions to autonomous maritime navigation research in recent time.

2.1 Overview Papers

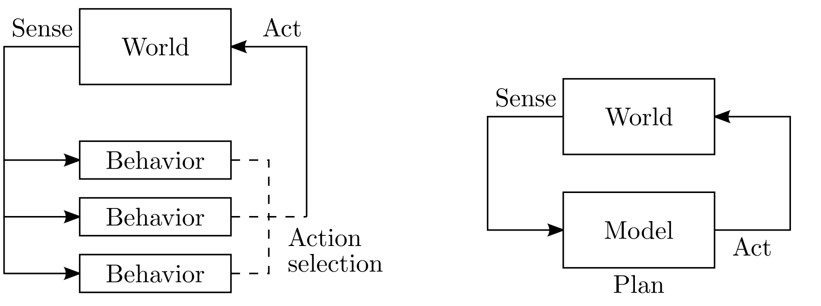
Unfortunately, the amount of overview or review papers on the topic of maritime path planning and collision avoidance systems is next to none. The studies most often referred to are presented in this section.

Campbell, Naeem, and Irwin (2012) review the state of research on ASVs in 2012. The article focuses on GNC and motion planning aspects and highlight development needs. It lists Unmanned Surface Vehicle (USV) prototypes for research (Majohr, Buch, & Korte, 2000; Bibuli, Bruzzone, Caccia, Indiveri, & Zizzari, 2008; Naeem, Sutton, & Chudley, 2006) and military applications (Corfield & Young, 2006; Rafael Advanced Defense Systems Ltd., 2010; Larson, Bruch, & Ebken, 2006). The authors continue with an analysis of the most common control and guidance laws used, together with obstacle detection and avoidance architecture and map representations. The paper considers only one type of local methods, potential field methods. Potential field methods (Khatib, 1986) has been proposed for USV collision avoidance in the past (Lee, Kwon, & Joh, 2004), however they are associated with several inherent limitations such as oscillation problems near obstacles and in narrow passages (Koren & Borenstein, 1991). Evolutionary algorithms and heuristic A*-like methods are suggested for path planning purposes. Finally, a review of the main COLREGs rules and how to apply COLREGs for a multiple USVs in cooperation is given. The article gives an overall good introduction to the challenges faced when developing an ASV, however it lacks depth on collision avoidance methods.

Tam, Bucknall, and Greig (2009) review the development of collision avoidance techniques and path planning for ships in a historical context. The article provides a good and comprehensive historical background on the subject.

2.2 Planning Algorithms for Ocean Vehicles

Benjamin, Leonard, Curcio, and Newman (2006) propose a behavior-based method for avoiding collisions and adhering to COLREGs (Fig. 2.1). The behavior-based methodology takes away the need for a single complex world model of which decisions has to be made from and replaces it with a discrete decision space. The



(a) Behavior-based control scheme (b) Conventional control scheme

Figure 2.1: Differences between behavior-based control and conventional control. Behavior-based control compose vehicle behavior into distinct modules that are developed and operate largely in isolation, and coordinated through an action selection mechanism (Benjamin, Leonard, Curcio, & Newman, 2006).

primary difficulty associated with behavior-based control is how to ensure the action selected is in the overall best interest of the robot or vehicle. The two main approaches to this is to either select one of the actions and apply it or to apply the sum of two or more actions (averaging). The authors suggest using interval programming (an optimization technique) to find the best weighted sum of actions. The system has been tested on a physical platform and shows promising results. The behavior-based method significantly reduce the level of complexity of the behavioral analysis, since the system acts as a finite state machine. *I.e.*, the system is not able to take any actions other than the already predefined ones. However, guaranteeing stability and COLREGs compliance may prove to be difficult, if not impossible. As such, the method may still be considered experimental and needs to be more rigorously tested. For example, the behavior at the boundary between COLREGs rules or in situations where the best course of action may change as the situation progresses.

Larson et al. (2006, 2007) propose a hybrid approach with an A*-based method for global navigation and a behavior-based common world model as the reactive (or local) planner. The global path planner utilizes the VO method (Fiorini & Shiller, 1993) to determine safe velocity ranges for avoiding moving obstacles. If changing the velocity alone does not guarantee collision avoidance, the path planner changes the path by using Projected Obstacle Areas (POAs) (Fig. 2.2), *i.e.*, an estimate of the region an obstacle is likely to occupy in the future. The

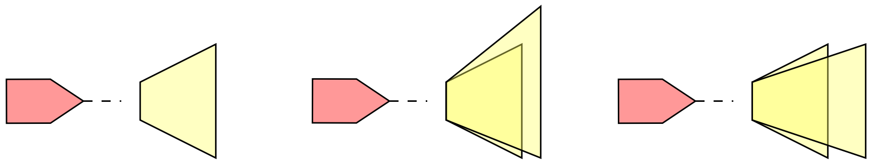
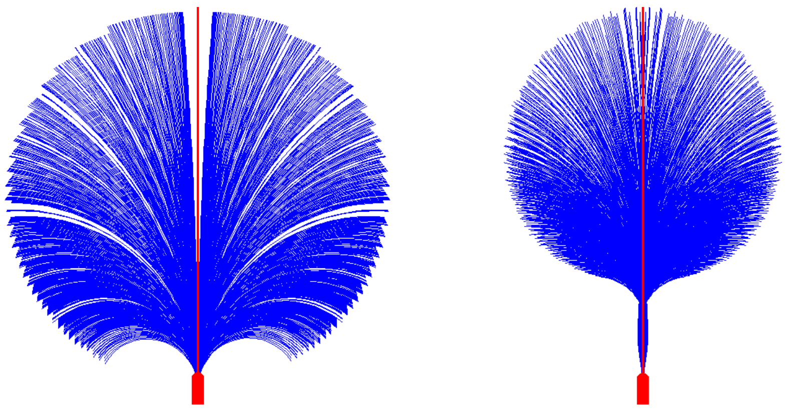

(a) Standard POA (b) Starboard action (c) Increased surge velocity

Figure 2.2: Projected Obstacle Areas (POAs). The POAs define regions in the obstacle map dynamic obstacles are likely to occupy during the next planning cycle.

local world model is a fusion of all near-field sensors and individual behaviors vote on specific navigation solutions within the local model. This approach has its roots from the Morphin planning algorithm (Simmons & Henriksen, 1996). The system calculates an array of arcs and each behavior votes on these. For the obstacle avoidance behavior, each arc is given a score based on the distance the vehicle can travel before encountering an obstacle. Clearly, this is similar to the Dynamic Window (DW) approach (Fox, Burgard, & Thrun, 1997). The system is deployed on a SEADOO Challenger 2000 sport boat, but the authors does not present any results from sea trials or simulated scenarios. In addition, COLREGs compliance is only implemented as a part of the path planner and not the reactive planner, however a sufficiently fast replan rate for the path planner might rectify this.

In Loe (2007) several popular methods are compared and provide a basis for the approach used in Loe (2008). A Matlab simulation environment was implemented in order to compare algorithm performance, and based on the results a hybrid approach with A*-guided Rapidly-exploring Random Trees (RRTs) (LaValle, 1998) for global path planning and the dynamic window algorithm for local collision avoidance was recommended. Loe (2008) propose modifications to the methods to incorporate proper COLREGs behavior and improve their overall performance. Most notably, more of the vessel dynamics are considered in the DW algorithm – in particular sway motion and acceleration constraints (Fig. 2.3). Thus changing the method from exploring circular arcs to more dynamically feasible trajectories. The system was tested in Trondheimsfjorden in cooperation with Maritime



(a) Original (b) Modified

Figure 2.3: Modified DWA algorithm. Incorporating lateral velocities and accelerations allows for more accurate trajectories (Loe, 2008).

Robotics yielding promising results. Because of the random nature of RRTs, the paths generated are unpredictable and sub-optimal. The method is originally intended for use with high degree of freedom systems with nonholonomic constraints. To increase optimality and reduce unpredictability, an A* search is used to guide the RRTs. One may argue that it may be more effective to include vehicle dynamics in the A* search directly rather than use two separate methods.

Naeem, Xu, Sutton, and Tiano (2008) describe the system architecture for the *Springer USV*, an USV designed for environmental monitoring and pollutant tracking. The paper describes both hardware and software architecture, including sensor suite, system identification and GNC. A prediction error method was used for system identification with good results. The navigation system uses a fuzzy logic adaptive Kalman filter (FLA Kalman filter) as a fault-tolerant multi-sensor navigation strategy (Fig. 2.4). The guidance and control system consists of

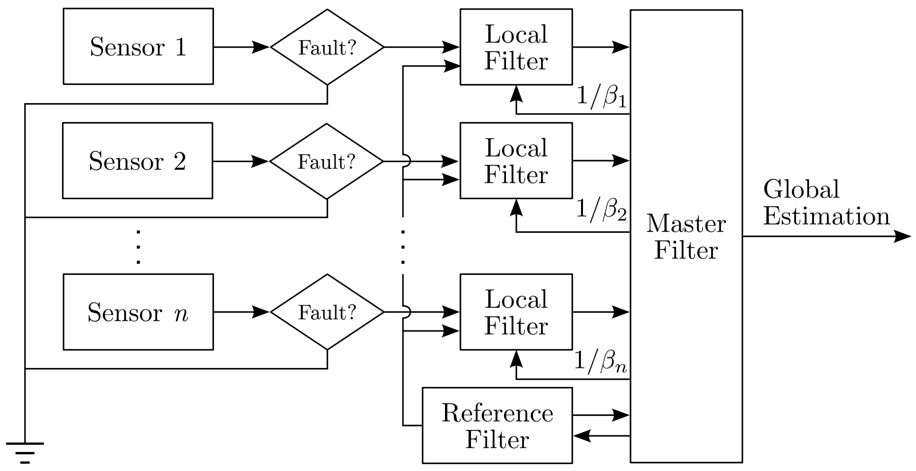


Figure 2.4: Fault-tolerant sensor fusion. If the sensor is free of faults, its signal is passed on to a local filter that compares its estimate against a reference filter. The local estimation sent to a master filter with a weight of $1/\beta_i$ for sensor i.

a Line of Sight (LOS) guidance algorithm in combination with a Linear Quadratic Gaussian (LQG) controller. The USV does not, however, feature any path planning or obstacle avoidance systems. Naeem, Irwin, and Yang (2012) explore the development of a collision avoidance scheme for the USV based on A* path planning and a manual bias for COLREGs compliant avoidance maneuvers. The article provides simulations of the system performing relatively simple avoidance maneuvers, *e.g.*, avoiding static obstacles and simple COLREGs maneuvers.

The Autonomous Maritime Navigation (AMN) project (Elkins, Sellers, & Monach, 2010) is one of the most mature autonomous marine system in existence today. Elkins et al. (2010) describe the system as a whole; its sensor suite, data fusion, real-time control system, communications and more. Over the course of the project three USVs was developed (Fig. 2.5). The article presents the results from several full-scale experiments with in depth analysis and an extensive discussion. Kuwata, Wolf, Zarzhitsky, and Huntsberger (2011, 2014) present the VO approach for moving hazard avoidance (Sec. 3.6), the collision avoidance strategy used in the AMN project. They argue that other methods such as fuzzy logic, evolutionary algorithms, interval programming, and 2D grid maps does not scale well to multiple traffic boats and multiple COLREGs rules. Because of this, these

Figure 2.5: Powervent USV prototype, the second of three USV prototypes used in the AMN project (image courtesy of U.S. Navy).

methods are ill suited for robotic platforms with hard real-time demands. The VO method on the other hand scales linearly with the number of obstacles. Furthermore, due to the nature of the velocity obstacle approach, augmenting the collision avoidance system to comply with the main COLREGs rules is relatively straight forward (Sec. 4.4) as the VO already has information on which side of the hazard the USV will pass (the USV will either pass on the left, right or move away from the hazard (Sec. 3.6.2)). Full scale experimental results using a USV, a 12m traffic boat and two 7 m RHIBs are presented. Their system uses two pairs of stereo cameras to estimate the position and velocity of the traffic boats (Huntsberger, Aghazarian, Howard, & Trotz, 2011). In particular, two scenarios are presented: "Head-On and Crossing" and "Overtake, Head-On and Crossing". These are both relatively complex scenarios that requires applying multiple COLREGs maneuvers. In these scenarios, the system behaves very well and shows promising results for the VO algorithm for collision avoidance.

Švec et al. (2013, 2014) introduce a dynamics aware (*i.e.*, system-identified, nonlinear USV model) COLREGs compliant planning algorithm for target following. Furthermore, the algorithm is capable of predicting the future motion of obstacles using worst-case and probabilistic predictive motion models. The vessel utilized in the paper is the WAM-V USV14, with a differential thrust system. The equations of motion are given in a standard 3-DOF form (Fossen, 2011). By applying a control action $\mathbf{u}_{c,d,k} \in \mathcal{U}_{c,d}(\mathbf{s}_j) \subseteq \mathcal{U}_c(\mathbf{x}_j)$ from a discretized set of dynamically feasible control actions $\mathcal{U}_{c,d}$, multiple trajectories may be generated. A discrete control action primitive $u_{c,d,k} \in \mathcal{U}_{c,d}$ is represented as a sequence of USV poses $\{\boldsymbol{\eta}_i\}_{i=1}^{L_k} \in \mathcal{X}_\eta \times \mathcal{T}$. An A* search is used to find the "best" control action, *e.g.*, the one that makes most progress towards the goal. By forward simulating the control actions (using the dynamic model of the system) and checking for collisions, a set $\mathcal{U}_{c,obs}$ is found, defining the obstacles in the control space. This is similar to the Generalized VO scheme (Wilkie, van den Berg, & Manocha, 2009). When the collision free control space has been determined, COLREGs compliant maneuvers are found by examining the relative bearing, heading and position between the USV and the obstacle, comparable to the method used by Kuwata et al. (2011, 2014). The authors present results from on-water trials showing good behavior in both COLREGs and target following scenarios. Švec et al. (2014) also include an excellent literature review on technology applied to USV platforms.

Savvaris, Oh, and Tsourdos (2014) present the C-Enduro, a USV designed to operate at sea for up to three months at a time. The USV employs an algorithm similar to the collision cone approach (Chakravarthy & Ghose, 1998) or velocity obstacle (Fiorini & Shiller, 1998) for reactive collision avoidance, using the relative velocity between the USV and other ships to determine safe velocities. The system determines COLREGs rules based on which quadrant an obstacle ship is in the body-fixed coordinate system of the USV. The authors only present simulated data to verify its performance in minimal scenarios.

Lee et al. (2004) propose the Modified Virtual Force Field (MVFF) method as a modification of the classical Virtual Force Field (VFF) method (Borenstein & Koren, 1989). The MVFF method operates in either a "track-keeping" or "collision avoidance" mode. The objective of the first mode is to bring the vehicle to a desired track, presuming it has gone astray from this track. The second mode handles static and dynamic obstacles whilst adhering to COLREGs. Two forces $\vec{F}_a = \alpha\vec{e}_a$ and $\vec{F}_r = \beta\vec{e}_p$ are defined. They act as an attractive force towards the goal and a repulsive force from an obstacle respectively, where $\vec{e}_a$, $\vec{e}_r$ are unit vectors in the two directions and α, β are parameters determined using a set of fuzzy rules. Several simulated results are given with promising results, however all the simulations only show single obstacle avoidance. Koren and Borenstein (1991) demonstrated the limitations of potential field methods and the VFF method in particular, but Lee et al. (2004) does not address any of these difficulties. It is likely that the method would suffer from oscillations in more complex multi-obstacle environments.

There has been several other studies on using fuzzy logic or machine learning techniques for collision avoidance, *e.g.*, Perera, Carvalho, and Soares (2009), Zhang et al. (2014), but none has provided more than simulated results in minimal scenarios.

2.3 Other Planning Algorithms

Planning algorithms for maritime applications accounts for only a fraction of the total research in the field. Most research today is focused on WMRs and car-like robots. In many cases, this research is almost directly applicable to ocean vehicles. An excerpt of the available literature is presented here.

Rodríguez-Seda, Tang, Spong, and Stipanović (2014) developed a nonlinear controller for trajectory tracking and dynamical collision avoidance. The controller is based on an input-output linearizing feedback controller where system has the dynamics of a unicycle (two drive-wheels WMR). It is split in two, with one part for trajectory tracking and one part for collision avoidance. The article is rigorous, with Lyapunov-based proofs for stability of the methods, however due to the assumptions made about the system, the methods may be less applicable to surface vessels. In particular, the decoupling of surge force and yaw moment is not valid for underactuated surface vessels. Furthermore, only stability for the yaw angle is proven, not *asymptotic* stability, rendering the yaw angle uncontrollable. This may cause challenges in an attempt to incorporate COLREGs compliance into the method. In addition, the collision avoidance control law is based on "avoidance functions" that are similar to potential field methods (Khatib, 1986), known for their fundamental challenges (Koren & Borenstein, 1991).

The Defense Advanced Research Projects Agency (DARPA) Grand Challenges (Wikipedia, 2015b) has been catalysts for several new and promising autonomous systems. Montemerlo et al. (2008) present the autonomous robotic vehicle "Junior" that participated in the DARPA Urban Challenge. The article gives an overview of the whole autonomous system, including software architecture, environment detection, localization and navigation. Its navigation system consists of a road network navigation module and a free-form navigation module, the latter being most applicable to maritime navigation. The free-form planner, named the hybrid-state A* search, is described in detail in Dolgov, Thrun, Montemerlo, and Diebel (2010). As the name suggests, it is based on the very popular A* search (Hart, Nilsson, & Raphael, 1968), but is modified to better suit the nonholonomic nature of a car. By using the vehicle dynamics to expand viable nodes, the algorithm ensures path feasibility at the cost of completeness. During the initial search, only three control actions are considered for each node expansion: left turn, right turn and straight ahead. Later, a gradient-based smoothing technique is used to optimize the path. The authors also describe several clever heuristic methods to improve the search speed and path quality. One of the techniques is to use Voronoi-based potential fields to recognize environmental structures and mitigate "contour hugging" (Nord, 2010; Stenersen, 2014), a phenomena caused by the fact that the shortest path in an environment with obstacles tend to lie along the boundary of one or more of these obstacles. The algorithm performs very well in several scenarios and may be very well suited as a global planner for an ASV (Stenersen, 2014).

Marder-Eppstein et al. (2010) present a system for robust autonomous navigation in an indoor office environment. The paper describes all the necessary methods required for such a system, including local and global planners, a Voxel-based 3D mapping algorithm for modeling unknown space as well as an open-source implementation of the whole system together with a simulation environment. The framework used is the now well-known Robotic Operating System (Open Source Robotics Foundation, 2014). The system uses A* as a global guide for a local DW algorithm. The A* search has long been the probably most popular method for path planning, but it has some critical drawbacks making it less suitable for path planning for ocean vehicles. One of which is its tendency to produce paths containing 90° turns, rendering paths impossible for an ocean vehicle to follow. In practice, it is common to apply some sort of smoothing algorithm to the paths afterwards to rectify this. To mitigate "contour hugging", all obstacles are expanded by the safety radius of the robot. This is not ideal when navigating at sea, where it is preferable to keep good distance from the shore and a too large safety radius may render normally feasible paths unfeasible. The Voronoi-based guiding heuristics (Dolgov et al., 2010) may solve this efficiently (Stenersen, 2014).

Chapter 3

Theoretical Background

3.1 Surface Vessel Modeling

The marine craft equations of motion can be written on a general vectorial form (Fossen, 2011)

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{J}_\Theta(\boldsymbol{\eta})\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{g}(\boldsymbol{\eta}) + \mathbf{g}_0 &= \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{ext}}, \end{aligned} \tag{3.1}$$

Using the notation of Fossen (2011):

$$\begin{aligned} \mathbf{v}^e_{b/n} &= \text{linear velocity of the point } o_b \text{ with respect to } \{n\} \text{ expressed in } \{e\} \\ \boldsymbol{\omega}^b_{n/e} &= \text{angular velocity of } \{n\} \text{ with respect to } \{e\} \text{ expressed in } \{b\} \\ \mathbf{f}^n_b &= \text{force with line of action through the point } o_b \text{ expressed in } \{n\} \\ \mathbf{m}^n_b &= \text{moment about the point } o_b \text{ expressed in } \{n\} \\ \boldsymbol{\Theta}_{nb} &= \text{Euler angles between } \{n\} \text{ and } \{b\} \end{aligned}$$

For simplicity, we operate with only two coordinate frames in this thesis: the North, East, Down (NED) frame, $\{n\}$, and the body-fixed frame, $\{b\}$. The three vectors in Equation (3.1) describing the motion of the surface vessel may now be defined

$$\boldsymbol{\eta} = \begin{bmatrix} \mathbf{p}^n_{b/n} \\ \boldsymbol{\Theta}_{nb} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} \mathbf{v}^b_{b/n} \\ \boldsymbol{\omega}^b_{b/n} \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} \mathbf{f}^b_b \\ \mathbf{m}^b_b \end{bmatrix}, \tag{3.2}$$

where $\boldsymbol{\eta} \in \mathbb{R}^3 \times SO(3)$ is the position and orientation vector, $\boldsymbol{\nu} \in \mathbb{R}^6$ the linear and angular body-fixed velocity vector and $\boldsymbol{\tau} \in \mathbb{R}^6$ the generalized force vector respectively. The vector components follow the SNAME (1950) notation and naming conventions:

$$\begin{aligned} \mathbf{p}^n_{b/n} &= (N,\, E,\, D)^\top \in \mathbb{R}^3 & \boldsymbol{\Theta}_{nb} &= (\phi,\, \theta,\, \psi)^\top \in SO(3) \\ \mathbf{v}^b_{b/n} &= (u,\, v,\, w)^\top \in \mathbb{R}^3 & \boldsymbol{\omega}^n_{b/n} &= (p,\, q,\, r)^\top \in \mathbb{R}^3 \\ \mathbf{f}^b_b &= (X,\, Y,\, Z)^\top \in \mathbb{R}^3 & \mathbf{m}^b_b &= (K,\, M,\, N)^\top \in \mathbb{R}^3 \end{aligned}$$

where motion in the $(x,\, y,\, z)$ directions are known as surge, sway and heave motion and rotation about the same axes is known as roll, pitch and yaw respectively.

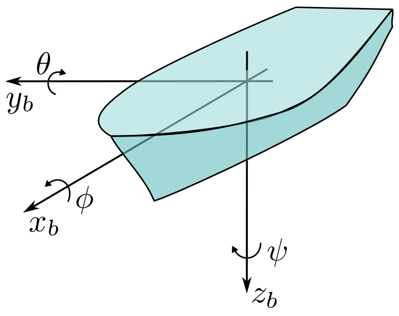


Figure 3.1: 6-DOF marine vessel naming and notation.

In this thesis, a simplified 3-DOF vessel model (Fossen, 2011) is used to describe the surface vehicle motions in the horizontal plane. This model is chosen for its generality, as it may describe the motion of a wide range of surface vehicles with sufficient accuracy. By extracting the components related to motion in surge, sway and yaw from (3.1), the 3-DOF equations of motion may be written

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{J}(\psi)\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \boldsymbol{\tau} + \boldsymbol{\tau}_{\text{ext}}, \end{aligned} \tag{3.3}$$

where $\boldsymbol{\eta} = (N,\, E,\, \psi)^\top = (x,\, y,\, \psi) \in \mathbb{R}^2 \times SO(2)$, $\boldsymbol{\nu} = (u,\, v,\, r)^\top \in \mathbb{R}^3$ and $\boldsymbol{\tau} = (X,\, Y,\, N)^\top \in \mathbb{R}^3$. For simplicity, the body origin is chosen such that it coincides with the vessel's center of gravity.

The transformation matrix $\mathbf{J}(\psi)$ in (3.3) is equal to the rotation matrix

$$\mathbf{J}(\psi) \triangleq \mathbf{R}_{z,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.4}$$

$\mathbf{M} = \mathbf{M}_{RB} + \mathbf{M}_A$ is the mass matrix consisting of the rigid-body mass and hydrodynamic added mass.

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix}, \quad \mathbf{M}_A = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & N_{\dot{v}} & N_{\dot{r}} \end{bmatrix}, \tag{3.5}$$

where m is the vessel mass and I_z is the moment of inertia about the z-axis.

$\mathbf{C}(\boldsymbol{\nu}) = \mathbf{C}_{RB}(\boldsymbol{\nu}) + \mathbf{C}_A(\boldsymbol{\nu})$ is the Coriolis and centripetal matrix. The rigid-body and added Coriolis matrices for our system are

$$\begin{aligned} \mathbf{C}_{RB}(\boldsymbol{\nu}) &= \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix}, \\ \mathbf{C}_A(\boldsymbol{\nu}) &= \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix}. \end{aligned} \tag{3.6}$$

Finally, $\mathbf{D}(\boldsymbol{\nu})$, is the nonlinear damping matrix. It may be defined as

$$\mathbf{D}(\boldsymbol{\nu}) = \mathbf{D}_L + \mathbf{D}_{NL}(\boldsymbol{\nu}) \tag{3.7}$$

where $\mathbf{D}_L$ accounts for linear damping and $\mathbf{D}_{NL}$ accounts for nonlinear damping

$$\begin{aligned} \mathbf{D}_L &= -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix}, \\ \mathbf{D}_{NL}(\boldsymbol{\nu}) &= -\begin{bmatrix} X_{|u|u}|u| + X_{uuu}u^2 & 0 & 0 \\ 0 & Y_{|v|v}|v| + Y_{vvv}v^2 & 0 \\ 0 & 0 & N_{|r|r}|r| + N_{rrr}r^2 \end{bmatrix}. \end{aligned} \tag{3.8}$$

3.1.1 Actuator Forces and Low-level Control

The actuator forces are given through the generalized force vector $\boldsymbol{\tau}$. It is assumed that the vessel has only two control inputs: the propeller force and rudder angle, and therefore the vessel is *underactuated* (Fossen, 2011). This is modeled with the force vector

$$\boldsymbol{\tau} = \begin{bmatrix} X \\ Y \\ N \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ l_r F_y \end{bmatrix}, \tag{3.9}$$

where l_r is the rudder length. $F_x \propto n \cos\delta$ and $F_y \propto n \sin\delta$, given propeller shaft speed n and rudder deflection angle δ.

In the simulator, feedback linearization is used (Sec. 4.2.1), however in a practical situation where an accurate vessel model is unattainable, other control methods, such as a multi-variable PID controller, would most likely yield better results.

3.1.2 Environmental Forces

Environmental disturbances act upon the vessel through the disturbance vector $\boldsymbol{\tau}_{\text{ext}}$. In this thesis, the environmental forces are split in two parts: slow varying (*e.g.* wind and current forces), $\boldsymbol{\tau}_c$, and fast varying (first-order wave forces), $\boldsymbol{\tau}_w$.

Slow Varying Forces

Slow varying forces such as ocean currents, wind and second-order wave forces are combined into one constant force acting in the NED frame. It is assumed that these forces are irrotational. Thus, the combined ocean current and wind effect is given as

$$\boldsymbol{\tau}_c = \begin{bmatrix} X_c \\ Y_c \\ 0 \end{bmatrix}. \tag{3.10}$$

First-order Wave Forces

First-order wave forces are modeled as filtered Gaussian white noise (Brown & Hwang, 2012). The filter is a second-order wave transfer function approximation (Fossen, 2011) on the form

$$h(s) = \frac{K_w s}{s^2 + 2\lambda\omega_0 s + \omega_0^2}, \quad K_w = 2\lambda\omega_0\sigma, \tag{3.11}$$

where σ describes the wave intensity, λ is a damping coefficient and ω_0 is the dominating wave frequency. The waves affect the motion of the ASV in surge, sway and yaw. Thus three independent wave filters are needed for modeling the wave disturbance in all degrees of freedom.

A state space representation of the wave filter may be written as

$$\begin{aligned}
\begin{bmatrix} \dot{\boldsymbol{\xi}}_w \\ \dot{\boldsymbol{\tau}}_w \end{bmatrix} &= \underbrace{\begin{bmatrix} 0 & \mathbf{I} \\ -\boldsymbol{\Omega}_0^2 & -2\boldsymbol{\Lambda}\boldsymbol{\Omega}_0 \end{bmatrix}}_{\mathbf{A}_w} \begin{bmatrix} \boldsymbol{\xi}_w \\ \boldsymbol{\tau}_w \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \mathbf{K}_w \end{bmatrix}}_{\mathbf{E}_w} \mathbf{w}, \\
y &= \underbrace{\begin{bmatrix} 0 & \mathbf{I} \end{bmatrix}}_{\mathbf{C}_w} \begin{bmatrix} \boldsymbol{\xi}_w \\ \boldsymbol{\tau}_w \end{bmatrix},
\end{aligned} \tag{3.12}$$

where $\boldsymbol{\xi}_w \in \mathbb{R}^3$ is an internal filter state, $\boldsymbol{\tau}_w \in \mathbb{R}^3$ is the generalized wave forces and $\mathbf{w} \in \mathbb{R}^3$ is a vector of Gaussian white noise. $\mathbf{I} \in \mathbb{R}^{3\times3}$ is the identity matrix, and the matrices $\boldsymbol{\Omega}_0, \boldsymbol{\Lambda}, \mathbf{K}_w \in \mathbb{R}^{3\times3}$ are chosen as

$$\begin{aligned}
\boldsymbol{\Omega}_0 &= \operatorname{diag}(\omega_{0,X}, \omega_{0,Y}, \omega_{0,N}), \\
\boldsymbol{\Lambda} &= \operatorname{diag}(\lambda_X, \lambda_Y, \lambda_N), \\
\mathbf{K}_w &= \operatorname{diag}(K_{w,X}, K_{w,Y}, K_{w,N}).
\end{aligned} \tag{3.13}$$

3.2 Configuration Spaces

Spong, Hutchinson, and Vidyasagar (2006) define a robot's configuration, $\mathbf{q}$, as *"... a complete specification of the location of every point on the robot."* Furthermore, the configuration space, $\mathcal{C}$, is defined as *"the set of all possible configurations"*.

For a 3-DOF surface vehicle, its configuration is $\boldsymbol{\eta} = (x, y, \psi)^\top$ and its configuration space is $\mathcal{C} = \mathbb{R}^2 \times SO(2)$. The set of configurations for which the vehicle collides with an obstacle is referred to as the *configuration space obstacle* and is defined as

$$\mathcal{CO} = \{\boldsymbol{\eta} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{\eta}) \cap \mathcal{O} \neq \emptyset\}, \tag{3.14}$$

where $\mathcal{O} = \cup \mathcal{O}_i$ is the total space occupied by all obstacles and $\mathcal{A}(\boldsymbol{\eta})$ is the space occupied by robot A for a given configuration $\boldsymbol{\eta}$. The set of collision-free configurations, the *free configuration space*, is

$$\mathcal{C}_{\text{free}} = \mathcal{C} \backslash \mathcal{CO}. \tag{3.15}$$

To ease computational demands, higher order configuration spaces are often collapsed in to a simple two- or three-dimensional cartesian space (Fig. 3.3) by approximating the vessel as a disc (Fig. 3.2) or a sphere.

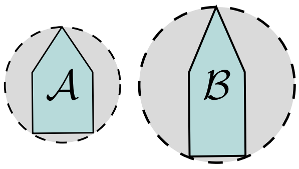
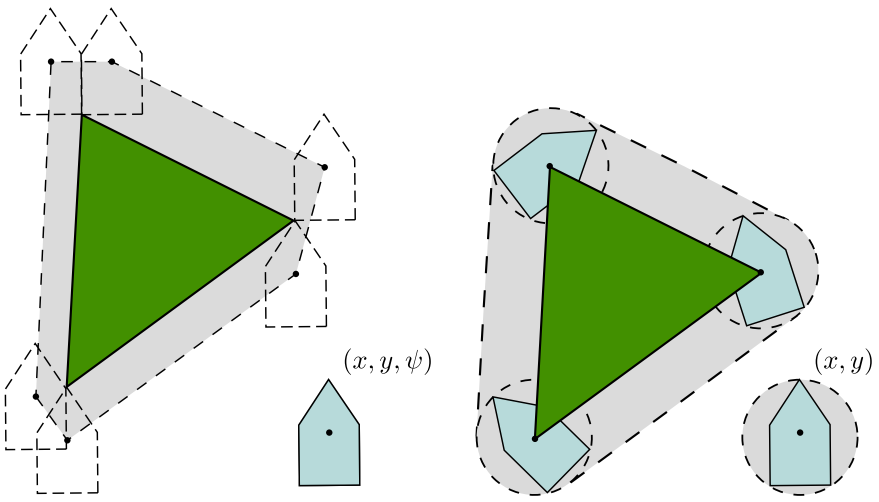


Figure 3.2: Approximating a vessel as a disc. With a long slender vessel as B, the disc approximation worsens.



(a) A two-dimensional snapshot of the three-dimensional configuration space

(b) Collapsed configuration space

Figure 3.3: Collapsing a three-dimensional configuration space. By approximating the vessel as a disc of radius r, the three-dimensional configuration space may be collapsed into two dimensions by "padding" all obstacles as shown in (b).

3.3 Planning Algorithms

The term *planning algorithms* was first introduced in LaValle (2006) as a broad term referring to a common ground between the fields of artificial intelligence, robotics and control theory. The term encompasses a broad range of algorithms often distinguished in two categories: local and global methods.

3.3.1 Local Methods

Local methods, often reffered to as *reactive* or *dynamic*, considers the immediate environment obtained through available sensor information and plan accordingly. They are responsive and orders of magnitude faster than global methods. However, as they only considers a subset of the configuration space, they are susceptible to local minima and may get "stuck" (Fig 3.4). Some examples of local methods are the Dynamic Window (DW) (Fox et al., 1997), the Virtual Force Field (VFF) (Borenstein & Koren, 1989) and Vector Field Histogram (VFH) (Borenstein & Koren, 1991).

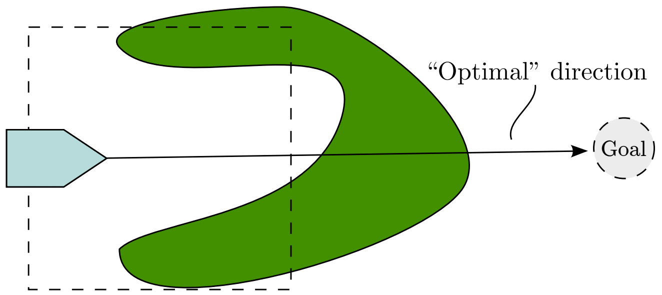


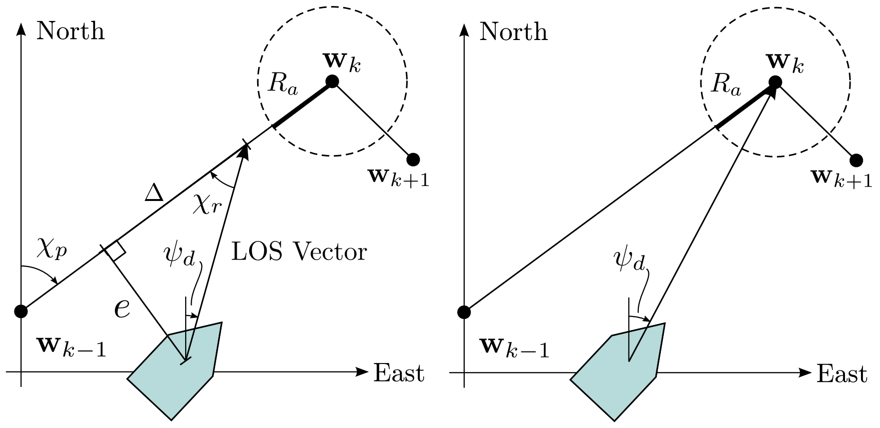
Figure 3.4: Local method getting stuck near local minima. As the vessel travels towards the goal, it is unable to recognize the dead-end. A local method is a "greedy" method, meaning that it only considers solutions optimal at the current time step. Getting unstuck requires moving away from the obstacle, a "sub-optimal" choice, for several time steps. The dashed rectangle indicates the field of vision of the local method.

3.3.2 Global Methods

Global methods considers the full configuration space. They use a priori knowledge, *e.g.*, a map, to find a complete sequence of states describing a complete motion from the initial state to the final state. Many global methods are *complete*, meaning that if such a sequence of states exists, it will find it. These methods are sometimes referred to as *deliberate* in the literature. As the global methods compute all subsequent motions and not just the next, they are naturally slower than their local counterparts. By reducing accuracy, using heuristics or by other means, some methods may run in a fraction of a second, but they are still in general regarded as slow. Examples of global methods are A* (Hart et al., 1968), Rapidly-exploring Random Trees (RRTs) (LaValle, 1998) and Hybrid-state A* (Montemerlo et al., 2008; Dolgov et al., 2010).

3.4 Guidance Laws

Most local methods expect a single goal configuration or a control input reference, but often it is desirable to be able to give the system a set of waypoints or a path obtained from global methods. Thus, a path tracking scheme is needed. Popular methods are the simple pure pursuit scheme (Fossen, 2011, p. 244) or the more advanced Line of Sight (LOS) method (Fossen, Breivik, & Skjetne, 2003; Fossen, 2011) (Fig. 3.5).



(a) Line of sight guidance (b) Pure pursuit guidance

Figure 3.5: Guidance algorithms. The LOS algorithm (a) steers the vessel onto the line between the waypoints, where as the pure pursuit method (b) simply steers the vessel towards its current waypoint.

Pure Pursuit Guidance

The pure pursuit method (Fig. 3.5b) is the simplest form of waypoint following and is given as

$$\psi_d = \text{atan2}(y_k - y,\, x_k - x), \quad u_d = u_0, \tag{3.16}$$

where $\mathbf{w}_k = (x_k,\, y_k)^\top$ is the coordinate of the current waypoint and u_0 is a predefined desired surge speed. The method essentially steers the vessel towards the current waypoint.

Line of Sight Guidance

The LOS method (Fig. 3.5a) steers the vessel towards the straight line in between the previous and current waypoint. The desired heading (or course) and surge velocity set-points are given as

$$\psi_d = \chi_p + \chi_r(e), \quad u_d = u_0, \tag{3.17}$$

where

$$\chi_p = \operatorname{atan2}(y_k - y_{k-1},\, x_k - x_{k-1}) \tag{3.18}$$

is the path-tangential angle,

$$\chi_r = \operatorname{atan2}(-e,\, \Delta) \tag{3.19}$$

is the velocity-path relative angle,

$$e = -(x - x_k)\sin\chi_p + (y - y_k)\cos\chi_p \tag{3.20}$$

is the cross-track error and Δ is the predefined lookahead distance (Fig. 3.5a).

Optionally, one may want to add integral effect to counteract the effects of unknown disturbances such as ocean current. This is know as the Integral Line of Sight (ILOS) method and the velocity-path relative angle changes to

$$\chi_r = \operatorname{atan2}(-K_p e - K_i \int_0^t e(\tau)\,\mathrm{d}\tau,\, \Delta) \tag{3.21}$$

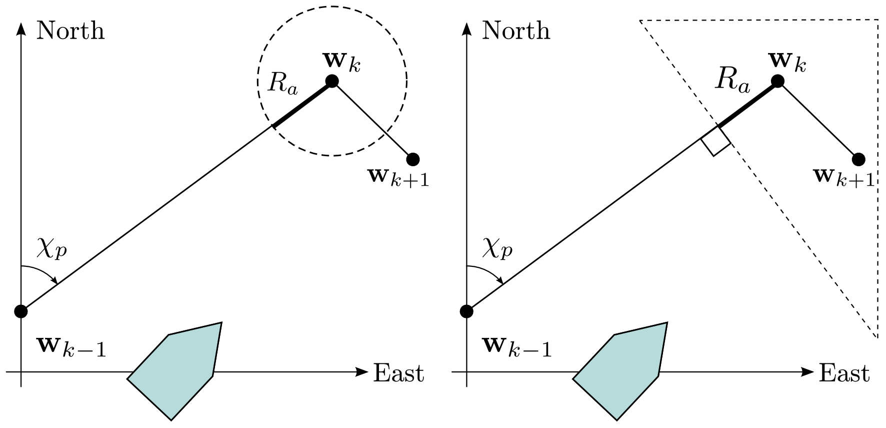
One should always be careful when adding integral action as it adds a destabilizing effect. It is common to limit the integrator (anti-windup) and reset at a waypoint switch if the new path-tangential angle differs from the old by more than a certain limit (*e.g.* 45°).

Switching Criteria

When following a path described by a set of waypoints, one needs to know when to switch to the next waypoint. One such method is known as the "circle of acceptance" method (Fig. 3.6a). Given the current waypoint $\mathbf{w}_k = (x_k, y_k)^\top$, the switching criterion is defined as follows

$$(x_k - x)^2 + (y_k - y)^2 \leq R_a^2, \tag{3.22}$$

i.e., when Equation (3.22) holds, the controller switches to the next waypoint.



(a) Circle of acceptance method (b) Progress along path method

Figure 3.6: Switching criterions. The circle of acceptance method (a) will make sure every waypoint is visited, whereas the progress along path method (b) allows deviation from the path.

An alternative strategy is to switch waypoint whenever the vessel "passes" the current waypoint, *i.e.*, when the vessel crosses the line normal to the waypoint line. This method is known as the "progress along path" method (Fig. 3.6b) (Loe, 2008; Stenersen, 2014) or "along-track" method (Fossen, 2011). The switching criterion now becomes

$$(x_k - x)\cos\chi_p + (y_k - y)\sin\chi_p \leq R_a \tag{3.23}$$

Which of the switching criteria to use depends on the intention of the waypoints. If it is critical that all waypoints are visited, then the former criterion is best suited. However, if the purpose of the waypoints is to make up a path for the vessel to follow, the latter should be used.

3.5 COLREGs – Following the Rules of the Road

The International Maritime Organization (IMO) established the International Regulations for Avoiding Collisions at Sea (COLREGs) in 1972 (International Maritime Organization, 2003). COLREGs is based on a long history of regulations for preventing collisions at sea, dating back to 1840. At that time there was no single set of international rules and practices, giving rise to dangerous confusion between vessels at the risk of colliding (Wikipedia, 2015c). Over the next century an international standard emerged, culminating into the now well known COLREGs.

COLREGs is divided into five parts (A–E), but in this thesis it is the rules covered in *Part B – Steering and Sailing* that is most applicable. Of the rules in this part, rules 8 (b) and (d), 13, 14, 15, 16 and 17 are most relevant. There are *three* main scenarios to be considered: head-on, overtaking and crossing.

3.5.1 The Rules

Rule 8: Action to Avoid Collision

(b) 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.

(d) 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.

Rule 13: Overtaking

(a) 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.

(b) 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 sternlight of that vessel but neither of her sidelights.

(c) *When a vessel is in any doubt as to whether she is overtaking another, she shall assume that this is the case and act accordingly.*

(d) *Any subsequent alteration of the bearing between the two vessels shall not make the overtaking vessel a crossing vessel within the meaning of these Rules or relieve her of the duty of keeping clear of the overtaken vessel until she is finally past and clear.*

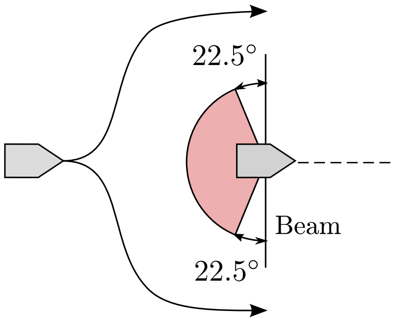


Figure 3.7: COLREGs rule 13: overtaking. In an overtaking situation the overtaking vessel may pass the other vessel on either side as long as sufficient distance is kept.

Rule 14: Head-on

(a) *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.*

(b) *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 of the other in a line or nearly in a line and/or both sidelights and by day she observes the corresponding aspect of the other vessel.*

(c) *When a vessel is in any doubt as to whether such a situation exists she shall assume that it does exist and act accordingly.*

Rule 15: Crossing

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

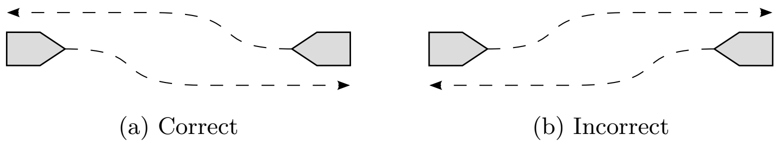


Figure 3.8: COLREGs rule 14: head-on. In a head-on situation *both* vessels are give-away vessels and are to change their course starboard.

shall, if the circumstances of the case admit, avoid crossing ahead of the other vessel.

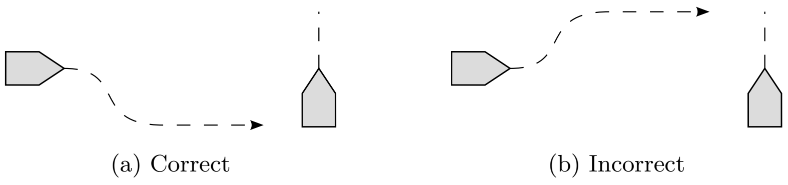


Figure 3.9: COLREGs rule 14: crossing. In a crossing situation the give-away vessel is the one with the other on its starboard side.

Rule 16: Action by Give-away Vessel

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 17: Action by Stand-on Vessel

(a) (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 manoeuvre 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 compliance with these Rules.

(b) When, from any cause, the vessel required to keep her course and speed finds herself so close that collision cannot be avoided by the action of the give-way vessel alone, she shall take such action as will best aid to avoid collision.

(c) *A power-driven vessel which takes action in a crossing situation in accordance with subparagraph (a)(ii) of this Rule to avoid collision with another power-driven vessel shall, if the circumstances of the case admit, not alter course to port for a vessel on her own port side.*

(d) *This Rule does not relieve the give-way vessel of her obligation to keep out of the way.*

3.5.2 COLREGs for an ASV

To determine a COLREGs situation, a combination of previous approaches is used (Kuwata et al., 2014; Loe, 2008; Švec et al., 2013). A collision situation is first identified by computing the Closest Point of Approach (CPA), given the current poses of the ASV and the potential hazard.

Closest Point of Approach

Let $\mathbf{p}_A$ be the position of the ASV and $\mathbf{p}_B$ be the position of the obstacle and let $\mathbf{v}_A$ and $\mathbf{v}_B$ be their velocity vectors in the global frame. Then, the time to CPA may be found as (Kuwata et al., 2014)

$$t_{\text{CPA}} = \frac{(\mathbf{p}_B - \mathbf{p}_A) \cdot (\mathbf{v}_A - \mathbf{v}_B)}{\|\mathbf{v}_A - \mathbf{v}_B\|} \tag{3.24}$$

and the distance between the vessel at CPA (Fig. 3.10) is then

$$d_{\text{CPA}} = \|(\mathbf{p}_A + \mathbf{v}_A t_{CPA}) - (\mathbf{p}_B + \mathbf{v}_B t_{\text{CPA}})\| \,. \tag{3.25}$$

Note that if $\|\mathbf{v}_A - \mathbf{v}_B\| \to 0$ then $t_{\text{CPA}} \to \pm\infty$, where the physical interpretation of this is that the vessels follow the same course with equal speed and the distance between the vessels will remain constant.

Thus, the vessel is deemed to be in a collision situation if

$$0 \leq t_{\text{CPA}} \leq t_{\max} \quad \wedge \quad d_{\text{CPA}} \leq d_{\min} \tag{3.26}$$

COLREGs Rule Selection

When a possible collision situation is present, it remains to identify the applicable COLREGs rule. An efficient and easy way of determining the situation is to

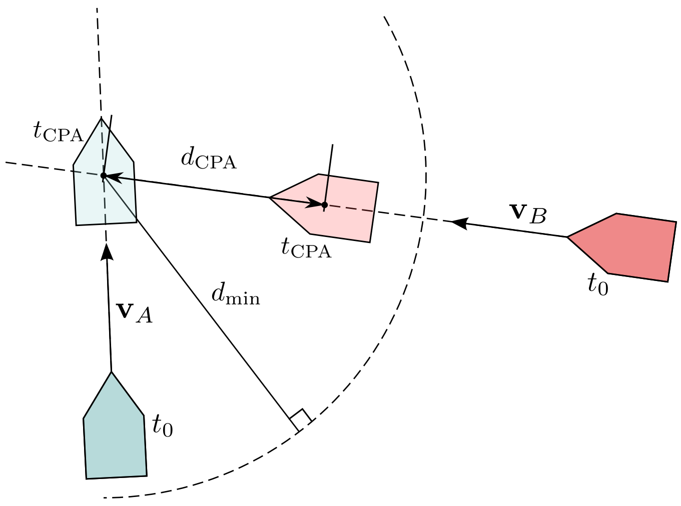


Figure 3.10: Closest Point of Approach. At time t_{CPA}, the two vessels reaches the Closest Point of Approach (CPA), with the distance d_{CPA}. In this case $d_{\mathrm{CPA}} < d_{\min}$ and the situation is classified as a potential collision situation.

calculate the relative bearing β between the ASV and the approaching hazard. The relative bearing is found as

$$\beta = \operatorname{atan2}(y_A - y_B,\, x_A - x_B) - \psi_B \tag{3.27}$$

With the relative bearing found, the COLREGs situation is determined according to Figure 3.11. Four different sectors are defined, each corresponding to a different COLREGs situation. The limits for the overtaking sector are given directly from Rule 13, however the limits for the remaining sectors are more difficult to determine. A size of 30° was used in Loe (2008), Benjamin et al. (2006), Colito (2007) and this choice seems reasonable. Thus, the sectors are defined as

1. Head-on: $\beta \in [-15°, 15°)$
2. Crossing from right: $\beta \in [15.0°, 112.5°)$
3. Overtaking: $\beta \in [112.5°, 180°) \cup [-180°, -112.5°)$
4. Crossing from left: $\beta \in [-112.5°, -15°)$

Note that the relative bearing determines the type of COLREGs situation, not if such a situation exists. This is established using the combined information from the relative bearing and the vessels' velocity vectors. Consider, for example,

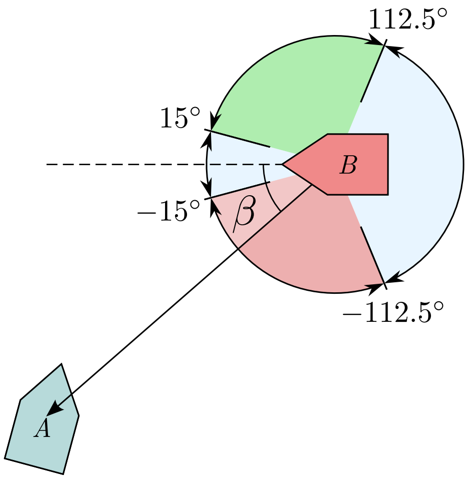


Figure 3.11: COLREGs rule selection. The choice of COLREGs rule is based on the relative bearing of the vessels. In this situation, the lower left vessel is the give-away vessel in a crossing from right situation.

a situation where the relative bearing is in the overtaking zone, but the angle between the velocity vectors is greater than 90°, then this cannot be considered an overtaking situation as the vessels are on opposite courses.

Furthermore, even though the relative bearing changes during a COLREGs maneuver, it does not mean that the situation is necessarily over. If, for example, the ASV is overtaking another vessel and seeing it on its left hand (port) side, the overtaking situation is not over when the relative bearing enters the crossing (from left) zone and the ASV is still the give-way vessel (Sec. 6.2.2).

3.6 Velocity Obstacles

The Velocity Obstacle (VO) is a collision avoidance concept that has been reinvented several times over the last century under different names (Wikipedia, 2014), such as the maneuvering-board approach (Tychonievich et al., 1989), the velocity obstacle (Fiorini & Shiller, 1993), collision cones (Chakravarthy & Ghose, 1998) and forbidden velocity maps (Damas & Santos-Victor, 2009). There has also been proposed several extensions to the algorithm, *e.g.*, generalized velocity obstacles (Wilkie et al., 2009), reciprocal n-body velocity obstacles (van den Berg, Guy, Lin, & Manocha, 2011a) and acceleration-velocity obstacles (van den Berg, Snape, Guy, & Manocha, 2011b).

3.6.1 The Velocity Obstacle

First, we introduce a general definition:

Definition 3.1 *The Velocity Obstacle (VO) for A induced by B is the set of all relative velocities of A with respect to B that will result in a collision between A and B.*

In other words, the velocity obstacle is the set of velocities that, if chosen from, will eventually lead to a collision between the two vessels (Fig. 3.12).

Let $\mathbf{p} \in \mathbb{R}^2$ denote the position of a vessel and $\mathbf{v} \in \mathbb{R}^2$ denote its velocity. The relative position of the obstacle with respect to the vessel is $\mathbf{p}_{AB} = \mathbf{p}_B - \mathbf{p}_A$, likewise their relative velocity is $\mathbf{v}_{BA} = \mathbf{v}_A - \mathbf{v}_B$. Furthermore, let a ray starting from $\mathbf{p}$ going in the direction of $\mathbf{v}$ be defined as

$$\lambda(\mathbf{p}, \mathbf{v}) = \{\mathbf{p} + \mathbf{v}t \mid t \geq 0\}. \tag{3.28}$$

Given a vessel of shape $\mathcal{A}$ and an obstacle of shape $\mathcal{B}$, the VO of obstacle B in the velocity space of A may formally be written as

$$VO_{A|B} = \{\mathbf{v}_A \mid \lambda(\mathbf{p}_A, \mathbf{v}_{BA}) \cap (\mathcal{B} \oplus -\mathcal{A}) \neq \emptyset\}, \tag{3.29}$$

where $\mathcal{A} \oplus \mathcal{B} = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in \mathcal{A}, \mathbf{b} \in \mathcal{B}\}$ is the Minkowski sum of the sets $\mathcal{A}$ and $\mathcal{B}$ and $-\mathcal{A} = \{-\mathbf{a} \mid \mathbf{a} \in \mathcal{A}\}$ is the reflection of the set $\mathcal{A}$. Note that is analogous to

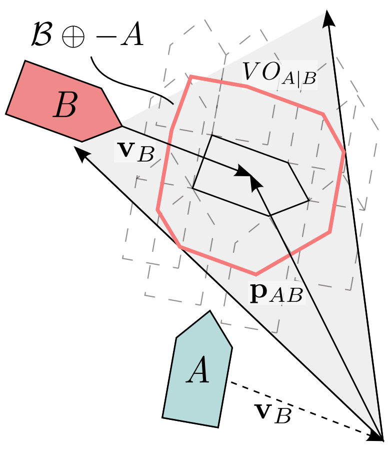
$$VO_{A|B} = \{\mathbf{v}_A \mid \lambda(\mathbf{p}_A, \mathbf{v}_{BA}) \cap \mathcal{CO} \neq \emptyset\}, \tag{3.30}$$

using the definition of configuration spaces given in Section 3.2 (Fig. 3.12a).

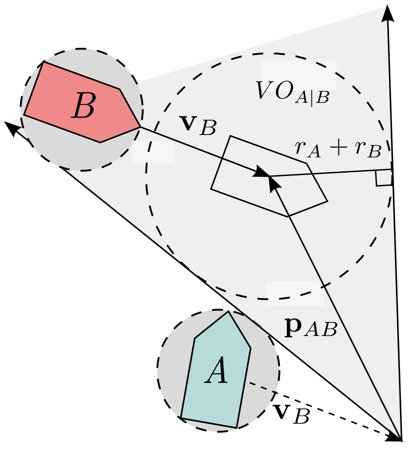
The definition can be simplified by assuming disc shaped vessels with the combined radius $r_{AB} = r_A + r_B$, giving

$$VO_{A|B} = \{\mathbf{v}_A \mid \lambda(\mathbf{p}_A, \mathbf{v}_{BA}) \in D(\mathbf{p}_{AB}, r_{AB})\}, \tag{3.31}$$

where $D(\mathbf{x}, r)$ is a disk with center $\mathbf{x}$ and radius r (Fig. 3.12b). Simplifying the shape of the vessels is equivalent to collapsing the configuration space from $\mathcal{C} = \mathbb{R}^2 \times SO(2)$ to $\mathcal{C} = \mathbb{R}^2$ (Sec. 3.2).



(a) The velocity obstacle using Minkowski addition.



(b) The velocity obstacle with a collapsed configuration space.

Figure 3.12: The velocity obstacle $VO_{A|B}$ may be found using Minkowski addition (a) or by reducing the problem by assuming disc shaped vessels (b). The red polygon in (a) is the resulting configuration space obstacle found by taking the Minkowski sum. If vessel A choose a velocity within the VO region, the vessels will collide in finite time.

3.6.2 Properties of the Velocity Obstacle

The definitions of the velocity obstacle in Equations (3.29) and (3.31) are useful in the mathematical sense, but not well suited for a control system implementation. By investigating some of the geometrical properties of the method, the process of determining if a velocity is in the VO region is greatly simplified.

The VO region has the geometric shape of a cone[1], and may therefore be represented as

$$VO_{A|B} = \{\mathbf{v}_A \mid \mathbf{v}_{BA} \cdot \mathbf{p}^{\perp}_{AB,\,\text{left}} \geq 0 \wedge \mathbf{v}_{AB} \cdot \mathbf{p}^{\perp}_{AB,\,\text{right}} \geq 0\} \tag{3.32}$$

where $(\cdot)$ is the vector dot product and $\mathbf{p}^{\perp}_{AB,\,\text{left}}$ and $\mathbf{p}^{\perp}_{AB,\,\text{right}}$ are vectors perpendicular to the left and right edges of the cone, respectively (Fig. 3.13) (Guy et al., 2009). Noting the geometric relations in Figure 3.12 and 3.13, these vectors may

[1]This is true for both the general and simplified definition.

be found as

$$\mathbf{p}_{AB,\,\text{left}}^{\perp} = \mathbf{R}(-\alpha + \pi/2)\frac{\mathbf{p}_{AB}}{\|\mathbf{p}_{AB}\|}, \quad \mathbf{p}_{AB,\,\text{right}}^{\perp} = \mathbf{R}(\alpha - \pi/2)\frac{\mathbf{p}_{AB}}{\|\mathbf{p}_{AB}\|}, \tag{3.33}$$

where $\mathbf{R}$ is the rotation matrix

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{3.34}$$

and α is the angle between the center line and the cone edges, given as

$$\alpha = \arcsin\left(\frac{r_A + r_B}{\|\mathbf{p}_{AB}\|}\right). \tag{3.35}$$

Furthermore, the region outside of the VO may be split into three distinct parts (Fig. 3.13) (Kuwata et al., 2014). Choosing a velocity in one of these regions yields three distinct maneuvers.

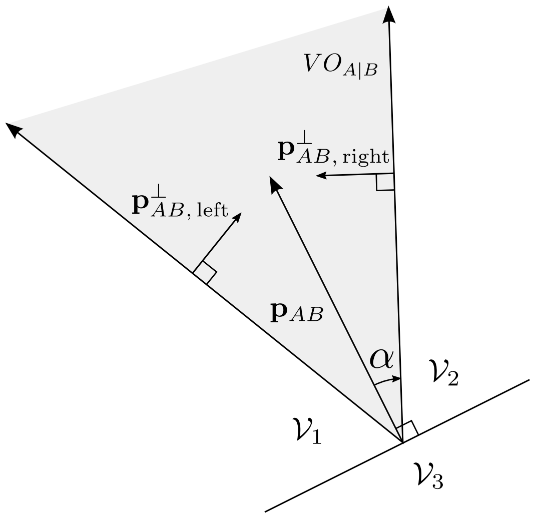


Figure 3.13: Velocity obstacle regions. The velocity obstacle cone splits the velocity space into four regions: the cone itself, $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$.

The first set, $\mathcal{V}_1$, is found as

$$\mathcal{V}_1 = \left\{\mathbf{v}_A \mid \mathbf{v}_A \notin VO_{A|B} \cup \mathcal{V}_3 \wedge [\mathbf{p}_{AB} \times \mathbf{v}_{BA}]_z < 0\right\}, \tag{3.36}$$

where the $[\cdot]_z$ operator extracts the z component of the vector. If a velocity is chosen from $\mathcal{V}_1$, vessel A will pass B seeing it on its right hand side. The second set, $\mathcal{V}_2$, is found as

$$\mathcal{V}_2 = \left\{ \mathbf{v}_A \mid \mathbf{v}_A \notin \mathcal{V}_1 \cup \mathcal{V}_3 \cup VO_{A|B} \right\}. \tag{3.37}$$

If a velocity is chosen from $\mathcal{V}_2$, vessel A will pass B seeing it on its left hand side. The third set, $\mathcal{V}_3$, is found as

$$\mathcal{V}_3 = \{ \mathbf{v}_A \mid \mathbf{p}_{AB} \cdot \mathbf{v}_{BA} < 0 \}. \tag{3.38}$$

If the velocity is chosen from $\mathcal{V}_3$, vessel A will move away from the other vessel. Note that definitions of these sets remains valid even for the formal definition of the VO given in Equation 3.29.

Chapter 4

System Implementation

The system is implemented in both C++ and Python using the ROS framework. One of the objectives of the thesis was to create a collision avoidance system ready for full-scale integration. With this in mind, the choice of ROS as the main development framework was an easy one.

The reader is referred to Appendix A and B for details regarding ROS and the implementation.

4.1 System Architecture

The system is designed to be as modular as possible and this is achieved by dividing the system into distinct components. The strict modular approach is made easy with ROS due to its message passing architecture. Three main modules have been developed:

- A simulator with low-level controllers
- LOS and pure pursuit guidance controllers
- A VO controller

These are implemented as their own ROS packages (or nodes), sharing information through message passing.

The control flow of the system (Fig. 4.1) is similar to common guidance scheme with the addition of the VO controller for obstacle avoidance. The ROS implementation for a simulated system (Fig. 4.2) follows the same structure where the components are implemented as individual packages. Note the *obstacle tracker* node that simulates a system for sensing other vessels in the vicinity of our own and determining their state. In the current implementation of the system, other ships are simply additional instances of the vessel simulator. The obstacle tracker subscribes to their state information and publishes it as a collection of states for the ASV to subscribe to.

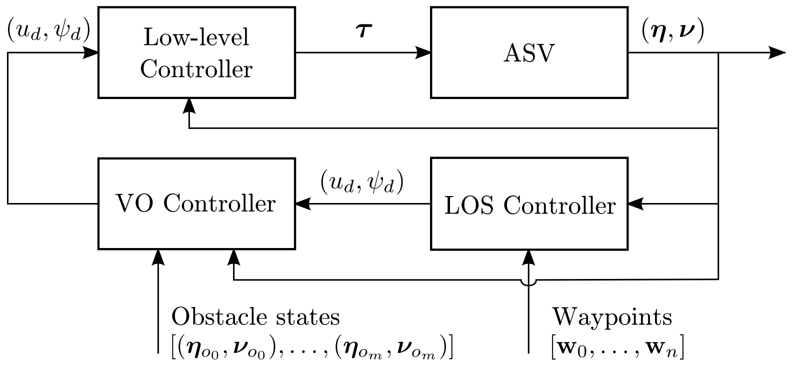

(u_d, ψ_d)
Low-level Controller
$\boldsymbol{\tau}$
ASV
$(\boldsymbol{\eta}, \boldsymbol{\nu})$
VO Controller
(u_d, ψ_d)
LOS Controller
Obstacle states
$[(\boldsymbol{\eta}_{o_0}, \boldsymbol{\nu}_{o_0}), \dots, (\boldsymbol{\eta}_{o_m}, \boldsymbol{\nu}_{o_m})]$
Waypoints
$[\mathbf{w}_0, \dots, \mathbf{w}_n]$

Figure 4.1: Control system block diagram. The LOS controller feeds the VO-controller with desired set-points in surge speed and yaw angle. These set-points determines the minima in the VO velocity field with no obstacles present.

In a real application, the simulator may simply be replaced by sensor drivers, a state estimator and (a) motor controller(s) (Fig. 4.3). Drivers for common sensors are already available in the ROS distribution and there are also several implementations of common state estimators such as the Extended Kalman Filter (EKF), Adaptive Monte-Carlo Localization (AMCL) and Unscented Kalman Filter (UKF). Unfortunately, most state estimator implementations are tailored towards mobile robot applications.

4.2 Simulator

The simulator is an implementation of the equations of motion for a general 3-DOF surface vessel (Sec. 3.1) together with a numerical integration scheme for solving the nonlinear differential equations. It is written to be very general and may load any parameter configuration from the ROS parameter server (Sec. A.2.2) at run-time.

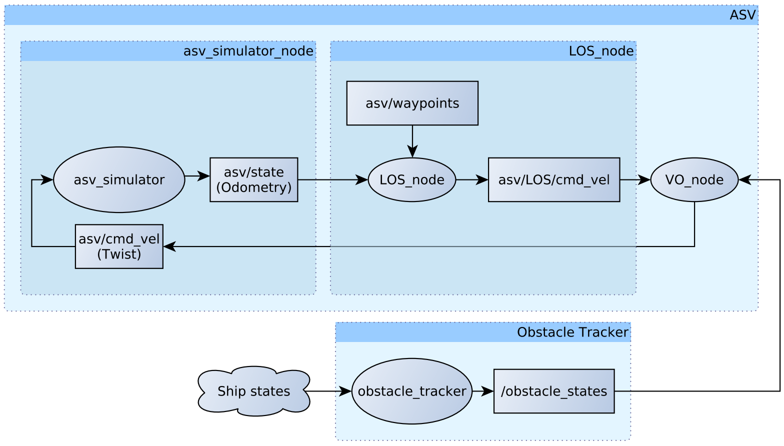


Figure 4.2: Information flow in simulated system. This graph shows how the different nodes in the system communicate. The obstacle tracker node gathers odometry data from all the simulated obstacle ships and publishes their states. The obstacles are just other instances of the simulator under different namespaces. By using a dedicated node for “tracking” these ships we can, for example, easily add a simulated disturbance or sensor noise.

The numerical integration method chosen in this thesis is a standard first order Euler method.

$$x_{k+1} = x_k + hf(x_k, t_k) \tag{4.1}$$

This method is chosen for it simplicity and speed. Its accuracy is in general poor, but suits the needs in this thesis.

4.2.1 Low-level Controllers

Two feedback-linearizing controllers and one conventional PD-controller are implemented in the simulator: a speed controller, yaw rate controller and a heading controller respectively. The speed controller is on the form

$$\begin{aligned} F_x &= (-mv + Y_{\dot{v}}v + Y_{\dot{r}}r)r \\ &\quad - (X_u + X_{|u|u}|u| + X_{uuu}u^2)u + K_{p,u}m(u_d - u), \end{aligned} \tag{4.2}$$

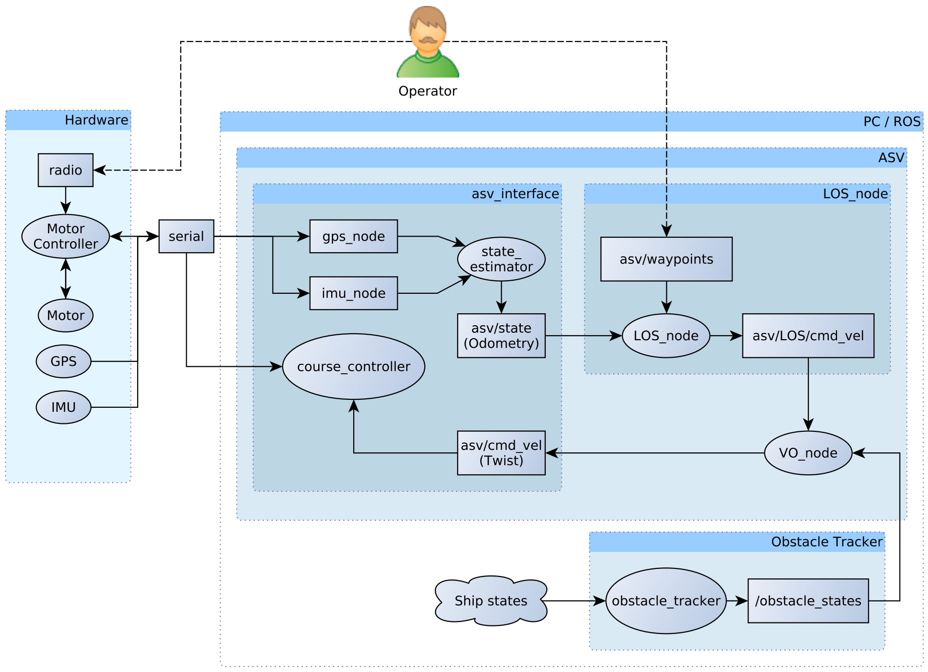


Figure 4.3: Information flow in real system. Simply replacing the simulator node with a hardware interface is enough to launch the system on a real platform.

the yaw rate controller

$$F_y = (mu - X_{\dot{u}}u)r - (Y_v v + Y_r r + Y_{|v|v}|v|v + Y_{vvv}v^3) + \frac{K_{p,r}I_z}{l_r}(r_d - r), \quad (4.3)$$

and finally the heading PD-controller is

$$F_y = \frac{K_{p,\psi}I_z}{l_r}((\psi_d - \psi) - K_{d,\psi}r). \quad (4.4)$$

Both the yaw and yaw rate controller are implemented and may easily be interchanged at run-time. However, in practice only the yaw controller is used in this thesis as the VO algorithm outputs a speed and heading reference. The parameters used for the simulator and controllers for the simulations are listed in Table 4.1.

Table 4.1: Simulator parameters. The vessel parameters are based on the Viknes 830 (Loe, 2008). Note that it is assumed $\mathbf{M}_A = \mathbf{C}_A = 0$, which yields a slightly unrealistic model, however it captures the vessel dynamics sufficiently.

(a) Vessel parameters

Parameter	Value	Unit
m	3980.0	kg
I_z	19703.0	kg/m^2
X_u	-50.0	kg/s^2
$X_{\|u\|u}$	-135.0	kg/m^2
X_{uuu}	0.0	kg/(m·s)^2
Y_v	-200.0	kg/m^2
$Y_{\|v\|v}$	-2000.0	kg/s^2
Y_{vvv}	0.0	kg/(m·s)^2
N_r	-3224.0	kg·m^2/s
$N_{\|r\|r}$	0.0	kg·m^2
N_{rrr}	-3224.0	kg·m^{2}s
N_v	0.0	kg·m/s
Y_r	0.0	kg·m/s
$F_{x,\max}$	13100.0	N
$F_{x,\min}$	-6550.0	N
$F_{y,\max}$	645.0	N
l_r	4.0	m

(b) Controller parameters

Parameter	Value	Unit
$K_{p,u}$	0.1	1/s
$K_{p,\psi}$	5.0	1/s
$K_{d,\psi}$	1.0	s

4.3 Guidance

An ILOS guidance controller (Sec. 3.4) has been implemented as a part of the control structure (Fig. 4.1). The integrator is limited through anti-windup and reset when switching waypoints if the new path-tangential angle differs more than 45° from the previous. The ILOS controller also implements both the “circle of acceptance” and “progress along path” switching criteria. The parameters used by the guidance system are summarized in Table 4.2.

4.4 Velocity Obstacle Controller

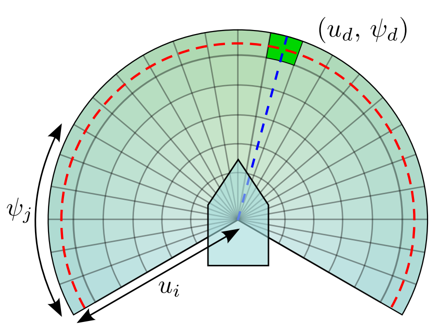
The velocity obstacle controller implements the VO algorithm described in Section 3.6. It consists of two main parts: a dynamic and static part.

Table 4.2: Guidance system simulation parameters. In the simulations (Chap. 5), no constant disturbance has been added and therefore the integral gain is set to zero.

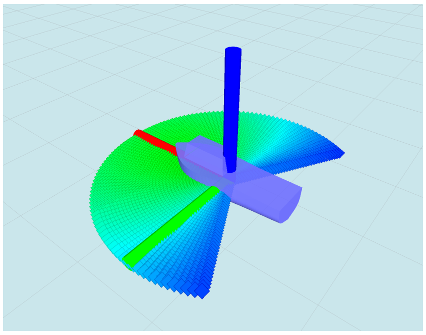
Parameter	Value	Unit
R_a	20.0	m
Δ	40.0	m
K_i	0.0	1/s

4.4.1 The Velocity Field

At each update of the controller, it searches a predefined discretized velocity field which is centered around the current velocity of the vessel (Fig. 4.4).



(a) Discretized velocity field



(b) Velocity field implemented in ROS

Figure 4.4: VO controller velocity field. The controller loops through the discretized set of velocity pairs assigning each a cost according to Equation (4.6).

The field is defined as two dimensional grid consisting of (u_i, ψ_j) velocity pairs. The surge speed and heading angle is linearly scaled with N_u samples in the range $[u_{\min}, u_{\max}]$ and N_ψ samples in the range $[-\psi_{\max} + \psi, \psi_{\max} + \psi]$, respectively. First, an error velocity vector is defined as

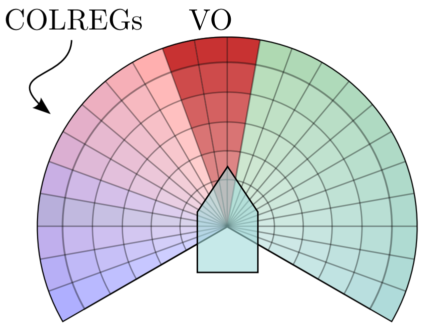
$$\tilde{\mathbf{v}}_{i,j} = \begin{bmatrix} u_d \cos\psi_d - u_i \cos\psi_j \\ u_d \sin\psi_d - u_i \sin\psi_j \end{bmatrix}, \tag{4.5}$$

then a cost is assigned to each velocity pair (u_i, ψ_j) according to the following cost function

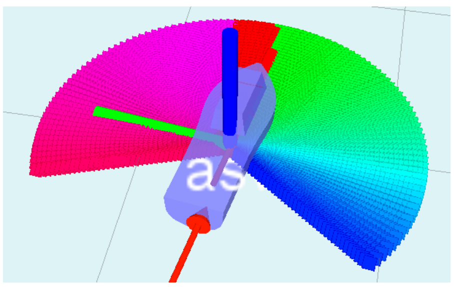
$$C = \alpha \tilde{\mathbf{v}}_{i,j}^\top \mathbf{Q} \tilde{\mathbf{v}}_{i,j} + \beta f_{\text{COLREGs}}(u_i, \psi_j) + \gamma f_{\text{collision}}(u_i, \psi_j), \tag{4.6}$$

where $\mathbf{Q} \in \mathbb{R}^{2\times 2}$ is a positive definite weighing matrix. The constants α, β and γ are scaling factors. Changing the elements of $\mathbf{Q}$ allows for different weighing of the cross-track and along-track velocity. In the simulations, $\mathbf{Q}$ is simply set as the identity matrix.

The functions f_{COLREGs} and $f_{\text{collision}}$ are Boolean functions that returns "0" or "1". The function f_{COLREGs} returns "1" if the vessel is in a collision situation (Sec. 3.5.2) and choosing the given velocity pair results in a COLREGs defying maneuver. This is done using the properties established in Section 3.6.2. Specifically, if a velocity is $\mathcal{V}_1$, choosing it would cause a COLREGs-defying maneuver when the ASV is the give-way vessel. The function $f_{\text{collision}}$ returns "1" if the vessel is in a collision situation and the given velocity pair is inside the velocity obstacle.



(a) Discretized velocity field with COLREGs and VO regions



(b) Velocity field with COLREGs and VO regions in ROS

Figure 4.5: Velocity fields with COLREGs and VO regions. The velocity field shown in (a) and (b) corresponds to a "head-on" situation. Choosing velocities from within the VO cone would set the ASV on a collision course. The velocities left of the VO cone would cause the ASV to pass the incoming vessel on the left side, thus defying COLREGs.

4.4.2 Static Obstacles

The VO algorithm natively supports static obstacles by simply considering static obstacles as dynamic with zero velocity. However, there are multiple challenges associated with this approach.

Firstly, in most practical implementations static obstacles are represented as occupied cells in a grid map. Thus, a VO cone must be generated for each of the occupied cells. This would be highly inefficient as the number of occupied cells in the map could be huge. Moreover, it is only the cells along the obstacle boundary that need to be tested, since it is not possible to collide with an internal cell of the obstacle without passing through the boundary first.

Secondly, one may construct a scenario as shown in Figure 4.6. If the goal pose is in front of the obstacle, it may never be reached. The reason for this is that any course towards an obstacle is a collision course, and therefore deemed “illegal” by the VO method. Note that this may also be a problem with stationary dynamic obstacles (see discussion in Sec. 6.1).

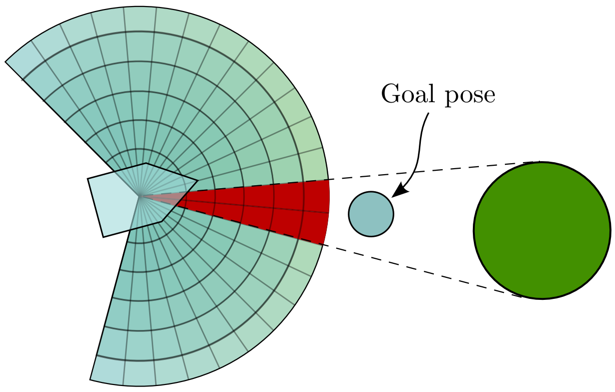


Figure 4.6: If static obstacles are treated as stationary dynamic obstacles, any course towards the obstacle will be deemed as “illegal”. This causes problems if the goal pose is in front of a static obstacle.

A Solution

The solution devised in this thesis was to loop through each velocity pair (u_i, ψ_j) and traverse a ray, λ, given as

$$\lambda(\mathbf{p}_A, u_i, \psi_j) = \mathbf{p}_A + \begin{bmatrix} \cos\psi_j & -\sin\psi_j \\ \sin\psi_j & \cos\psi_j \end{bmatrix} \begin{bmatrix} u_i t_i \\ 0 \end{bmatrix}, \quad t \in (0, t_{\max}], \tag{4.7}$$

where $\mathbf{p}_A$ is the position of the vessel, and test if the ray intersects an occupied cell (Fig. 4.7). By selecting $t_{\max} = d/u_{\max}$, where $u_{\max}$ is the largest possible surge velocity, one may define the distance d as the maximum distance the algorithm searches for obstacles.

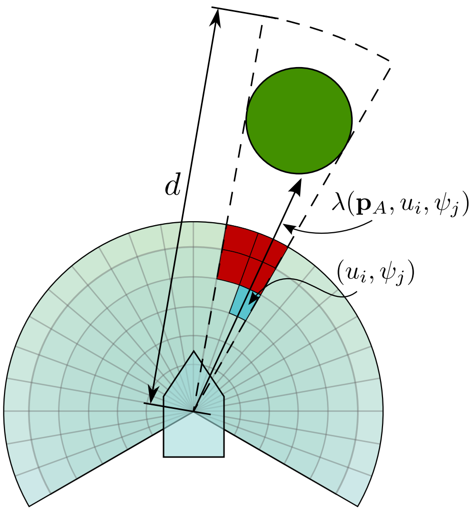


Figure 4.7: Static obstacles with the velocity obstacle algorithm. In the proposed algorithm, for each discretized direction, a ray is cast using the corresponding velocities sorted in descending order. If a velocity is found to be OK, each subsequent velocity in the same direction is marked as OK.

The algorithm is given in Listing C.2.

Chapter 5

Simulation Results

In this chapter, the results from several scenarios will be presented. The scenarios have been developed to show the versatility and robustness of the method. Having said that, it is impossible to test for everything and in the discussion (Sec. 6.1) some of the special cases were the algorithm faces particular challenges are highlighted and discussed.

There are three main scenarios considered:

1. Overtaking and crossing from the right

2. Overtaking, crossing and head-on

3. Crossing left and right

These scenarios are simulated both with and without external disturbances. The simulation platform information is summarized in Table 5.1. Videos of the simulations presented in this chapter are available on YouTube[1].

A combined safety radius of 20 m is used in the simulations presented here[2]. This is well below what should be used in a real situation, but illustrates the VO algorithm performance.

[1]http://bit.ly/1G07k18

[2]The length of the vessels used in the simulation is ca. 8 m. The safety radius is individually configurable for each vessel.

Table 5.1: Simulation platform description.

Computer	Dell Optiplex 990
Processor	Intel Core i5-2500 @ 3.30 GHz × 4
Memory	Samsung DDR3 1333 MHz 4GB × 2
Operating system	Ubuntu 14.04 “Trusty Tahr”
ROS version	“Indigo Igloo”
Guidance system version	1.1.0

Table 5.2: Simulation parameters

(a) Overtaking and crossing from the right

	ASV		Ship 1		Ship 2	
	East [m]	North [m]	East [m]	North [m]	East [m]	North [m]
Initial position	0.0	0.0	150.0	150.0	0.0	100.0
Waypoint 1	0.0	300.0	-150.0	150.0	0.0	300.0

(b) Overtaking, crossing and head-on

	ASV		Ship 1		Ship 2	
	East [m]	North [m]	East [m]	North [m]	East [m]	North [m]
Initial position	150.0	0.0	150.0	0.0	150.0	150.0
Waypoint 1	0.0	0.0	0.0	0.0	-150.0	150.0
Waypoint 2	0.0	300.0	0.0	300.0		
Waypoint 3	0.0	0.0				

(c) Crossing left and right

	ASV		Ship 1		Ship 2	
	East [m]	North [m]	East [m]	North [m]	East [m]	North [m]
Initial position	0.0	0.0	150.0	300.0	-150.0	50.0
Waypoint 1	0.0	150.0	-50.0	200.0	-150.0	200.0
Waypoint 2	-150.0	300.0	0.0	0.0	100.0	300.0

5.1 Ideal Conditions

For the simulations in this section it is assumed ideal conditions, *i.e.*, no external disturbances such as wind, current or waves.

5.1.1 Overtaking and Crossing from the Right

In this scenario, the ASV is moving in a straight line from south to north when a slow moving vessel appears in front of it. At the same time, another vessel comes in from the right such that the ASV needs to avoid two vessels at once. This is the easiest of the scenarios, considering that only a simple starboard maneuver is needed for avoiding the other two vessels. The parameters used for the vessels are found in Table 5.2a and the simulation is shown in Figure 5.4.

From Figure 5.4 it is immediately clear that the system behaves very well in this situation. The ASV signals its intentions early by altering its course starboard, thus adhering to COLREGs rule 13 and 15 (Sec. 3.5). The ASV passes both vessel with a margin of about 20 m. Still, one should note that the ASV cuts it close in front of the overtaken vessel and should ideally keep a larger safety distance (see discussion in Sec. 6.2.2).

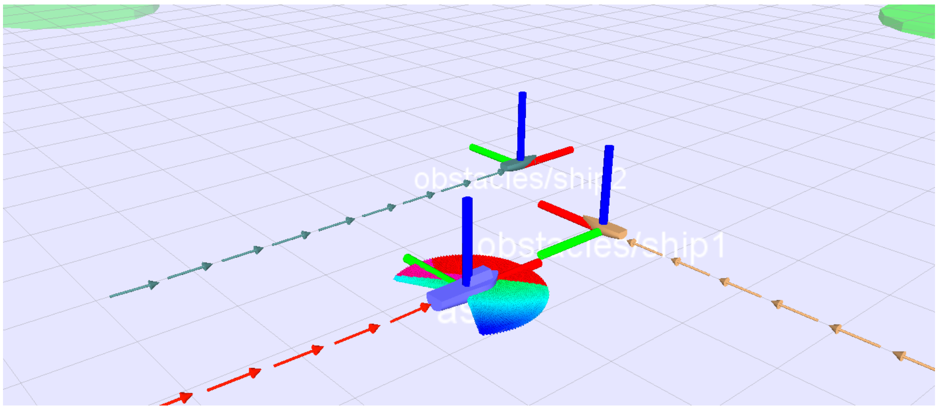


Figure 5.1: ROS: “Overtaking and crossing from the right” screenshot. The green circles seen in the upper left and right corners are waypoint indicators.

5.1.2 Overtaking, Crossing and Head-on

This scenario tests all the COLREGs situations where the ASV is the give-way vessel. The ASV needs to perform an overtaking, a crossing from right and a head-on maneuver – in that order. Moreover, this scenario tests the algorithms ability to track the COLREGs status of other vessels over time as the first vessel changes its status from an overtaken vessel to a head-on vessel . The simulation parameters are summarized in Table 5.2b and the resulting simulation is shown in Figure 5.5.

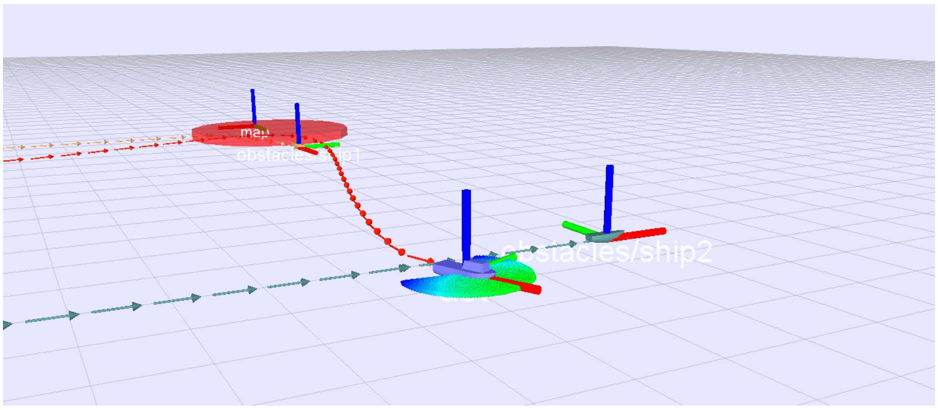


Figure 5.2: ROS: "Overtaking, crossing and head-on" screenshot.

The algorithm tackled this situation flawlessly (Fig. 5.5). If there is something to comment, it is that the ASV could have turned the opposite way at the second waypoint.

5.1.3 Crossing Left and Right

This is a more complex scenario, where the obstacle ships alter their courses and blocks the path for the ASV. This causes the COLREGs situation to change as the situation progresses. Due to the location of the waypoints, the ASV is also "locked in". This scenario tests the algorithm's ability to execute more intricate maneuvers. The parameters used for the vessels are found in Table 5.2c and the simulation is shown in Figure 5.6.

From the simulation (Fig. 5.6) some of the versatility of the VO method becomes apparent. As the ASV approaches its first waypoint, it alters it course slightly

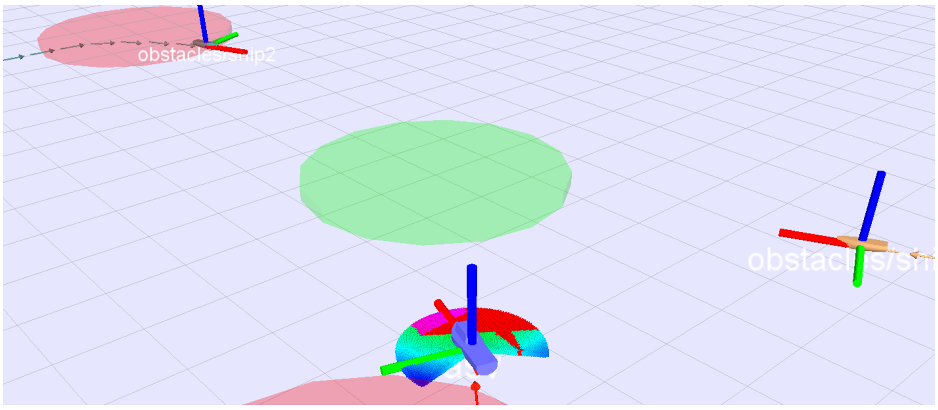


Figure 5.3: ROS: “Crossing right and left” screenshot.

starboard whilst slowing a bit down to pass the oncoming vessel from the upper right. Now, the vessel previously on a parallel course has changed its course and is now crossing from left. Therefore, to avoid collision the ASV slows down to almost halt and passes behind the vessel coming in from the left. Just as the first vessel is passed, the second changes its course heading towards our own. The ASV then speeds up and passes behind the second vessel as well and reaches its final waypoint successfully. This test shows how the VO method responds to a changing environment. Some of the disadvantages of the method are also demonstrated in this simulation, in particular the fact that the method does not consider the vessel dynamics. In Figures 5.6f–i, the ASV gets closer than the 20 m safety radius due to the VO controller expecting the ASV to instantaneously follow any velocity vector. Nevertheless, the ASV avoids any collision and adheres to all COLREGs regulations.

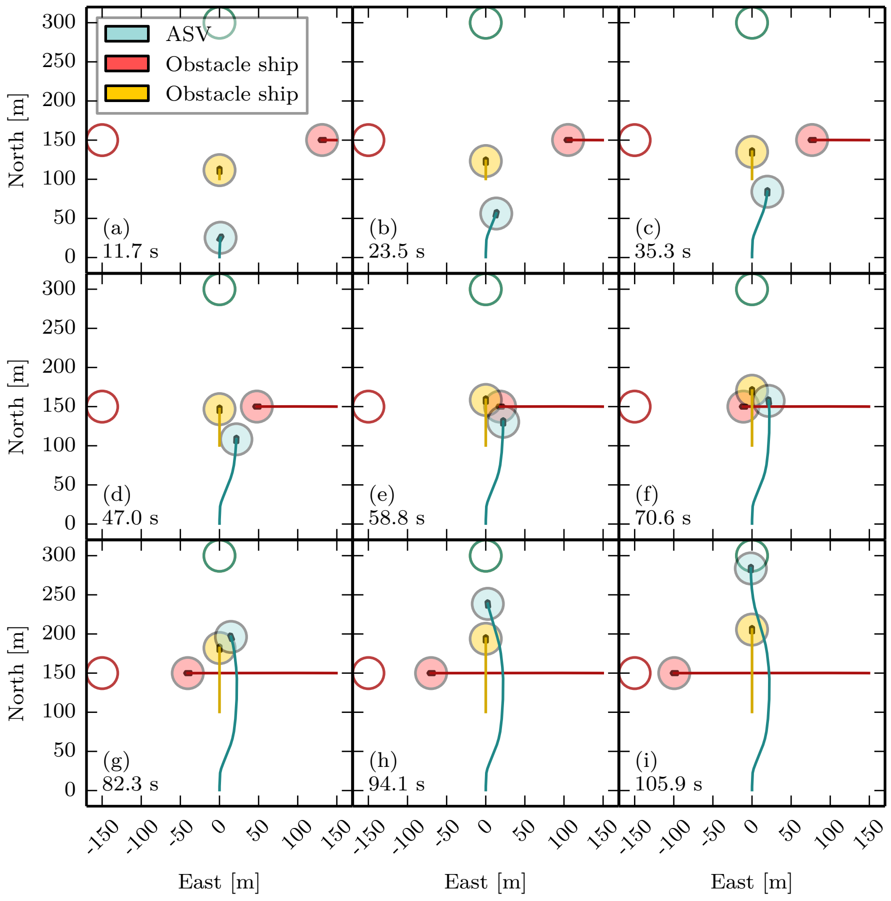


Figure 5.4: Overtaking and crossing from right. In the first two plots, we see clearly that the ASV has started the avoidance maneuver. It steers effortlessly away from the other vessels. Note that the inscribed circles around the vessels have a radius of 20 m, therefore even though it might seem like the ASV passes the red vessel (coming in from the right) very closely, it keeps a distance of 20 m.

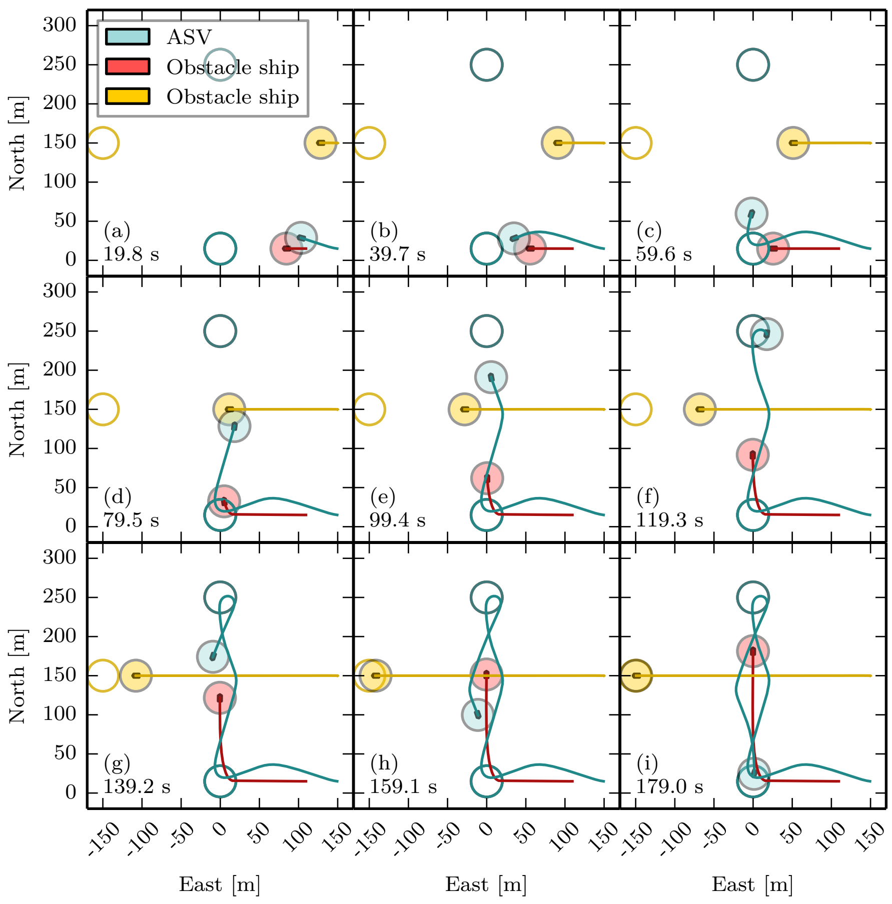


Figure 5.5: Overtaking, crossing and head-on. The ASV is the give-away vessel in all three main COLREGs scenarios: overtaking, crossing from right and head-on. It handles all situations very good. Note the clear and distinct avoidance maneuvers.

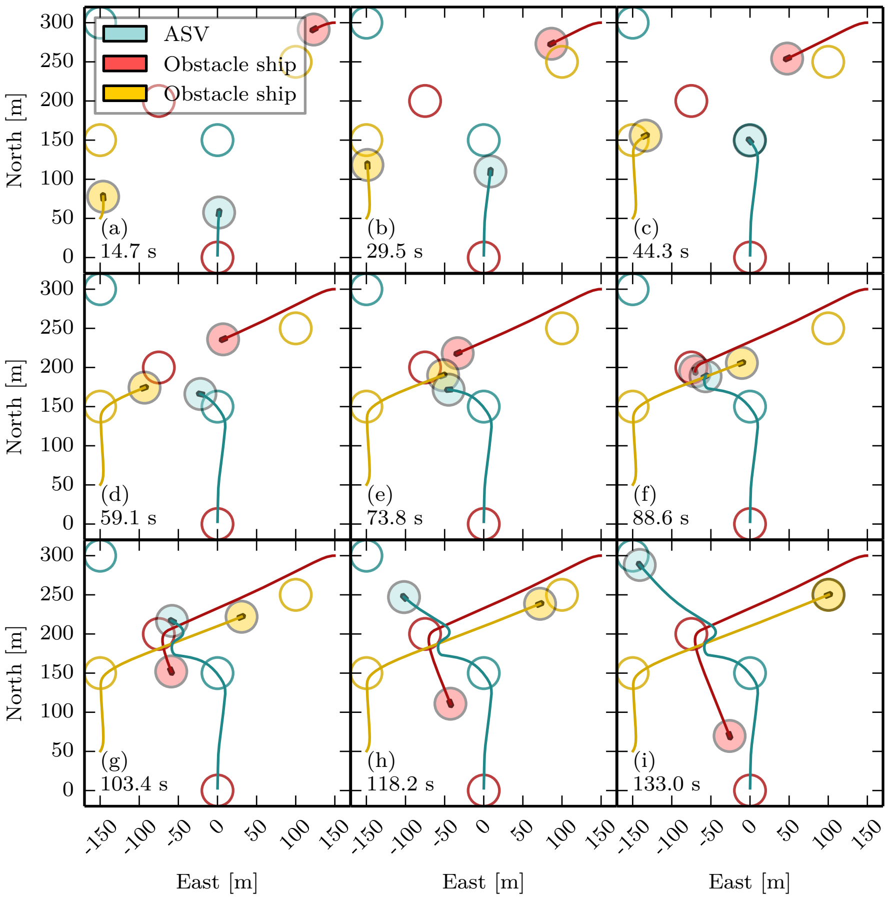


Figure 5.6: Crossing right and left. The vessel starts its avoidance maneuver for the vessel approaching from the right, but when it reaches it first waypoint, the vessel on a parallel course changes its course and becomes a crossing vessel (from left). This causes the ASV to slow down and wait for both vessels to pass before continuing to its last waypoint.

5.2 Wave Disturbances

Now we consider the same scenarios, but with the addition of wave disturbances. This addition causes unpredictable forces to act upon all the vessels. The parameters used in the wave model (Sec. 3.1.2) is summarized in Table 5.3.

Table 5.3: Wave model parameters. The parameters are tuned based on recommendations in Fossen (2011) and vessel response during simulations.

Parameter	Value	Parameter	Value	Parameter	Value
$K_{w,X}$	4693.9	λ_X	0.12	$\omega_{0,X}$	0.80
$K_{w,Y}$	6750.1	λ_Y	0.15	$\omega_{0,Y}$	0.90
$K_{w,N}$	7078.4	λ_N	0.10	$\omega_{0,N}$	0.80

5.2.1 Overtaking and Crossing from the Right

In Figure 5.8 the response of the VO algorithm is seen when waves are added to the simulation. The wave forces and moment acting on the vessels are given in Figure 5.7.

As the simulation shows (Fig. 5.8), the ASV displays very satisfactory performance despite repeatably being struck by wave forces of several thousand Newton. The oscillatory motion caused by the waves leads to some fluctuation in the control output from the VO controller. Interestingly, due to the low-pass characteristic of the vessel dynamics, the vessel is unable to follow such a rapid change in set-points, diminishing the wave effects.

5.2.2 Overtaking, Crossing and Head-on

In Figure 5.10 the response of the VO algorithm is seen when waves are added to the simulation. The forces acting on the vessels are given in Figure 5.9.

As the simulation progresses, the effect of waves on the yaw of the ASV increase significantly until it peaks at around 5 kNm at $t \approx 120$ s (Fig. 5.9). Bear in mind that the waves does not only affect the ASV directly through forces and moments, but also via the shifting velocity vectors it induces in the other vessels. By altering the heading of the other vessels, the factor for which the VO method is most sensitive, the waves may indirectly give rise to additional oscillatory motion.

5.2.3 Crossing Left and Right

This scenario is the most sensitive to disturbances, because of the small margins. The added wave forces and moments are found in Figure 5.11 and the simulation in Figure 5.12.

In general, the ASV complete this scenario with satisfactory performance. It passes behind the right-crossing vessel with a rather small margin, yet it manages to avoid both vessels and reach the goal. Note that the left-crossing vessel is the give-way vessel in this situation and disobeys the COLREGs regulations.

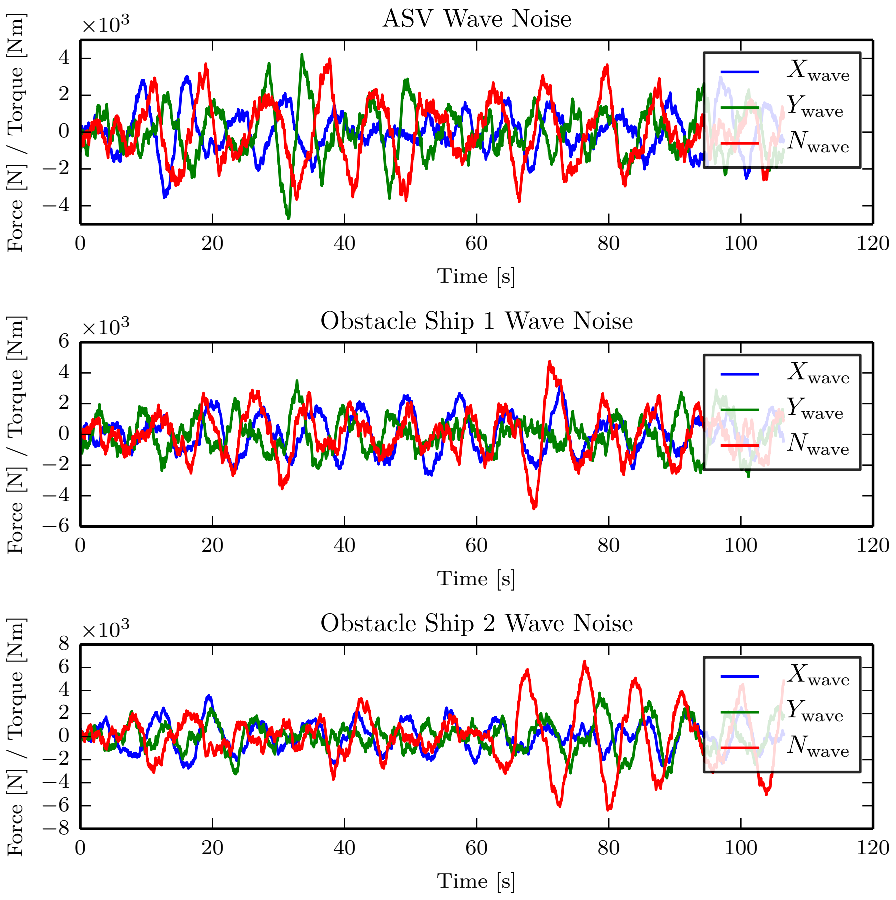


Figure 5.7: Overtaking and crossing from right wave noise. The vessels are exposed for up to 6 kNm of torque in yaw and about 3-4 kN in surge and sway, corresponding to very rough sea.

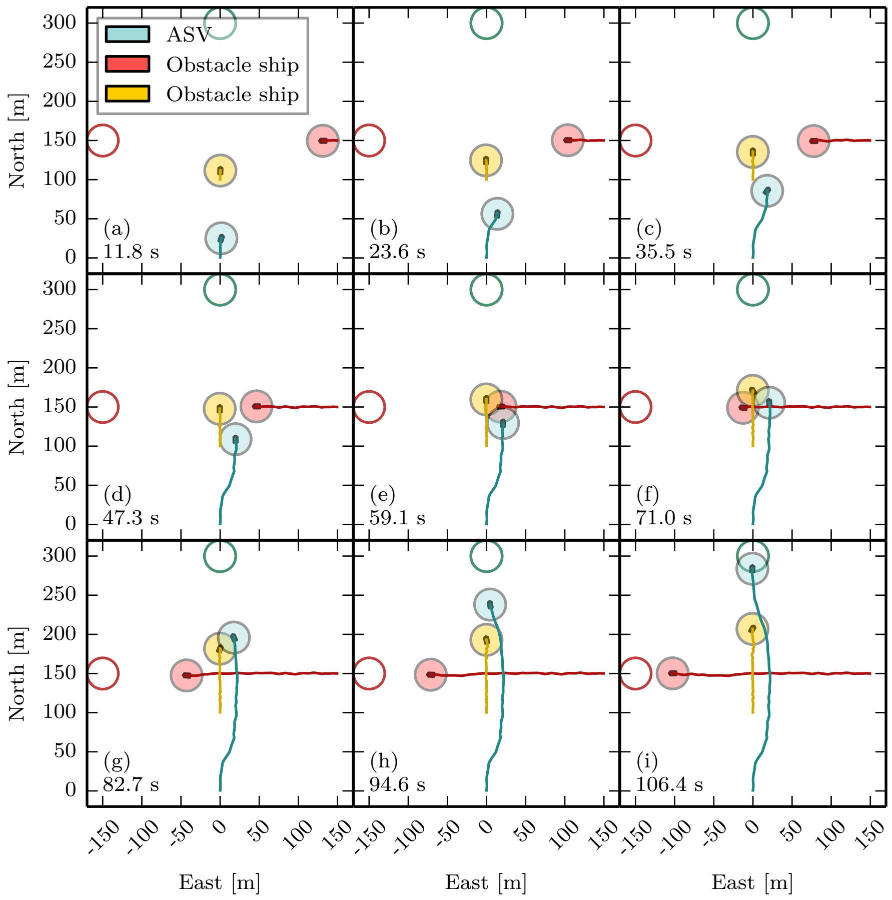


Figure 5.8: Overtaking and crossing from right with wave disturbances. Despite the rough sea state, the ASV does not have any difficulties passing the two vessels while still complying to COLREGs and maintaining safe distance.

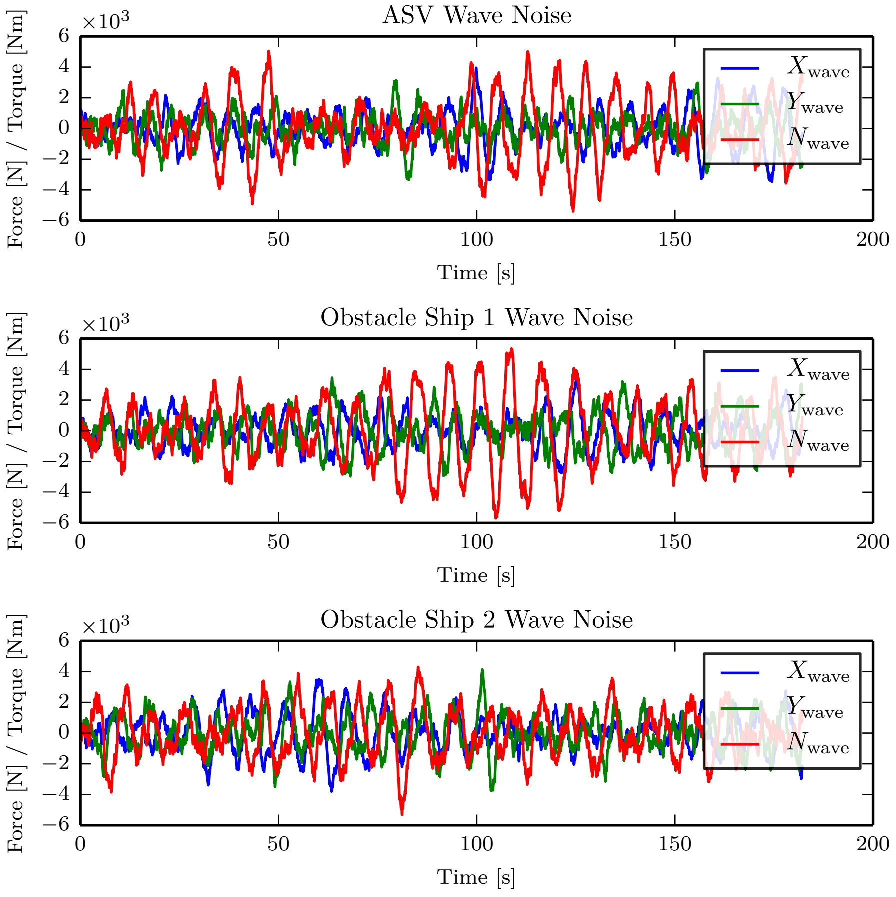


Figure 5.9: Overtaking, crossing and head-on wave noise.

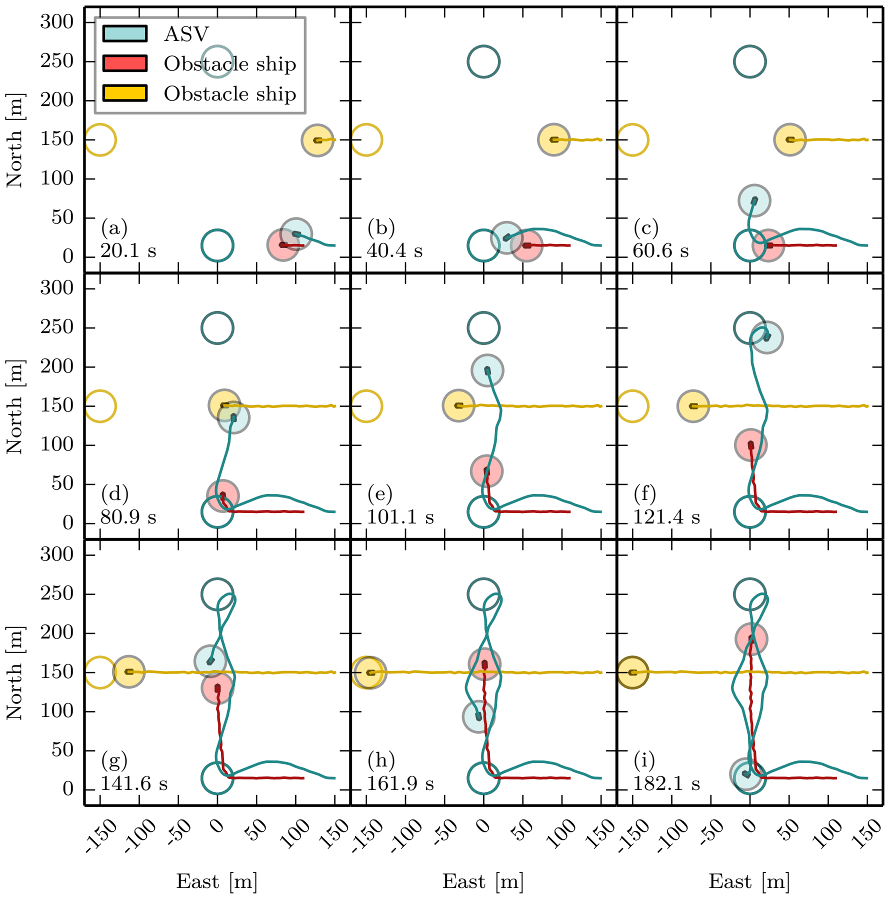


Figure 5.10: Overtaking, crossing and head-on with wave disturbances. The ASV safely avoids all vessels.

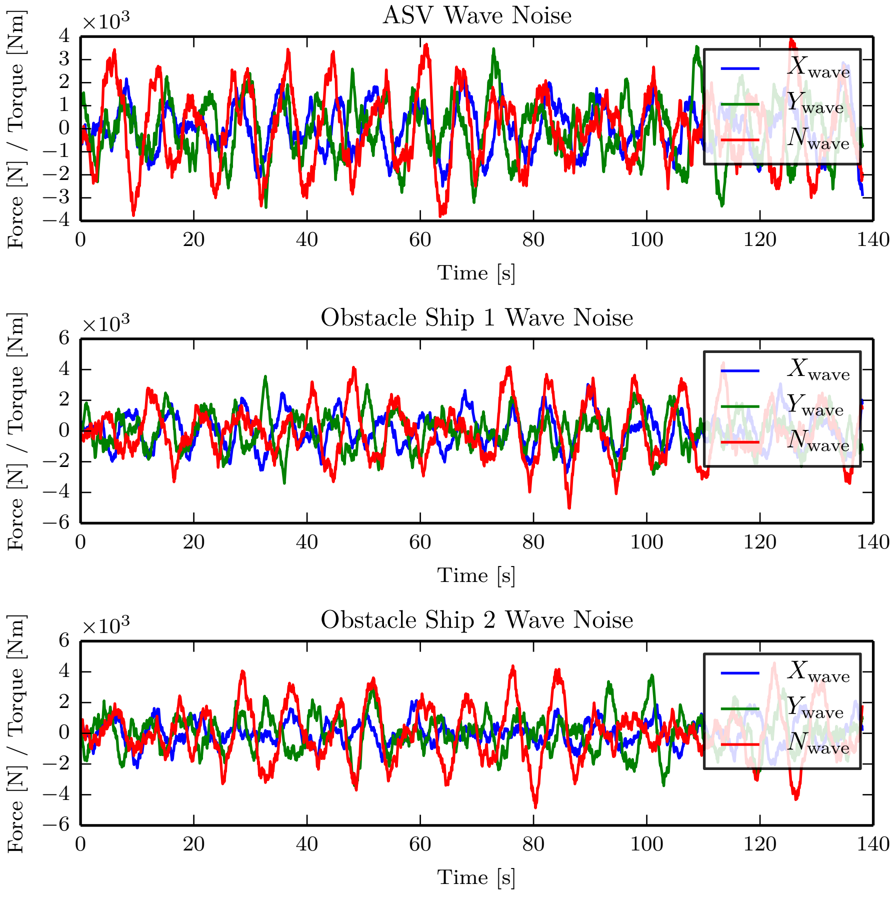


Figure 5.11: Crossing left and right wave noise.

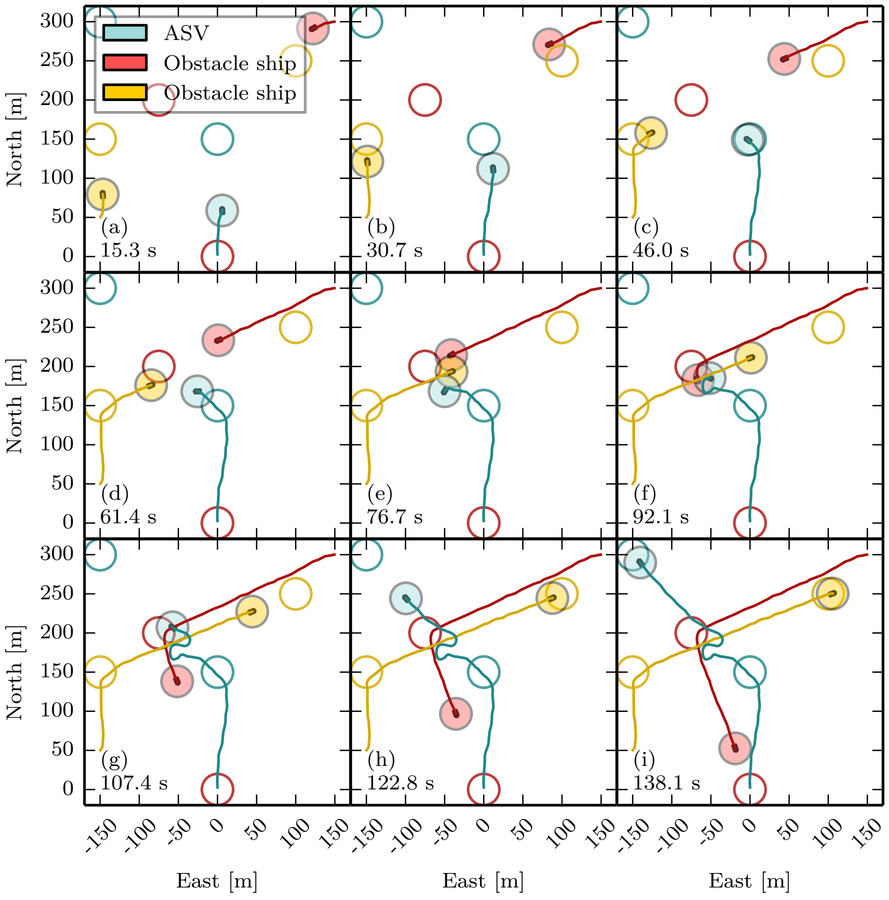


Figure 5.12: Crossing left and right with wave disturbances. Overall satisfactory performance in this scenario. While it waits for the first vessel to cross from left, it is tossed left and right by the waves. Just as the first vessel is passed, the second heads straight towards the ASV. Nevertheless, it manages to safely pass both vessels.

Chapter 6

Discussion

The simulation results (Chap. 5) demonstrate the capability of the VO algorithm for collision avoidance. It is immediately clear that the method tackles all main COLREGs situations (overtaking, head-on and crossing) with ease and performs overall very well in more complex scenarios. The inherent simplicity of the method provides predictable and robust maneuvering.

What is interesting however, is to identify the situations where the algorithm does not perform as well. This is the focus of the following sections, where actions to mitigate these challenges will also be discussed.

6.1 Static Obstacle Oscillations

When approaching static or near static obstacles, a situation where the algorithm is unable to chose the side to pass the obstacle might arise (Fig. 6.1). The vessel will start to move in an increasingly oscillatory pattern as the controller switches the side to pass on. It should be noted that this is not a problem with the VO method itself, but an effect of the combination with LOS for path following. When the passing side for the obstacle is undefined, the increasing LOS velocity-path relative angle causes the optimal set-point to switch between the opposing boundaries of the VO cone (Fig. 6.2).

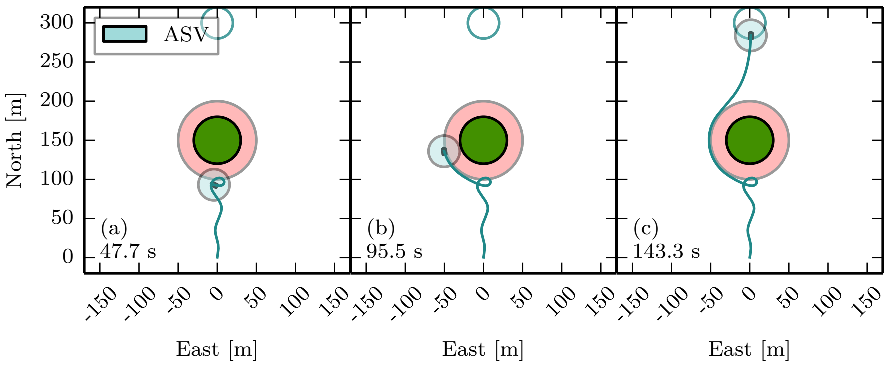


Figure 6.1: The velocity obstacle experiences oscillations near static obstacles. The VO controller is unable to choose which side to pass the obstacle and eventually the vessel enters the illegal region (safety buffer) around the obstacle.

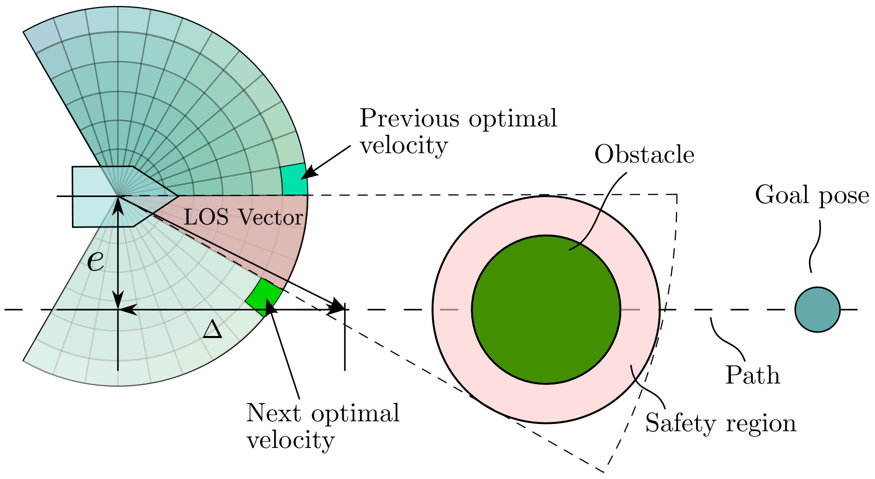


Figure 6.2: Velocity obstacle oscillations explained. The LOS controller tries to force the vessel to return to its path, unaware of the obstacle obstructing the path. As the cross-track error, e, increases, the optimal set-point (as seen by the VO controller) will switch to the opposite side of the VO cone – changing the passing side.

6.1.1 Solutions

Define a Lower Bound on the Lookahead Distance

One way to mitigate this problem is to define an upper bound on the size of unknown static and dynamic obstacles and from that find a lower bound on the lookahead distance, Δ. Ideally, as the VO planner is a local method (Sec. 3.3.1), it should be paired with a global planner in order to not be trapped in local minima and better plan missions and routes. Thus, large known (static) obstacles, such as islands, is assumed to be accounted for.

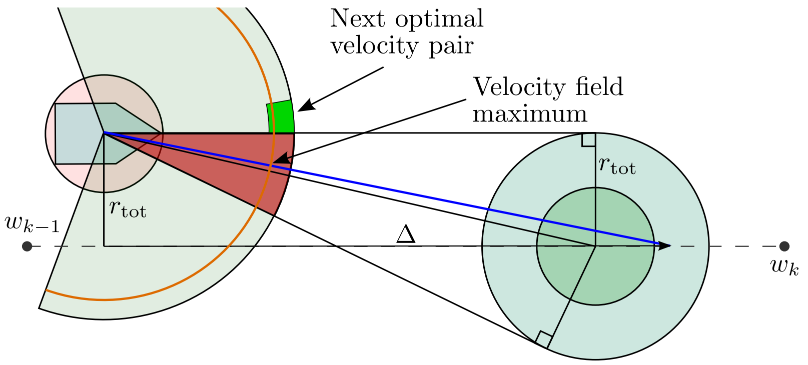


Figure 6.3: Lower bound for LOS lookahead distance. Which side of the center line of the VO cone the optimal velocity (the intersection between u_d and ψ_d given by LOS) is determines the next selected velocity. If the lookahead distance is chosen such that is always greater than the maximum padded radius of the unknown obstacle and the distance the VO controller "sees" combined, oscillatory switching is avoided.

The problem arises if a sufficiently large *unknown* obstacle appears, either a ship at rest or other unknown obstacles. Using an upper bound on the size of the unknown object we may define a lower bound on the lookahead distance (Fig. 6.3). Assuming a safety radius r_{obs} for the obstacle and a ASV safety radius r_{ASV} the padded radius is the combined radius of these, *i.e.*,

$$r_{\text{tot}} = r_{\text{ASV}} + r_{\text{obs}}. \tag{6.1}$$

The LOS lookahead distance lower bound may then be defined as

$$\Delta > \Delta_{\text{min}} = \lambda_{\text{max}} + r_{\text{tot}}, \tag{6.2}$$

where $\lambda_{\max}$ is the largest distance the algorithm "sees" (Sec. 4.4.2). Using this method, the vessel is able to pass the obstacle safely (Fig. 6.4).

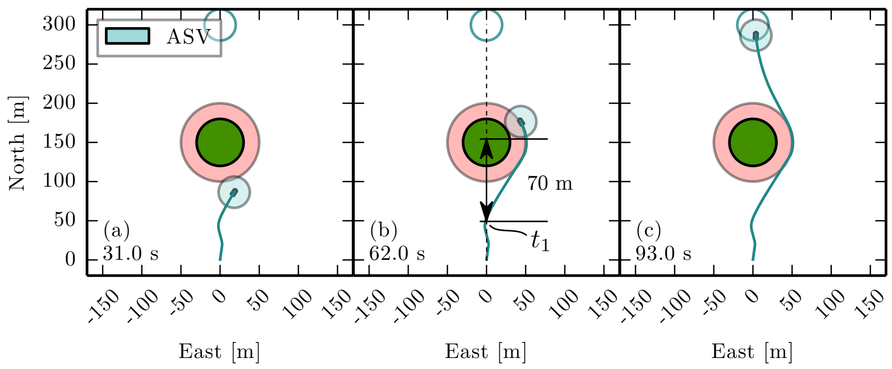


Figure 6.4: Oscillations reduced when $\Delta > r_{\text{obs}} + r_{\text{ASV}}$. Here $r_{\text{obs}} = 30$ m, $r_{\text{ASV}} = 20$ m and $\lambda_{\max} \approx 140$ m. $\Delta = 70$ m is chosen lower than the bound found in Equation (6.2) to illustrate the effect in Figure 6.3. Notice that at time t_1, the lookahead vector passes the center of the obstacle and the oscillations stop.

Unfortunately, the bound on the lookahead distance may be very restrictive and this solution does not take the initial choice of passing side into account. In addition, a large lookahead distance will result in slower convergence to the path once the obstacle is passed.

"Pick a Side"

Another approach is to simply "pick a side", *i.e.*, as the vessel approaches the obstacle it chooses to either pass the obstacle on the left or right hand side (Fig. 6.5). This is commonly known as *edge following* or *wall following* (Borenstein & Koren, 1989). This is quite easy to implement for the VO method as one only need to choose the passing side and define the velocities in the opposing side of the VO cone as "illegal", similarly to the COLREGs behavior.

Other Solutions

A third method is to reformulate the optimization problem (Eq. (4.6)) to a similar form as the DW objective function (Fox et al., 1997). One could, for example,

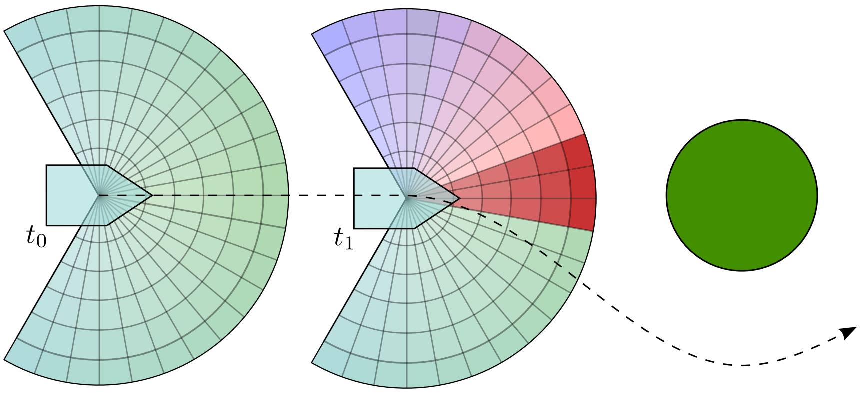


Figure 6.5: Passing a static obstacle by "picking a side". At time t_1 the ASV detects the obstacle and defines on side of the VO cone as "illegal" by adding a significant cost to those velocities.

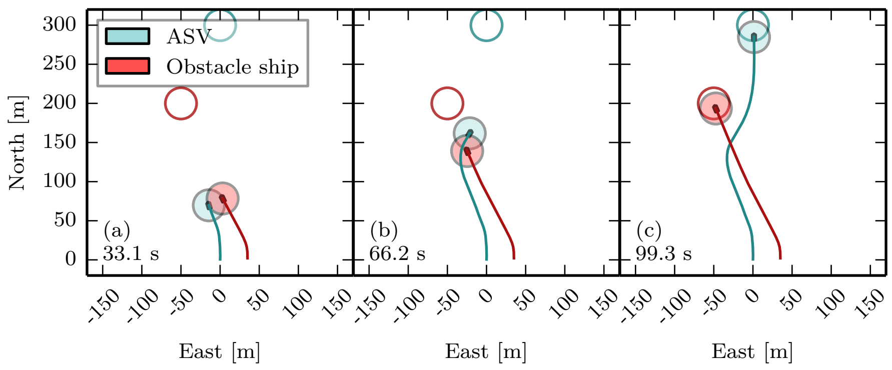
add a cost associated with changing course. However, with such a scheme much care has to be taken, as the added cost may make it less desirable to change course even if the ASV is on a collision course.

It may also be a solution to abandon the LOS controller completely during an evasive maneuver and re-enable it when the maneuver is complete, *e.g.*, when path is again directly visible for the ASV. One could also argue that if a sufficiently large static obstacle is detected on the path, a higher-level planner should be invoked and re-plan a route past the obstacle.

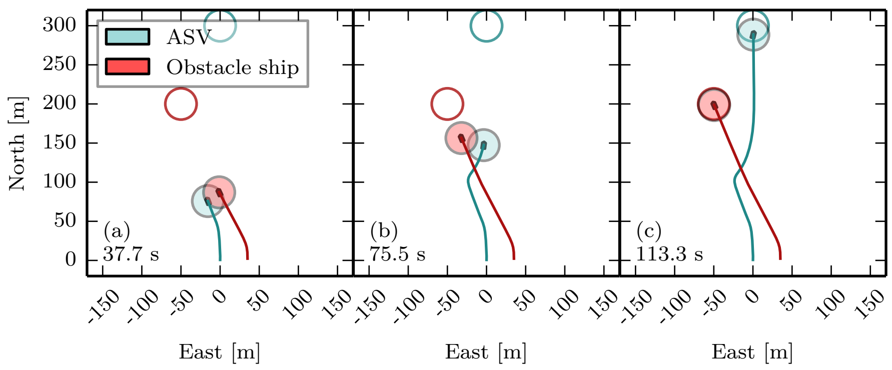
6.2 COLREGs Challenges

6.2.1 In Between Rules

The ASV may find itself on the boundary between two COLREGs situations, *e.g.*, locked between a crossing from right and an overtaking situation. Initially, the VO controller handles this situation poorly (Fig. 6.6a). The initial implementation monitored the COLREGs situation using only the relative bearing alone, thus regarding a crossing situation to be over when the ASV passes the beam of another ship. An updated controller does not change the COLREGs situation until the other vessel is passed safely (Sec. 6.2.2). Now, the ASV does not pass in front of the other, but the dead-locked situation remains (to some extent).



(a) Not maintaining initial COLREGs situation



(b) Maintaining initial COLREGs situation

Figure 6.6: Locked in a crossing from right situation. In this situation the ASV is on boundary of a crossing from right situation. If the control system determines the COLREGs situation by relative bearing alone (a), the vessel speeds in front of the other, violating COLREGs. Alternatively, by maintaining the initial COLREGs situation, the ASV passes the other vessel on the correct side (b).

A similar situation may occur if another vessel speeds in front of our own such that the situation changes from a crossing from left situation to an overtaking situation (Fig. 6.7). It is not necessarily straight forward to determine the right course of action here. The other vessel is initially the give-way vessel, yet the ASV may also be regarded as an overtaking vessel.

Ideally, these situations should be recognized by the guidance system and some

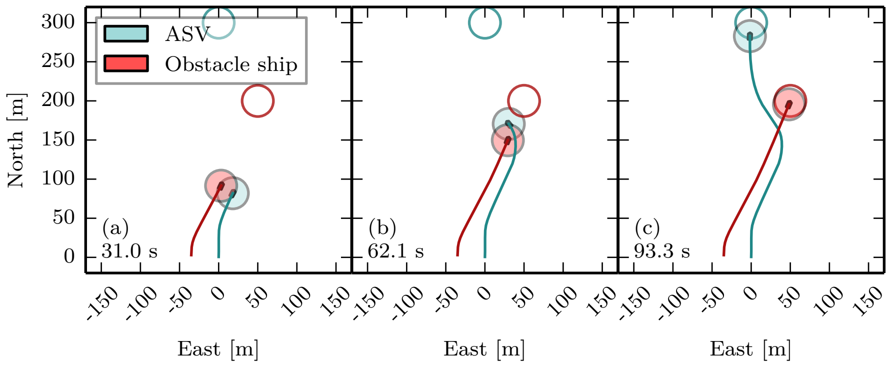


Figure 6.7: Locked in a crossing from left/overtaking situation. The obstacle ship speeds in front of the ASV, causing a deadlock where the ASV is in between the overtaking and crossing from left situation. Finally, the ASV crosses in front of the other vessel to reach its waypoint.

mitigating action taken. One approach is to monitor the relative bearing between the vessels together with the velocity-path relative angle, χ_r, to see if the bearing remains constant and χ_r increases. If this is the case, the deadlocked situation is a fact and the ASV needs to take some evasive maneuver, *e.g.*, reduce its speed and pass behind the other vessel.

6.2.2 COLREGs Ambiguity

A recurring problem whilst developing a maritime collision avoidance system is the imprecise language and definitions in COLREGs (Lund, 2008). For instance, in an overtaking situation, the situation is deemed to be over when “she is finally past and clear” (Sec. 3.5.1). The rules also repeatedly use unquantifiable phrases such as keeping “a safe distance”.

Duration of COLREGs Maneuver

As mentioned in Section 3.5.2, the COLREGs situation does not necessarily change when the relative bearing enters a new zone. Therefore, the current implementation does not consider an overtaking situation to be over until the relative bearing between the vessels is between $[-15°, 15°)$, *i.e.*, within the head-on zone (Fig. 3.11). For the crossing from right situation, the situation is deemed to be over when the relative bearing is below $-112.5°$, *i.e.*, when the relative bearing enters the overtaking zone.

6.2.3 Clearance During Overtaking

An issue with the current system is that it passes the overtaken vessel with not a great deal of clearance (Fig. 6.8). This is caused by properties of both the VO controller and the LOS controller.

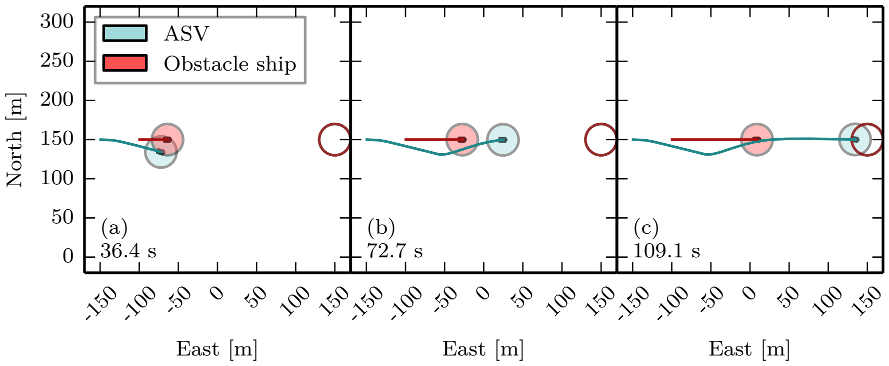


Figure 6.8: Insufficient clearance during overtaking scenario. When overtaking another vessel, the ASV shall keep away of the vessel being overtaking until it is finally past and clear.

The distance the ASV keeps from the other ship should at all times be greater than their combined safety radius, however because the VO controller expects discrete jumps in velocity, this distance is slightly reduced.

To mitigate this problem, one could use a more comprehensive configuration space and not approximate the vessels as disc shaped objects. Specifically, by approximating the vessels as box-shaped instead and expanding the footprint of the obstacles in their traveling direction, better clearance may be achieved. This adds to the computational complexity as the VO region needs to be found using the Minkowski sum. This sum results in a convex polygon for which the tangent lines with respect to the ASV position needs to be found (Fig. 3.12a). Once the tangent lines $\mathbf{p}_{AB,\,\text{left}}$ and $\mathbf{p}_{AB,\,\text{right}}$ has been identified, Equation (3.32) may be used to determine if a velocity is inside the VO cone.

Chapter 7

Future Work

Following the discussion, it is apparent that there is room for further improvement and more work outside of the scope of this thesis. In this section, some of the possible extensions to this thesis will be mentioned and recommendations for further development of the collision avoidance system as well.

7.1 Simulator

The simulator may be extended in several ways. Expecting accurate simulations when using a first-order Euler method may be naive, and changing the integration method should be considered. This change is expected to be relatively straight forward, in particular for an explicit method such as the Runge-Kutta 4^{th} order.

One could also consider increasing the complexity of the simulator by extending the vessel model, modeling the disturbances better (wind, waves, etc.) and extending it with sensor models. Thus effectively turning it into something similar to a Hardware-in-the-Loop (HIL) simulator (Fig. 7.1). This may greatly ease the transition from the simulation environment to an experimental setup.

7.2 Global Planner

Today, the GNC system is without a global planner. If the vessel is to become truly autonomous, this should be implemented. In several practical applications

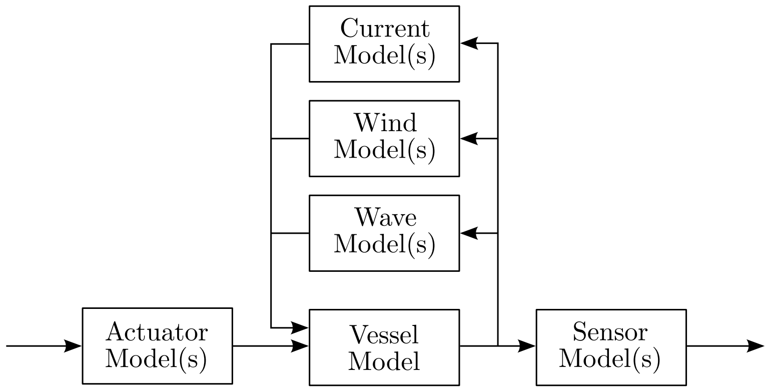


Figure 7.1: Extensions to the simulator node. By adding a more realistic interface to the simulator node, the system can be tested more rigorously. The actuator model(s) should provide an interface mimicking the actual actuator(s) and similarly, the output of the sensor model(s) should mimic the real sensors. For example, the sensor model(s) should simulate Global Positioning System (GPS) position and velocity measurements, with the position given as latitude, longitude and height.

a hybrid scheme with global path planning and local collision avoidance is preferred (Marder-Eppstein et al., 2010; Larson et al., 2006, 2007; Loe, 2008) and is recommended here as well.

The global planner could preferably take (some of) the ASV dynamics into account. Grid searching methods will often ignore the nonholonomic nature of the vessel and return paths containing, *e.g.*, several 90° turns, which are impossible for the ASV to follow. This is often be mitigated by post-processing the paths with a smoothing algorithm. The hybrid-state A* search (Montemerlo et al., 2008; Dolgov et al., 2010) is suggested as a possible candidate and is summarized in this section. The reader is referred to Dolgov et al. (2010) for more details.

7.2.1 A* Search

The A* search (Hart et al., 1968) provides a very flexible paradigm for path planning. The basic method itself is very simple and given in Listing C.1. It is immensely popular and has inspired several similar algorithms, such as D* (Stentz, 1994), Field D* (Ferguson & Stentz, 2005), Theta* (Daniel, Nash, Koenig, & Felner, 2010) and many more. The algorithm is a *best-first* search that searches through a set of interconnected nodes and uses heuristics to guide its search. Its flexibility stems from its indifference towards how nodes are explored and use of heuristics.

Heuristics or a heuristic function, is a way of ranking alternatives in a search algorithm. A heuristic function $h(n)$ gives an estimate of the cost of reaching the goal from a given node n. The function is said to be admissible if it does not overestimate the cost of reaching the goal. Furthermore, it is said to be monotonic (or consistent) if for two adjacent nodes x, y, with the distance $d(x, y)$, the following holds

$$h(x) \leq d(x, y) + h(y) \tag{7.1}$$

A* is a *complete* algorithm, *i.e.*, if a valid path to the goal exists, the algorithm will find it.

The search graph may be represented in several ways (Campbell et al., 2012), but an occupancy grid is the most commonly used.

7.2.2 Hybrid-state A* Search

Hybrid-state A* is, as the name suggests, based on the A* methodology, but incorporates vehicle dynamics in the node expansion (Fig. 7.2). This ensures that paths are drivable, an important property for non-holonomic vehicles. Moreover, the hybrid-state A* method searches in three dimensions (x, y, θ) to keep track of the vehicle pose at a given node.

The algorithm differs from the traditional A* in how it explores neighboring nodes in the search grid. It associates each grid cell with a continuous state $\mathbf{s} = (x,\, y,\, \theta)^\top$, starting at the initial state $\mathbf{s}_0 = (x, y, \theta)_0^\top$ it forward simulates the system for a given steering action

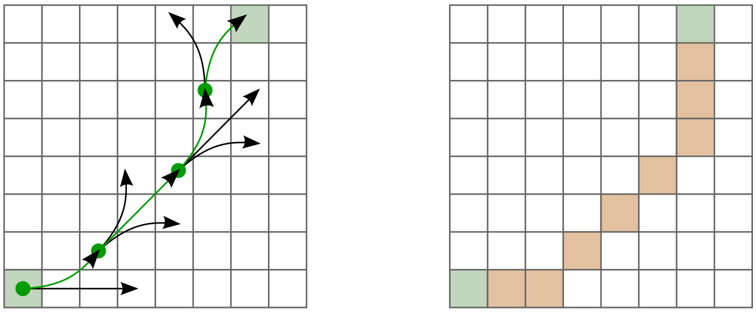
$$\delta\theta_i \in [\delta\theta_{\min},\, \dots,\, \delta\theta_{\max}]. \tag{7.2}$$

The forward simulation of node k gives a set of states

$$S_k = \{\mathbf{s}_{k,0},\, \dots,\, \mathbf{s}_{k,i},\, \dots,\, \mathbf{s}_{k,N}\}, \tag{7.3}$$

each within a grid cell. If a cell has already been visited, but the state associated with it has a higher cost than the newly expanded state $\mathbf{s}_{k,i}$, it is replaced by $\mathbf{s}_{k,i}$ and re-prioritized on the A* open set.

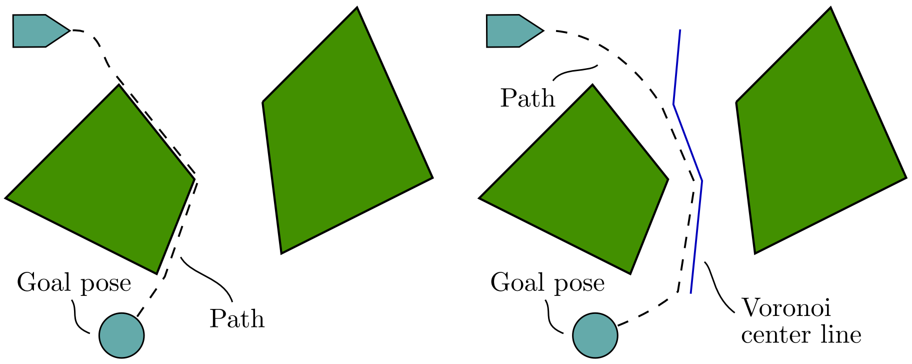
The hybrid-state A* search shares some of the same challenges as the traditional A* search and other similar algorithms. In particular, contour hugging is a



(a) Hybrid-state A* (b) Traditional A*

Figure 7.2: A* search methods. The hybrid-state A* (a) considers a starting pose $\mathbf{s}_0 = (x, y, \theta)_0^\top$ and applies a steering action for each node expansion in the search, simulating the vehicle motion for a given forward velocity v_0. The traditional A* (b), on the other hand simply considers moving to the (at most) eight neighboring nodes at any node.

problem (Fig. 7.3). Dolgov et al. (2010) suggest using a potential field, deemed the Voronoi field, to guide the algorithm away from edges. This method is applicable to other heuristic-based method as well and could also be used to guide an RRT method.



(a) Contour hugging phenomena (b) Avoiding contour hugging using the Voronoi method

Figure 7.3: Using the Voronoi field to alleviate contour hugging. The Voronoi field assigns a higher cost to deviating from the center line.

7.3 Velocity Obstacle Controller

Although the VO controller performs very good, there are several improvements possible.

7.3.1 Better ROS Integration

The ROS distribution features a navigation stack, *i.e.*, a generalized framework for integrating global and local controllers together with other high-level behavior[1]. The VO controller could benefit from an integration with this framework. This could be done by writing the VO controller as a `base_local_planner`[2] plugin. In the ROS distribution several planners are already implemented in this framework, allowing the VO method to be coupled with one of these.

7.3.2 Velocity Obstacles as a Filter

In the current implementation, the VO method acts as a filter on a velocity field by defining regions of non-COLREGs compliant and collision velocities. The velocity field itself is a simple (u, ψ) grid with a quadratic cost function. This approach completely ignores the dynamics of the vessel, with the advantage of simplicity and platform independence at the cost of inaccuracy. For ships, with slow dynamics and sway motion, this inaccuracy may prove fatal. In particular, at close quarters (Figs. 5.6 and 5.12) neglecting the ship dynamics leads to risky maneuvers. The potential danger may be reduced by increasing the safety regions around the obstacle ships, however this may restrict the ASV's mobility.

The VO method could be used to filter velocity candidates for another method. Consider, *e.g.*, the Dynamic Window (DW) approach (Fox et al., 1997). This is a well known and popular method, especially in mobile robotics. By simplifying the equations of motion for a synchro-drive WMR, it can be shown that for a given velocity pair (u_i, r_j) the vehicle will follow a circular arc if $r_j \neq 0$ and otherwise a straight line. Based on different criteria, an arc is selected and the corresponding velocity pair is applied as a set-point to the velocity controllers and the process is repeated after Δt seconds. For the velocity pair (u_i, r_j), the heading Δt seconds along its trajectory will be

$$\psi_{i,j} = \psi + r_j \Delta t, \tag{7.4}$$

[1]http://wiki.ros.org/move_base
[2]http://wiki.ros.org/base_local_planner

giving a velocity vector of

$$\mathbf{v}_{i,j} = \begin{bmatrix} u_i \cos \psi_{i,j} \\ u_i \sin \psi_{i,j} \end{bmatrix}. \tag{7.5}$$

The velocity $\mathbf{v}_{i,j}$ may now be tested against the properties found in Section 3.6.2 (Fig. 7.4).

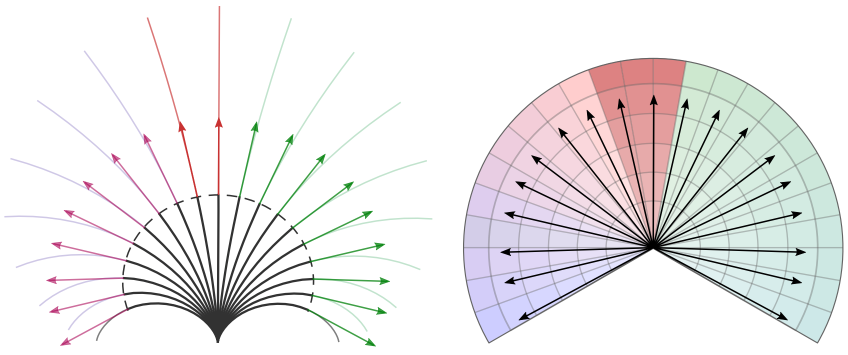

(a) DWA velocity arcs (b) Velocity vectors in the VO field

Figure 7.4: Using the Velocity Obstacle algorithm as a filter for the Dynamic Window method. The dotted arc in (a) signifies the position along an arc after Δt seconds. At this point we find the velocity vectors $\mathbf{v}_{i,j}$ and may test them against the velocity obstacle regions (b).

These properties may very well be a highly efficient way of incorporating dynamic obstacles as well as COLREGs compliance. It may also improve the run-time of the method by eliminating velocity arcs early on. On the simulation platform used in this thesis (Tab. 5.1) and with the current implementation, VO method itself has a run-time of approximately 10-30 ms depending on the amount of dynamic obstacles present.

It is possible modify the DW method to increase accuracy in the presence of sway motion (Loe, 2008; Eriksen, 2015).

7.4 Stability

The question of stability for the collision avoidance method has not been considered in this thesis. As far as the author is aware, there is no formal proof of stability for the VO method and the possibility of such a proof should be explored. The ILOS method however, is proven to be stable (Caharija, 2015). Thus, it should be noted that without any obstacles present, the VO controller simply follows the set-points of the ILOS controller and is therefore stable in this situation.

7.5 Experimental Testing

Experimental testing adds an important dimension to the validity of an algorithm. In a real environment, the method is tested for its robustness against all the factors a simulation is unable to incorporate. The implemented system is ready for testing on a real platform. One of the goals when working on this thesis was to test the system on a physical platform, but due to critical sensor failure during testing, full-scale tests were unfortunately not completed.

7.5.1 Practial Considerations

There are numerous challenges to be tackled in a practical implementation of an ASV. This section seeks to briefly highlight the main challenges and others' solution to these.

One of the most difficult challenges in an autonomous system is to accurately sense and interpret the current situation. Variables such as the ASV's state together with an estimate of other vessels' states are critical to know.

Several authors (Benjamin et al., 2006; van den Berg et al., 2011a) assume the vehicles or vessels share information about position, orientation and velocity. Others (Savvaris et al., 2014) rely solely on Automatic Identification System (AIS) data. This is probably not sufficient in most real-world scenarios. The AIS update rate may be between two seconds and three minutes and is only required for passenger ships (regardless of size) and international ships with gross tonnage of 300 or more (Wikipedia, 2015a).

Sensors

In practice, a wide range of sensors is necessary for an accurate portrayal of the vessel state and its surroundings. The types of sensors may be divided into two parts: sensors for estimating the vessel's own state or sensors for estimating other vessels' states. The main sensors recommended are:

- Global Positioning System (GPS) receiver(s)
- Inertial Measurement Unit (IMU)
- Radar(s)
- Stereo-camera(s)
- Automatic Identification System (AIS)
- Lidar

State Estimation

There are multiple algorithms in existence today for state estimation, the most popular being extensions to the common Kalman filter, such as the EKF or UKF, or nonlinear observers (Fossen, 2011; Vik, 2014). The ROS standard libraries already contain implementations of several popular localization algorithms[3], but most of these are tailored towards a WMR application and thus less suited for a maritime vessel. The reader is referred to Fossen (2011), Vik (2014) and references therein for more on observer design and localization methods for ocean vehicles.

Obstacle Detection

Reliably detecting and recognizing obstacles, in particular dynamic obstacles, is in general difficult. Other systems have used a combination of radar, lidar and computer vision (cameras) to detect other ships (Elkins et al., 2010; Huntsberger et al., 2011; Wang et al., 2011). One may also combine the estimated position and velocities of other ships with data and metadata provided by AIS. It is important to note that AIS data alone is not very reliable for obstacle tracking, *e.g.*, Loe (2008) had faulty heading estimates during experimental testing, deteriorating the obstacle avoidance algorithm's performance.

[3]http://wiki.ros.org/robot_localization

Chapter 8

Conclusion

In all, this thesis has presented a guidance and collision avoidance system capable of avoiding collisions at sea whilst complying with the rules and guidelines specified in COLREGs. Based on a comprehensive literature study, the VO method was chosen as the basis for collision avoidance. The GNC system together with an extensive vessel simulator has been implemented in ROS, providing a solid foundation for further development. Through simulations in diverse scenarios in combination with discussions of situations proving extra challenging, a thorough analysis of the VO method as a basis for a collision avoidance system is given.

The strength of the VO method lies in its simplicity. It assumes nothing about the dynamics of the vehicle, making it versatile and possible to deploy on a wide range of vessel types without modification. But, within its simplicity lies its weakness as well. The naive approach to the velocity field renders close quarter navigation difficult and dangerous. In particular, neglecting the non-holonomic nature of the vessel results in reckless maneuvers. Conservative safety buffers are used to mitigate this, but this restricts the ASV's capability to perform more high-fidelity maneuvers.

For better behavior in these situations, it is recommended that the naive velocity field is replaced by a set of velocities that more accurately portrays the dynamic capabilities of the vessel. This inevitably adds to the complexity of the control system and requires better models of the dynamics of the vessel. Having said that, even simple approximations of the vessel motion is expected to improve the accuracy and safety of the motion planning.

Appendix A

Robotic Operating System (ROS)

This appendix provides a very brief introduction to the main ROS concepts and conventions. The following sections are based on information from ROS.org[1] and the reader is encouraged to check it out for more detailed information and tutorials on ROS.

A.1 Introduction

ROS is not an operating system in the usual sense, but a software framework or middleware for developing robot applications. Similar to a conventional operating system, ROS provides services such as hardware abstraction, device control and implementation of commonly used functionality (Wikipedia, 2015d). The system is divided in nodes (or processes) that share information through message passing (Fig. A.1).

ROS is designed to be as *thin* as possible. Libraries written for ROS should be ROS agnostic with clean functional interfaces (Open Source Robotics Foundation, 2015). ROS is also language independent, C++, Python and Lisp are fully supported and experimental libraries exists for Java and Lua. ROS currently only runs on Unix-like platforms and Ubuntu is the primary supported operating system.

[1]http://wiki.ros.org

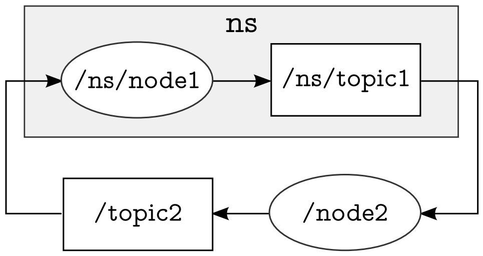


Figure A.1: Sharing information between processes using message parsing. In this example, the process (or node) `/ns/node1` is under the namespace `ns` and subscribes to the topic `/topic2` published by `/node2` and publishes to `/ns/topic1`.

A.2 ROS Concepts

A.2.1 ROS Filesystem Level

- ROS *packages* are the most common way code is organized in ROS. A package is comparable to a computer program, it may contain one or more nodes, configuration files, etc.

Listing A.1: Common package file structure.

```
package/
  include/
    package/
      package.h
      package_node.h
  launch/
  nodes/
    package_node.py
  src/
    package.cpp
    package_node.cpp
  cmakelists.txt
  package.xml
```

- *Package manifests* (`package.xml`) provide metadata about a package, such as: name, version, licence, etc.
- ROS uses *CMake* (`CMakeLists.txt`) to configure compilation of ROS packages.
- *Message (msg) types*, stored in `my_package/msg/MyMsg.msg`, defines the data structure for messages sent in ROS.

- *Services (srv) types*, stored in `my_package/srv/MySrv.srv`, defines the request and response data structures for services in ROS.

A.2.2 ROS Computation Graph Level

- *Nodes* are processes that perform computation. Examples of node tasks: collect IMU data, perform localization (*e.g.* Kalman filter) or control motor.
- The ROS *Master* provides name registration and lookup to the rest of the computation graph.
- The *Parameter Server* stores data by key in the *Master*. This allows nodes to fetch parameters at launch or even during run-time.
- Each node may publish or subscribe to *topics*. There may be multiple concurrent publishers and subscribers to a single topic. In general, publishers and subscribers are unaware of each others' existence.
- *Services* provides synchronous communication through request/reply interaction. Services are defined by a pair of message structs: one for the request and one for the reply.
- ROS provides *bags* as a format for saving and playing back ROS message data.

A.3 Conventions

The conventions used in the ROS ecosystem are collected in ROS Enhancement Proposals (REPs).

A.3.1 Position, Orientation and Velocity

In ROS, the position and orientation of a robot is known as its *pose*. The position is given as $\mathbf{p} = (x, y, z)^\top$ coordinate and the orientation as a quaternion $\mathbf{q} = (\varepsilon_1, \varepsilon_2, \varepsilon_3, \eta)^\top$. The velocity is split into linear velocity $\mathbf{v} = (u, v, w)^\top$ and angular velocity $\boldsymbol{\omega} = (p, q, r)^\top$.

A.3.2 Reference Frames

REP 103 (Foote & Purvis, 2010) describes the ROS standard units of measure and coordinate conventions. ROS coordinate frames are right handed. The body related coordinate frame standard is

- x forward,
- y left and
- z up.

The global reference coordinate systems use the East, North, Up (ENU) convention:

- x east,
- y north and
- z up.

Furthermore, REP 105 (Meeussen, 2010) describes the three main frames used for mobile platforms: *map*, *odom* and *base_link*.

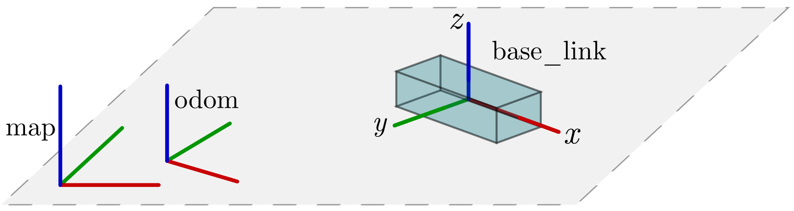


Figure A.2: ROS coordinate frames. The red, green and blue axes are the x, y and z axes respectively.

The map frame is a global world fixed frame, with its z-axis pointing upwards. This frame is discontinuous, *i.e.*, the transform from this frame to the odom frame may "jump" discretely as new measurements arrive.

The odom frame is also a world fixed frame, however this frame can drift over time without any bounds. It is therefore not very useful as a long-term global reference. However, as it is guaranteed to be continuous and is accurate as a short-term local reference, it is useful for local sensing and acting. The frame is typically computed based on information from wheel odometry, visual odometry or an Inertial Measurement Unit (IMU).

The base_link frame is rigidly attached to the mobile robot base.

In addition, REP 103 recommends defining an additional frame with a “_ned”-suffix for outdoor system where it is desirable to use the NED convention.

Appendix B

Implementation Details

The implementation of the system can be found at GitHub: https://github.com/thomsten/ros_asv_system, and is known there as the `ros_asv_system`. The main packages in the repository are:

- `asv_ctrl_vo`: an implementation of the VO algorithm (Fiorini & Shiller, 1998; Kuwata et al., 2011, 2014).
- `asv_msgs`: custom ROS message types.
- `asv_obstacle_tracker`: node simulating tracking obstacles. Subscribes to all available obstacle ship states and publishes a single list of them together with various metadata.
- `asv_path_trackers`: implementation of the pure pursuit and LOS guidance algorithms.
- `asv_simulator`: simulates the equations of motion of a general 3-DOF surface vessel.

Note that the implementation follows ROS conventions, *e.g.*, it uses the East, North, Up (ENU) coordinate system.

B.1 Installation Guide

In order to launch the system, there are a couple of requirements that needs to be in place. The user should be on a computer running Ubuntu with a working distribution of ROS. In this thesis Ubuntu 14.04 "Trusty Tahr" was used with ROS "Indigo Igloo", but the system is likely compliant with other distribution versions. For detailed installation instructions, tutorials and more, see http://wiki.ros.org/.

A brief step-by-step guide is given here.

1. Install a ROS distribution, *e.g.*,

   ```
   ~$ sudo apt-get install ros-indigo-desktop-full
   ```

   and run the following command (and optionally add it to your `.bashrc`):

   ```
   ~$ source /opt/ros/indigo/setup.bash
   ```

2. Create a catkin workspace[1]

   ```
   ~$ mkdir -p ~/my_catkin_ws/src
   ~$ cd ~/my_catkin_ws/src
   ~$ catkin_init_workspace
   ```

3. Download the `ros_asv_system` package from GitHub

   ```
   ~$ git clone https://github.com/thomsten/ros_asv_system/ --recursive
   ```

4. Build the packages

   ```
   ~$ catkin_make
   ```

5. Run a launch file, *e.g.*,

   ```
   ~$ roslaunch asv_system overtaking_and_crossing.launch
   ```

If there are package dependency issues, the packages are most likely available through the Ubuntu package manager or `rosinstall`[2]. For example, the `map_server` package is not distributed as a part of the `ros-indigo-desktop-full` package, but can be installed with

```
~$ sudo apt-get install ros-indigo-map-server
```

B.2 Package Details

B.2.1 ASV System

The `asv_system` package is for the most part a meta-package containing the top-most launch-files. In its `launch/`-folder one may find the necessary launch files for

[1]http://wiki.ros.org/catkin/workspaces
[2]http://wiki.ros.org/rosinstall

launching the different scenarios. An example launch file is given in Listing C.3. Note the flexibility of using nested launch files with parameter arguments: the same launch file is used to instantiate three `asv_simulator` instances.

The `asv_system` package contains launch files for all the scenarios in Chapter 5. A scenario is launched by simply typing

```
~$ roslaunch asv_system scenario_name.launch
```

in a terminal. This will launch the system together with RVIZ for visualization.

B.2.2 ASV Simulator

The `asv_simulator` package implements a general 3-DOF surface vessel simulator.

In `asv_simulator.cpp` the equations of motion of a general 3-DOF surface vessel according to Section 3.1 is implemented. All parameters of the system may be reconfigured using the ROS Parameter Server at launch. Default parameters are found in `config/parameters/viknes.yaml`.

Wave Filter

In `wave_filter.cpp` the wave filter described in Section 3.1.2 is implemented. The filter is implemented as its own class containing a single state space representation of the filter in Equation (3.11). By instantiating three instances of this class, the generalized wave forces may be generated independently.

When generating pseudo-random noise, it is important to be aware of which *seed* is used to initialize the pseudo-random sequence. In C++, simply using `rand()` will generate the same sequence of pseudo-random numbers each time the code is run, as the seed defaults to `srand(NULL)`. A common way to seed the pseudo-random generator is to use the current time, *i.e.*, `srand(time(NULL))`. However, this is not suitable in this application as several filters are instantiated almost simultaneously, causing all filters to be seeded with the same value. This is overcome by seeding the generator with the number of processor cycles since last reset using an assembler call. The wave filter uses the `rand48()` family of functions instead of the default `rand()`. Specifically, `drand48()` is used, which returns values in the range [0.0, 1.0). The `rand48()` functions are chosen mainly because the `rand48()` functions stores the state of the pseudo-random sequence in an internal buffer, rendering it thread safe.

Listing B.1: The assembler call `rdtsc` returns the number of processor cycles since last reset.

```
unsigned long long rdtsc(){
  unsigned int lo,hi;
  __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
  return ((unsigned long long)hi << 32) | lo;
}
```

B.2.3 Velocity Obstacle Controller

The `asv_ctrl_vo` package implements a local collision avoidance controller using the VO method as described in Section 4.4. A stripped down version of the main update loop is given in Listing C.4.

B.2.4 Path Trackers

The `asv_path_trackers` package implements two nodes: a pure pursuit and an ILOS controller. The code is based on the simulator from Stenersen (2014), but adopted for ROS. Most parameters are configurable from the parameter server.

Appendix C

Source Code Examples

Listing C.1: A* implementation using Python. Courtesy of Patel (2015).

```python
def a_star_search(graph, start, goal):
    # Open set sorted by lowest estimated cost
    open_set = PriorityQueue()
    open_set.put(start, 0)
    came_from = {}
    cost_so_far = {}
    came_from[start] = None
    cost_so_far[start] = 0

    while not open_set.empty():
        current = open_set.get()

        if current == goal:
            break

        for n in graph.neighbors(current):
            new_cost = cost_so_far[current] + graph.cost(current, n))
            if n not in cost_so_far or new_cost < cost_so_far[n]:
                cost_so_far[n] = new_cost
                priority = new_cost + heuristic(goal, n)
                open_set.put(n, priority)
                came_from[n] = current

    return came_from, cost_so_far
```

Listing C.2: Algorithm for checking static obstacles. By traversing the surge velocity in decreasing order the algorithm may mark all subsequent surge velocities as collision free if a velocity pair is collision free.

```cpp
void VelocityObstacle::checkStaticObstacles() {
  double px0  = asv_pose_[0];
  double py0  = asv_pose_[1];
  double psi0 = asv_pose_[2];

  double u   =  MAX_VEL_;
  double psi = -MAX_ANG_ + psi0, psi_max = MAX_ANG_ + psi0;

  double du   = -MAX_VEL_/VEL_SAMPLES_;
  double dpsi = 2.0*MAX_ANG_/ANG_SAMPLES_;
  double dt   = map_.getResolution() / MAX_VEL_; // Proportional to the grid resolution

  double px, py, dx, dy, t;

  bool velocity_ok;
  for (int psi_it = 0; psi_it < ANG_SAMPLES_; ++psi_it) {
    // Reset u
    u = MAX_VEL_;
    dx = cos(psi);
    dy = sin(psi);
    for (int u_it = VEL_SAMPLES_-1; u_it >= 0; --u_it) {
      // Reset t
      t = dt;
      velocity_ok = true;
      while (t <= T_MAX) {
        px = px0 + u*dx*t;
        py = py0 + u*dy*t;

        if (inObstacle(px, py)) {
            velocity_ok = false;
            break;
          }
        t += dt;
      }

      if (velocity_ok) {
        // This direction is ok. Mark all velocities from here as ok!
        for (int i=u_it; i >= 0; --i)
          setVelocity(i, psi_it, VELOCITY_OK);
        break;
      }
      else {
          setVelocity(u_it, psi_it, VELOCITY_NOT_OK);
      }
      u += du;
    }
    psi += dpsi;
  }
}
```

Listing C.3: Example of a launch file.

```
<launch>
  <include file="$(find asv_simulator)/launch/default.launch">
    <arg name="waypoint_file"
         value="$(find asv_simulator)/config/waypoints/overtaking_headon_crossing_asv.yaml"
     />
    <arg name="u_d" value="5.0" />
    <arg name="initial_state" value="[150.0, 15.0, 3.14, 1.,0.,0.]" />
  </include>

  <!-- Obstacle ship 1 -->
  <include file="$(find asv_simulator)/launch/default.launch">
    <arg name="waypoint_file"
         value="$(find asv_simulator)/config/waypoints/overtaking_headon_crossing_ship1.yaml
     " />
    <arg name="use_vo" value="False" />
    <arg name="vessel_model_file" value="$(find asv_simulator)/config/models/ship1.urdf" />
    <arg name="namespace" value="obstacles/ship1" />
    <arg name="shipname" value="ship1" />
    <arg name="initial_state" value="[110.0, 15.0, 3.14, 1.0, 0.0, 0.0]" />
    <arg name="u_d" value="1.5" />
  </include>

  <!-- Obstacle ship 2 -->
  <include file="$(find asv_simulator)/launch/default.launch">
    <arg name="waypoint_file"
         value="$(find asv_simulator)/config/waypoints/overtaking_headon_crossing_ship2.yaml
     " />
    <arg name="use_vo" value="False" />
    <arg name="vessel_model_file" value="$(find asv_simulator)/config/models/ship2.urdf" />
    <arg name="namespace" value="obstacles/ship2" />
    <arg name="shipname" value="ship2" />
    <arg name="initial_state" value="[150.0, 150.0, 3.14, 0.0, 0., 0.]" />
    <arg name="u_d" value="2.0" />
  </include>

  <node pkg="asv_obstacle_tracker"
        name="obstacle_tracker_node"
        type="obstacle_tracker_node.py"
        respawn="false"
        output="screen">
  </node>

  <node pkg="rviz"
        type="rviz"
        name="rviz"
        args="-d $(find asv_simulator)/config/rviz/three_vessels.rviz" />
</launch>
```

Listing C.4: Simplified pseudo-code for the main update function in the VO controller.

```
void VelocityObstacle::updateVelocityGrid() {
  double u0 = 0, u = 0;
  double psi0 = -MAX_ANG_ + asv_pose_[2];
  double t = 0;

  double du = MAX_VEL_/VEL_SAMPLES_, dpsi = 2*MAX_ANG_/ANG_SAMPLES_;

  Eigen::Vector2d va;
  // Rotate body-fixed veloctiy vector to ENU velocity vector
  rot2d(asv_twist_.head(2), asv_pose_[2], va);

  Eigen::Matrix2d Q; // Weighing matrix
  Q << 1.0, 0.0,
       0.0, 1.0;

  Eigen::Vector2d vref = Eigen::Vector2d(u_d_*cos(psi_d_), u_d_*sin(psi_d_));

  resetVelocityField(va, vref, Q);

  int ship_no = 0;
  std::vector<asv_msgs::State>::iterator it;
  for (it = obstacles_->begin(); it != obstacles_->end(); ++it) {
    Eigen::Vector3d obstacle_pose = Eigen::Vector3d(it->x, it->y, it->psi);
    Eigen::Vector3d obstacle_twist = Eigen::Vector3d(it->u, it->v, it->r);
    double combined_radius = RADIUS_ + it->header.radius;

    Eigen::Vector2d vb;
    rot2d(obstacle_twist.head(2), obstacle_pose[2], vb);

    Eigen::Vector2d pab = -asv_pose_.head(2) + obstacle_pose.head(2);

    // Relative bearing (Loe, 2008)
    double bearing = normalize_angle(atan2(-pab[1], -pab[0]) - obstacle_pose[2]);
    double angle_diff = normalize_angle_diff(asv_pose_[2] - obstacle_pose[2], asv_pose_[2]);

    bool collision_situation    = inCollisionSituation(asv_pose_, obstacle_pose, va, vb);
    colregs_t colregs_situation = inColregsSituation(bearing, angle_diff);

    // Based on the current COLREGs state evaluate the current situation
    handleSituation(collision_situation, colregs_situation, bearing, ship_no);

    if (!(collision_situation || apply_colregs))
      continue;

    // Collsion situation detected
    // Find velocity obstacle region
    double pab_norm = pab.norm(), alpha = 0.5*M_PI;
    if (pab_norm >= combined_radius)
      alpha = asin(combined_radius / pab_norm);

    // Left and right bounds pointing inwards to the VO. (Guy et. al. 2009)
    Eigen::Vector2d lb, rb;
    pab = pab/pab_norm;
    rot2d(pab,  alpha - 0.5*M_PI, lb);
    rot2d(pab, -alpha + 0.5*M_PI, rb);

    for (int u_it=0; u_it<VEL_SAMPLES_; ++u_it) {
      for (int t_it=0; t_it<ANG_SAMPLES_; ++t_it) {
        u = u0 + u_it*du;
        t = theta0 + t_it*dtheta;
        normalize_angle_diff(t, va_ref[1]);

        objval = getVOFieldValue(u_it, t_it);

        if (collision_situation && inVelocityObstacle(u, t, lb, rb, vb)) {
          setVelocity(u_it, t_it, VELOCITY_NOT_OK);
        }
        else if ((apply_colregs &&
                  violatesColregs(u, t, obstacle_pose, vb))) {
          setVelocity(u_it, t_it, VELOCITY_VIOLATES_COLREGS + objval/2.0);
        }
        else {
          /// ALREADY SET
          // setVelocity(u_it, t_it, objval);
        }
      }
    }
    ++ship_no;
  }
}
```

Bibliography

Adams, S. D. (2015, March). ReVolt: next generation short sea shipping. *The Guardian*. Retrieved from http://www.theguardian.com/dnv-gl-partner-zone/2015/mar/18/revolt-next-generation-short-sea-shipping

Benjamin, M. R., Leonard, J. J., Curcio, J. A., & Newman, P. M. (2006). A method for protocol-based collision avoidance between autonomous marine surface craft. *Journal of Field Robotics*, *23*(5), 333–346

Bibuli, M., Bruzzone, G., Caccia, M., Indiveri, G., & Zizzari, A. (2008). Line following guidance control: Application to the Charlie unmanned surface vehicle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '08* (pp. 3641–3646).

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

Borenstein, J. & Koren, Y. (1991). The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, *7*(3), 278–288.

Brown, R. G. & Hwang, P. Y. C. (2012). *Introduction to Random Signals and Applied Kalman Filtering* (4th ed.). John Wiley & Sons, Ltd.

Caharija, W. (2015). *Integral Line-of-Sight Guidance and Control of Underactuated Marine Vehicles* (PhD Thesis, Norwegian University of Science and Technology).

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

Chakravarthy, A. & Ghose, D. (1998). Obstacle avoidance in a dynamic environment: a collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *28*(5), 562–574

Colito, J. (2007). *Autonomous mission planning and execution for unmanned surface vehicles in compliance with the Marine Rules of the Road.* (Master's thesis, University of Washington).

Corfield, S. J. & Young, J. M. (2006). Unmanned surface vehicles – game changing technology for naval operations. In *Advances in Unmanned Marine Vehicles* (pp. 311–328). Control, Robotics & Sensors. Institution of Engineering and Technology.

Damas, B. & Santos-Victor, J. (2009, October). Avoiding moving obstacles: the forbidden velocity map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 4393–4398).

Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research*, *39*(1), 533–579.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010, January). Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *The International Journal of Robotics Research*, *29*(5), 485–501

Dragland, Å. (2014, August). Skip uten kaptein bak roret. *Gemini.* Retrieved from http://gemini.no/2014/08/skip-uten-kaptein-bak-roret/

Elkins, L., Sellers, D., & Monach, W. R. (2010). The Autonomous Maritime Navigation (AMN) project: Field tests, autonomous and cooperative behaviors, data fusion, sensors, and vehicles. *Journal of Field Robotics*, *27*(6), 790–818.

Eriksen, B.-O. H. (2015). *Horizontal Collision Avoidance for Autonomous Underwater Vehicles* (Master's Thesis, Norwegian University of Science and Technology).

Ferguson, D. & Stentz, A. (2005). Field D*: An interpolation-based path planner and replanner. In *International Symposium on Robotics Research (ISRR).*

Fiorini, P. & Shiller, Z. (1993). Motion planning in dynamic environments using the relative velocity paradigm. In *IEEE International Conference on Robotics and Automation* (pp. 560–565).

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

Flæten, S. Ø. (2014, September). Dette skipet er utslippsfritt og har ingen mennesker ombord. *Teknisk Ukeblad.* Retrieved from www.tu.no/industri/2014/09/20/dette-skipet-er-utslippsfritt-og-har-ingen-mennesker-ombord

Foote, T. & Purvis, M. (2010). *Standard Units of Measure and Coordinate Conventions.* Open Source Robotics Foundation. Retrieved May 9, 2015, from http://www.ros.org/reps/rep-0103.html

Fossen, T. I., Breivik, M., & Skjetne, R. (2003). Line-of-sight path following of underactuated marine craft. In *IFAC Conference on Manoeuvring and Control of Marine Craft* (pp. 244–249).

Fossen, T. I. (2011). *Handbook of marine craft hydrodynamics and motion control.* John Wiley & Sons, Ltd.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, *4*(1), 22–33.

Google Inc. (2014). Google Self-Driving Car Project. Retrieved November 19, 2014, from https://plus.google.com/+GoogleSelfDrivingCars

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., & Dubey, P. (2009). Clearpath: highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 177–187). ACM.

Hart, P., Nilsson, N., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107

Huntsberger, T., Aghazarian, H., Howard, A., & Trotz, D. C. (2011). Stereo vision-based navigation for autonomous surface vessels. *Journal of Field Robotics*, *28*(1), 3–18.

International Maritime Organization. (2003). *COLREG : Convention on the International Regulations for Preventing Collisions at Sea, 1972* (4th). International Maritime Organization.

Khatib, O. (1986). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, *5*(1), 90–98.

Klingenberg, M. (2014). Denne roboten sørger for at Ekornes ikke flagger ut. Retrieved June 11, 2015, from http://e24.no/digital/fremtidens-arbeidsliv/denne-roboten-soerger-for-at-ekornes-ikke-flagger-ut/23260216

Koren, Y. & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation* (pp. 1398–1404).

Kuwata, Y., Wolf, M. T., Zarzhitsky, D., & Huntsberger, T. L. (2011). Safe maritime navigation with COLREGS using velocity obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 4728–4734).

Kuwata, Y., Wolf, M., Zarzhitsky, D., & Huntsberger, T. (2014). Safe Maritime Autonomous Navigation With COLREGS, Using Velocity Obstacles. *IEEE Journal of Oceanic Engineering*, *39*(1), 110–119.

Larson, J., Bruch, M., & Ebken, J. (2006). Autonomous navigation and obstacle avoidance for unmanned surface vehicles. In *SPIE Defense and Security Symposium* (Vol. 6230, pp. 7–18).

Larson, J., Bruch, M., Halterman, R., Rogers, J., & Webster, R. (2007). *Advances in autonomous obstacle avoidance for unmanned surface vehicles.* Space and Naval Warfare Systems Center. San Diego, CA.

LaValle, S. M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Department of Computer Science, Iowa State University. Ames, IA 50011 USA.

LaValle, S. M. (2006). *Planning Algorithms.* Cambridge: Cambridge University Press

Lee, S.-M., Kwon, K.-Y., & Joh, J. (2004). A fuzzy logic for autonomous navigation of marine vehicles satisfying COLREG guidelines. *International Journal of Control Automation and Systems*, *2*, 171–181.

Loe, Ø. A. G. (2007). *Collision Avoidance Concepts for Marine Surface Craft* (Project Report, Norwegian University of Science and Technology, Trondheim).

Loe, Ø. A. G. (2008). *Collision Avoidance for Unmanned Surface Vehicles* (Master's thesis, Norwegian University of Science and Technology).

Lund, T. (2008). *Safe Speed in a Fog; Ancient Rules in a Modern Age.* Retrieved from http://www.jus.uio.no/nifs/forskning/prosjekter/sjosikkerhet/ressurser/startseminar2008/tor-c-lund-bakgrunn.pdf

Majohr, J., Buch, T., & Korte, C. (2000). Navigation and automatic control of the measuring dolphin (MESSIN). In *5th IFAC Conference on Manoeuvring and Control of Marine Crafts* (pp. 405–410).

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., & Konolige, K. (2010). The Office Marathon : Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation.*

Meeussen, W. (2010). *Coordinate Frames for Mobile Platforms.* Open Source Robotics Foundation. Retrieved May 9, 2015, from http://www.ros.org/reps/rep-0105.html

Montemerlo, M., Becker, J., Suhrid, B., Dahlkamp, H., Dolgov, D., Ettinger, S., ... Thrun, S. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, *25*(9), 569–597.

Naeem, W., Sutton, R., & Chudley, J. (2006). Modelling and control of an unmanned surface vehicle for environmental monitoring. In *UKACC International Control Conference.*

Naeem, W., Xu, T., Sutton, R., & Tiano, A. (2008). The design of a navigation, guidance, and control system for an unmanned surface vehicle for environmental monitoring. In *Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* (Vol. 222, *2*, pp. 67–79).

Naeem, W., Irwin, G. W., & Yang, A. (2012). COLREGs-based collision avoidance strategies for unmanned surface vehicles. *Mechatronics*, *22*(6), 669–678.

Narrative Science. (2014). Natural Language Generation - Artificial Intelligence Platform. Retrieved November 19, 2014, from http://www.narrativescience.com/

Nord, P. (2010). *Collision-Free Path Planning for Unmanned Surface Vehicles* (Master's thesis, Norwegian University of Science and Technology).

Open Source Robotics Foundation. (2014). Robot Operating System (ROS). Retrieved February 9, 2014, from http://www.ros.org

Open Source Robotics Foundation. (2015). ROS Introduction. Retrieved April 22, 2015, from http://wiki.ros.org/ROS/Introduction

Patel, A. (2015). Implementation of A*. Retrieved June 2, 2015, from http://www.redblobgames.com/pathfinding/a-star/implementation.html

Perera, L., Carvalho, J., & Soares, C. G. (2009). Autonomous guidance and navigation based on the COLREGs rules and regulations of collision avoidance. In *International Workshop "Advanced Ship Design for Pollution Prevention* (pp. 205–216).

Rafael Advanced Defense Systems Ltd. (2010). PROTECTOR – Unmanned Naval Patrol Vehicle. Retrieved May 8, 2015, from http://www.rafael.co.il/Marketing/288-1037-en/Marketing.aspx

Rodríguez-Seda, E. J., Tang, C., Spong, M. W., & Stipanović, D. M. (2014). Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing. *International Journal of Robotics Research*, *33*(12), 1569–1592.

Savvaris, A., Oh, H. N. H., & Tsourdos, A. (2014). Development of Collision Avoidance Algorithms for the C-Enduro USV. In *The 19th World Congress – The International Federation of Automatic Control* (pp. 12174–12181).

Simmons, R. & Henriksen, L. (1996). Obstacle avoidance and safeguarding for a lunar rover. In *AIAA Forum on Advanced Developments in Space Robotics.*

SNAME. (1950). *Nomenclature for treating the motion of a submerged body through a fluid* (1st ed.). Technical and research bulletin. Society of Naval Architects and Marine Engineers.

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot Modeling and Control.* John Wiley & Sons, Ltd.

Stenersen, T. (2014). *Guidance System for Autonomous Surface Vehicles* (Project Report, Norwegian University of Science and Technology).

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation* (pp. 3310–3317).

Švec, P., Shah, B., Bertaska, I., Alvarez, J., Sinisterra, A., Von Ellenrieder, K., ... Gupta, S. (2013, November). Dynamics-aware target following for an autonomous surface vehicle operating under COLREGs in civilian traffic.

In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 3871–3878).

Švec, P., Thakur, A., Raboin, E., Shah, B. C., & Gupta, S. K. (2014). Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots*, *36*(4), 383–405

Tam, C., Bucknall, R., & Greig, A. (2009). Review of Collision Avoidance and Path Planning Methods for Ships in Close Range Encounters. *Journal of Navigation*, *62*, 455–476.

Tychonievich, L., Zaret, D., Mantegna, J., Evans, R., Muehle, E., & Martin, S. (1989). A maneuvering-board approach to path planning with moving obstacles. In *11th International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 1017–1021).

van den Berg, J., Guy, S. J., Lin, M., & Manocha, D. (2011a). Reciprocal n-Body Collision Avoidance. In *Robotics Research* (pp. 3–19). Springer.

van den Berg, J., Snape, J., Guy, S. J., & Manocha, D. (2011b). Reciprocal collision avoidance with acceleration-velocity obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3475–3482).

Vik, B. (2014). *Integrated Satellite and Inertial Navigation Systems*. Department of Engineering Cybernetics, NTNU.

Wang, H., Wei, Z., Wang, S., Ow, C. S., Ho, K. T., Feng, B., & Lubing, Z. (2011). Real-time obstacle detection for unmanned surface vehicle. In *Defense Science Research Conference and Expo (DSR)* (pp. 1–4).

Wikipedia. (2014). Velocity obstacle — Wikipedia, The Free Encyclopedia. Retrieved June 3, 2015, from http://en.wikipedia.org/w/index.php?title=Velocity_obstacle&oldid=593276110

Wikipedia. (2015a). Automatic identification system — wikipedia, the free encyclopedia. Retrieved May 19, 2015, from http://en.wikipedia.org/w/index.php?title=Automatic_Identification_System&oldid=663056059

Wikipedia. (2015b). DARPA Grand Challenge — Wikipedia, The Free Encyclopedia. Retrieved May 22, 2015, from http://en.wikipedia.org/w/index.php?title=DARPA_Grand_Challenge&oldid=646957566

Wikipedia. (2015c). International Regulations for Preventing Collisions at Sea — Wikipedia, The Free Encyclopedia. Retrieved June 3, 2015, from http://en.wikipedia.org/w/index.php?title=International_Regulations_for_Preventing_Collisions_at_Sea&oldid=663130373

Wikipedia. (2015d). Robot Operating System – Wikipedia, The Free Encyclopedia. Retrieved May 9, 2015, from http://en.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=657161474

Wilkie, D., van den Berg, J., & Manocha, D. (2009). Generalized velocity obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5573–5578).

Zhang, R., Tang, P., Su, Y., Li, X., Yang, G., & Shi, C. (2014). An adaptive obstacle avoidance algorithm for unmanned surface vehicle in complicated marine environments. *IEEE/CAA Journal of Automatica Sinica*, *1*(4), 385–396.