Guidance System for Autonomous Surface Vehicles

Thomas Stenersen

December 20, 2014

Project Report

Department of Engineering Cybernetics
Norwegian University of Science and Technology

Supervisor: Professor Kristin Y. Pettersen

Co-advisor: Øystein Engelhardtsen (DNV GL)

Problem Description

Autonomous technology is spreading across numerous applications in an unprecedented pace. Amongst others, the DARPA grand challenge for autonomous ground vehicles has given an indication of what is possible to achieve using state-of-the-art sensors in combination with cleverly designed autonomous technology. DNV GL believes the increasing use of autonomy in vehicle systems eventually will propagate to the shipping industry and that we will see an increasing level of autonomous system used on ships in order to increase safety and to reduce impact from human induced errors.

The task of this assignment is to design a guidance system for an unmanned, autonomous surface vehicle (ASV). The system can assume a sensor package able to provide a sufficient picture of the ships' surroundings in order to navigate safely. The system should be able to plan and execute a safe route through various obstacles such as archipelago and other vessels, based on information from sensors and a priori chart data.

1. Do a literature survey on autonomous surface vehicles. Describe state-of-the-art technology of guidance systems for autonomous surface vehicles (ASV).
2. Design a guidance system for an ASV capable of navigating safely in various types of waters, and to act on suddenly occurring obstacles.
3. Create a simulator for the vehicle and the surroundings, and demonstrate the capabilities of the developed system.

Abstract

The past decade, we have seen rapid development in the field of autonomous navigation. One major catalyst for this has been the DARPA Grand Challenge, a competition for autonomous ground vehicles, which has inspired research on path planning and collision avoidance. Today, the Centre of Excellence "Centre for Autonomous Marine Operations and Systems (AMOS)", the "Maritime Unmanned Navigation through Intelligence in Networks (MUNIN)" project and DNV GL seeks to develop such technology in the context of marine applications.

Robust path planning and collision avoidance is crucial for autonomous navigation. In this report, several of the most popular planning algorithms is compared. Local, global and hybrid methods has been implemented and simulated in a comprehensive custom simulation environment. These are the Virtual Force Field (VFF) method, the Dynamic Window Approach (DWA), A* search and the hybrid-state A* search together with the hybrid combination of these.

Three different scenarios has been designed to illustrate the strengths and weaknesses associated with each method. In addition to demonstrating inherent problems with local and global methods they also highlight the advantages and disadvantages with each individual algorithm.

The simulation results together with the findings in other literature are discussed and on the basis of the discussion a guidance system recommendation is given.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years, the presence of autonomous technology has increased immensely. 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). Other examples include the more general-purpose robot PR2 by Willow Garage. One catalyst for research on autonomous navigation has been the DARPA[1] Grand Challenge, a competition for autonomous mobile robotic vehicles. Today, both MARINTEK through MUNIN (Dragland, 2014) and DNV GL envisions the use of autonomous surface vehicles (ASV) for transport at sea (Fig. 1.1b) (Flæten, 2014).

(a) Google self-driving car (image courtesy of Google)

(b) DNV GL's Revolt (image courtesy of DNV GL)

For reliable autonomous navigation at sea, a robust path planning and collision avoidance system is needed and in this report several of the most popular approaches to this challenge is compared and discussed.

[1] Defense Advanced Research Projects Agency

1.1 Background

Traditionally, most research done on the topic of *planning algorithms* has been done independently in the fields of robotics, artificial intelligence and control theory. Robotics traditionally has been concerned with the geometrical constraints of motion planning (e.g. the "piano moving"-problem). Artificial intelligence has dealt with finding sequences of logical operators or actions to transform a initial world state to a goal state. Control theory is generally concerned with problems regarding stability, feedback and optimality. However, today we see these fields intersecting (LaValle, 2006), either through a top-down approach as in computer science or a bottom-up approach from control theory.

1.1.1 Literature Review

The results in this report is based on the works of many authors. The articles, theses and reports referenced in this report has been found through extensive search in Scopus, DiVA, IEEE Xplore, Google Scholar or through reference lists. When searching, keywords related to autonomy, planning algorithms, surface vehicles and more have been used.

Specific Methods

Examples of methods from robotics are: using potential fields (Khatib, 1986), the dynamic window approach (DWA) (Fox, Burgard, & Thrun, 1997) and probabilistic roadmaps (Kavraki, Svestka, Latombe, & Overmars, 1996). From computer science, we have several shortest path algorithms such as: Dijkstra's algorithm (Dijkstra, 1959) and A* (Hart, Nilsson, & Raphael, 1968). Finally, from the control theoretic framework there is, *e.g.*, line of sight (LOS) guidance algorithm (Breivik & Fossen, 2004) or the use of Fermat's spirals for path parametrization (Lekkas, 2014; Dahl, 2013). Most of these methods are designed for one specific purpose or application area, which is important to keep in mind when discussing their performance. *E.g.*, as configuration spaces in robotics are in general complex and algorithms have computational time constraints, most methods focus on effectively guide the robot in a subset of the configuration space.

Complete Solutions

For ASVs, a general and robust solution is necessary. The planning system should not only work for a subset of the possible scenarios, but tackle all situations in a well-behaved and well-defined manner.

Larson, Bruch, and Ebken (2006, 2007) proposes 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 local world model was 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), but is also similar to the dynamic window approach (Fox et al., 1997). The degree of detail in Larson et al. (2006, 2007) is unfortunately low.

In Loe (2007) several popular methods is compared and this provided a basis for the approach used in Loe (2008). A Matlab simulation environment was implemented in order to compare the algorithms. Based on the results a hybrid approach with A* guided Rapidly-exploring Random Trees (RRTs) (LaValle, 1998) for global navigation and the dynamic window algorithm for local collision avoidance was recommended. In Loe (2008), several modifications to the methods was proposed such that the system should adhere to the *International Regulations for Preventing Collisions at Sea* or simply COLREGS (International Maritime Organization, 2003) and improve their overall performance. The system was also tested in Trondheimsfjorden in cooperation with Maritime Robotics where it performed good. However, because of the nature of RRTs, the paths generated are unpredictable and suboptimal. 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 be 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 (see discussion in section 4.2.3, p. 39).

The hybrid-state A* search, presented in Montemerlo et al. (2008), Dolgov, Thrun, Montemerlo, and Diebel (2010), is a path planning method developed in connection with the DARPA Urban Challenge for the robotic vehicle "Junior". The main contribution of Montemerlo et al. (2008) is a general description of the different components in a fully autonomous robotic vehicle. In Dolgov et al. (2010), a detailed presentation of the hybrid-state A* search is given with methods of improving speed and accuracy of the solutions. It also presents a trajectory smoothing technique for optimizing the paths generated by the search. Finally, a Voronoi based method for guiding the search in semi-structured environments is presented.

Marder-Eppstein, Berger, Foote, Gerkey, and Konolige (2010) presents a system for robust autonomous navigation in an indoor office environment. It 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 in addition to simulated 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 dynamic window algorithm.

1.1.2 Project Contribution

This report provides a review of the existing technology in autonomous navigation and collision avoidance. In addition, a comprehensive, powerful and flexible simulator environment was implemented using Python with NumPy, SciPy and Matplotlib. The simulator is also set up for easily interfacing real-world systems through the Robot Operating System (Open Source Robotics Foundation, 2014). Furthermore, several guidance systems has been developed and tested, based on the virtual force field, dynamic window, A* and hybrid-state A* algorithms.

1.2 Structure of the Report

The rest of the report is organized as follows. Chapter 2 presents the different planning algorithms implemented and in Chapter 3 their performance in different scenarios is presented. A discussion of the algorithm performances is had in Chapter 4 and a conclusion based on this is given in Chapter 5. Finally, the simulator is presented in Appendix A.

Chapter 2

Planning Algorithms

LaValle (2006) introduces the notion of *planning algorithms* as a broad term referring to a common ground between the fields of artificial intelligence, robotics and control theory. In this report, most algorithms stem from robotics, computer science or control theory and will therefore be referred to as *path* or *motion planners*. Planning algorithms are often separated in local and global methods. In general, most motion planners are local and most path planners global.

2.1 Preliminaries

2.1.1 Surface Vehicle Model

In this project, a 3-DOF vessel model (Fossen, 2011) is used to describe the surface vehicle motions as it may describe the motion of a wide range of surface vehicles with sufficient accuracy. This is also the model used in Loe (2007, 2008). The vehicle states are named according to SNAME (1950). The general equations of motion for a vessel may be written according to Fossen (2011) as

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \mathbf{J}(\psi)\boldsymbol{\nu} \\ \mathbf{M}\dot{\boldsymbol{\nu}} + \mathbf{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \mathbf{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \boldsymbol{\tau} + \boldsymbol{\tau}_{ext} \end{aligned} \tag{2.1}$$

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 (2.1) is equal to the rotation

matrix

$$\mathbf{J}(\psi) \triangleq \mathbf{R}_{z,\psi}^{\top} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

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

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix}, \quad \mathbf{M}_A = -\begin{bmatrix} X_{\dot{u}} & 0 & 0 \\ 0 & Y_{\dot{v}} & Y_{\dot{r}} \\ 0 & N_{\dot{v}} & N_{\dot{r}} \end{bmatrix}, \tag{2.3}$$

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 matrix for our system is, respectively

$$\mathbf{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & -mv \\ 0 & 0 & mu \\ mv & -mu & 0 \end{bmatrix}, \tag{2.4}$$

$$\mathbf{C}_A(\boldsymbol{\nu}) = \begin{bmatrix} 0 & 0 & Y_{\dot{v}}v + Y_{\dot{r}}r \\ 0 & 0 & -X_{\dot{u}}u \\ -Y_{\dot{v}}v - Y_{\dot{r}}r & X_{\dot{u}}u & 0 \end{bmatrix} \tag{2.5}$$

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

$$\mathbf{D}(\boldsymbol{\nu}) = \mathbf{D}_L\boldsymbol{\nu} + \mathbf{D}_{NL}(\boldsymbol{\nu})\boldsymbol{\nu}, \tag{2.6}$$

where $\mathbf{D}_L$ accounts for linear damping and $\mathbf{D}_{NL}$ accounts for nonlinear damping (Eq. (2.7)).

$$\mathbf{D}_L = -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix}, \tag{2.7}$$

$$\mathbf{D}_{NL}(\boldsymbol{\nu})\boldsymbol{\nu} = -\begin{bmatrix} X_{|u|u}|u|u + X_{uuu}u^3 \\ Y_{|v|v}|v|v + Y_{vvv}v^3 \\ N_{|r|r}|r|r + N_{rrr}r^3 \end{bmatrix} \tag{2.8}$$

Environmental disturbances, *e.g.*, wind and waves, acts upon the vessel through the disturbance vector $\boldsymbol{\tau}_{ext}$. In this thesis, these effects are not considered, yielding $\boldsymbol{\tau}_{ext} = 0$.

The actuator forces are given through the generalized force vector

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_X \\ \tau_Y \\ \tau_N \end{bmatrix} = \begin{bmatrix} F_X \\ F_Y \\ l_r F_y \end{bmatrix}, \tag{2.9}$$

where τ_X and τ_Y are the generalized forces applied to the x_b and y_b **axes**, respectively. τ_N is the torque applied about the z-axis. The vessel has only two control inputs: the propeller force and rudder angle, the vessel is *under actuated* (Definition 9.3 in Fossen (2011)).

It is assumed that the vessel has low-level controllers producing the appropriate forces and moments given set-points ψ_d (or r_d) and u_d. In the simulator, feedback linearization is used, 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.

2.1.2 Configuration Spaces

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

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

$$\mathcal{QO} = \{\boldsymbol{\eta} \in \mathcal{Q} \mid \mathcal{A}(\boldsymbol{\eta}) \cap \mathcal{O} \neq \emptyset\}, \tag{2.10}$$

where $\mathcal{O} = \cup O_i$. The set of collision-free configurations, the *free configuration space*, is

$$\mathcal{Q}_{\text{free}} = \mathcal{Q} \backslash \mathcal{QO} \tag{2.11}$$

2.2 Local Methods

Local methods searches the immediate environment and are often referred to as *reactive* or *dynamic*. They are often used in responsive collision avoidance schemes as most are orders of magnitude faster than global methods. However, as local methods only considers a subset of the configuration space, they are susceptible to local minima and may get "stuck" (Fig. 2.1).

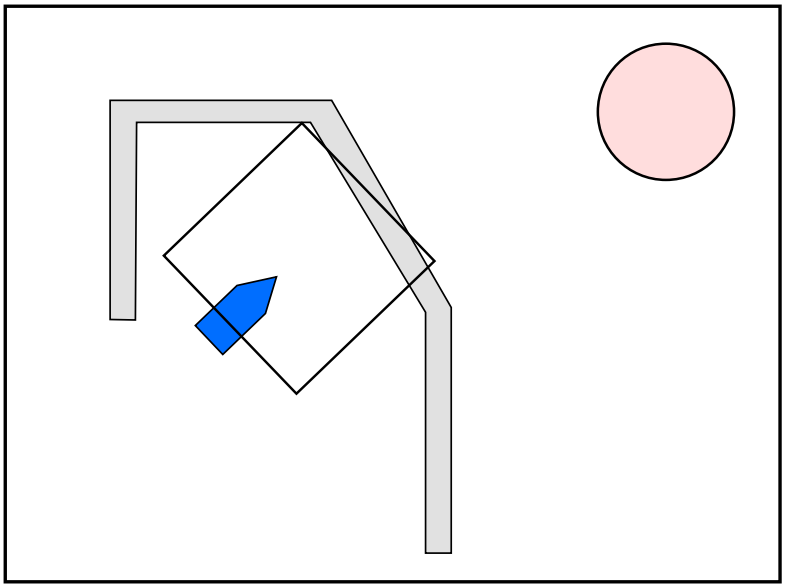


Figure 2.1: The difference between local and global methods. The inner square represents the field of view for the local controller, the outer rectangle is the configuration space and the red circle is the goal. In a situation like this a local method would get stuck.

2.2.1 Potential Fields

One of the easiest and most intuitive methods for motion planning and collision avoidance is the *Potential Field* method. It was first presented in Khatib (1986) and introduced a fast and efficient way of exploring $\mathcal{Q}_{\text{free}}$.

Let the vehicle be represented as a point particle in the configuration space under the influence of an artificial potential field $\mathbf{U}$. The potential field consists of an attractive component (the goal configuration) and repulsive components (obstacles). Ideally, this field should be constructed such that the goal configuration, $\mathbf{q}_g$, is the only (global) minima, however this is typically difficult or impossible to achieve.

The field is constructed as an additive field on the form

$$\mathbf{U}(\mathbf{q}) = \mathbf{U}_{\text{att}}(\mathbf{q}) + \mathbf{U}_{\text{rep}}(\mathbf{q}) \tag{2.12}$$

To find the minima and solve the problem, a simple gradient descent algorithm is used (Eq. (2.13)). The gradient may be seen as a generalized force acting upon the vessel, steering it towards the goal whilst being repulsed by obstacles.

$$\boldsymbol{\tau}(\mathbf{q}) = -\nabla \mathbf{U}(\mathbf{q}) \tag{2.13}$$

Virtual Force Fields

The potential field method implemented in this project is a variant of the Virtual Force Field (VFF) method (Borenstein & Koren, 1989), a real-time obstacle avoidance approach for fast mobile robots. Its main advantage is the use of uncertainty grids for representing obstacles, thus accommodating for uncertain sensor data and as well as sensor fusion.

Let $\mathbf{p}_o^b$ be the vector from the vehicle to an obstacle, then $\rho = \|\mathbf{p}_o^b\|_2$ is the distance between the vessel and the obstacle. Furthermore, let ρ_0 be the distance of influence of the obstacle. Then, the potential field generated by an obstacle may be defined as

$$\mathbf{U}_{\text{obs},i}(\mathbf{p}_o^b) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & \text{if} \quad \rho \leq \rho_0 \\ 0 & \text{else} \end{cases} \tag{2.14}$$

The force generated by this potential field is

$$\mathbf{F}_{\text{obs},i}(\mathbf{p}_o^b) = \begin{cases} \eta\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)\frac{\mathbf{p}_o^b}{\rho^3} & \text{if} \quad \rho \leq \rho_0 \\ 0 & \text{else} \end{cases} \tag{2.15}$$

Each point along the edge of a polygon is regarded as an individual obstacle with probability 1 in the uncertainty grid, as the sensors are assumed to be perfect in this project.

The attractive field may be expressed as

$$\mathbf{U}_{\text{att}}(\mathbf{x}) = \frac{1}{2}\zeta\|\mathbf{x} - \mathbf{x}_d\|^2 = \frac{1}{2}\zeta\|\mathbf{e}\|^2\,, \quad \mathbf{e} \triangleq \mathbf{x} - \mathbf{x}_d \tag{2.16}$$

This field is sometimes referred to as a *parabolic well potential* (Spong et al., 2006). The force generated by this field is

$$\mathbf{F}_{\text{att}}(\mathbf{x}) = -\zeta\mathbf{e}, \tag{2.17}$$

which converges linearly to zero as $\mathbf{x}$ approaches $\mathbf{x}_d$, which is desirable, however it also grows without bounds when the vessel moves away from $\mathbf{x}_d$. The attractive force works as a common P-controller. To improve stability a dissipative force, proportional to the velocity of the vessel is added (Loe, 2007). In addition, an upper bound is given to the force.

$$\mathbf{F}_{\text{att}}(\mathbf{x}) = -d\dot{\mathbf{x}} - \begin{cases} \zeta\mathbf{e} & \text{if} \quad \zeta\|\mathbf{e}\| \leq V_{\max} \\ V_{\max}\frac{\mathbf{e}}{\|\mathbf{e}\|} & \text{else} \end{cases} \tag{2.18}$$

The sum of forces, $\mathbf{F} = \mathbf{F}_{\text{att}} + \sum_i \mathbf{F}_{\text{obs},i}$, will be the control input to the vessel. Due to the underactuated nature of the vehicle, the sum of forces needs to be converted into a heading and speed reference. The heading reference is simply the direction of the force vector

$$\psi_d = \text{atan2}(F_y,\, F_x) \tag{2.19}$$

The speed reference is chosen as

$$u_d = (\mathbf{F}^\top \mathbf{d})\mathbf{d}, \quad \mathbf{d} = \begin{bmatrix} \cos\psi & \sin\psi \end{bmatrix}^\top, \tag{2.20}$$

thus slowing the vessel down when the force vector is directed away from the direction of travel.

2.2.2 The Dynamic Window Approach

The Dynamic Window Approach (DWA) was first introduced in Fox et al. (1997) as a collision avoidance method for synchro-drive wheeled mobile robots (WMRs). It differs from previous approaches by incorporating the dynamics of the vehicle and searching directly in the velocity space of the vehicle for control commands, thus ensuring feasibility. The derivations found in this section is also found in greater detail in Fox et al. (1997), but here the SNAME notation is used.

Circular Trajectories

The general equations of motion for a wheeled robot is given as

$$\begin{aligned} x(t_n) &= x(t_0) + \int_{t_0}^{t_n} u(t) \cdot \cos\psi(t)\, \mathrm{d}t \\ y(t_n) &= y(t_0) + \int_{t_0}^{t_n} u(t) \cdot \sin\psi(t)\, \mathrm{d}t \end{aligned} \tag{2.21}$$

Assuming constant velocities in the time interval $t \in [t_i, t_{i+1}]$, it can be shown that the equations of motion may be approximated as

$$\begin{aligned} x(t_n) &= x(t_0) + \sum_{i=0}^{n-1} F_x^i(t_{i+1}) \\ y(t_n) &= y(t_0) + \sum_{i=0}^{n-1} F_y^i(t_{i+1}) \end{aligned} \tag{2.22}$$

where

$$\begin{aligned} F_x^i(t) &= \begin{cases} \frac{u_i}{r_i}(\sin\psi(t_i) - \sin(\psi(t_i) + q\cdot(t-t_i))), & r_i \neq 0 \\ u_i\cos(\psi(t_i))\cdot t, & r_i = 0 \end{cases} \\ F_y^i(t) &= \begin{cases} -\frac{u_i}{r_i}(\cos\psi(t_i) - \cos(\psi(t_i) + q\cdot(t-t_i))), & r_i \neq 0 \\ u_i\sin(\psi(t_i))\cdot t, & r_i = 0 \end{cases} \end{aligned} \tag{2.23}$$

If $r_i = 0$, the vehicle will follow a straight line. Conversely, if $r_i \neq 0$, the vehicle trajectory describes a circle with center

$$\begin{aligned} M_x^i &= -\frac{u_i}{r_i}\sin\psi(t_i) \\ M_y^i &= \frac{u_i}{r_i}\cos\psi(t_i) \end{aligned} \tag{2.24}$$

and radius

$$M_r^i = \frac{u_i}{r_i} \tag{2.25}$$

Admissible Velocities

Thus, the pair (u_i, r_i) represents the velocities at interval i. The goal of the algorithm is to choose the best pair of velocities. This is done by first restricting the choice of velocities to the set of *admissible velocities*.

To ensure safe operation, velocities is selected so that the vehicle is able to come to a complete stop in the next interval. This is the set of admissible velocities and is defined as

$$V_a \triangleq \left\{ u, r \mid u \leq \sqrt{2\cdot \mathrm{dist}(u,r)\cdot \dot{u}_b}, \quad r \leq \sqrt{2\cdot \mathrm{dist}(u,r)\cdot \dot{r}_b} \right\}, \tag{2.26}$$

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

The Dynamic Window

Secondly, the choice of velocities is restricted to reside in the *dynamic window*. The dynamic window consists of the velocities reachable within the next time interval, Δt. It is centered around the current velocities (u_a, r_a) and is defined as

$$V_d \triangleq \{u, r \mid u \in [u_a - \dot{u}\Delta t,\, u_a + \dot{u}\Delta t], \quad r \in [r_a - \dot{r}\Delta t,\, r_a + \dot{r}\Delta t]\} \tag{2.27}$$

Furthermore, let V_s denote the set of velocities attainable by our vehicle, then the set of valid velocities for the next time interval is

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

This set is known as the *search space*.

Selecting the Optimal Velocities

The optimal velocities is found by maximizing the objective function

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

where $\sigma(\cdot)$ is function that smooths the weighted sum, resulting in more side-clearance from obstacles. An example of the selection of optimal velocities is shown in Figure 2.2.

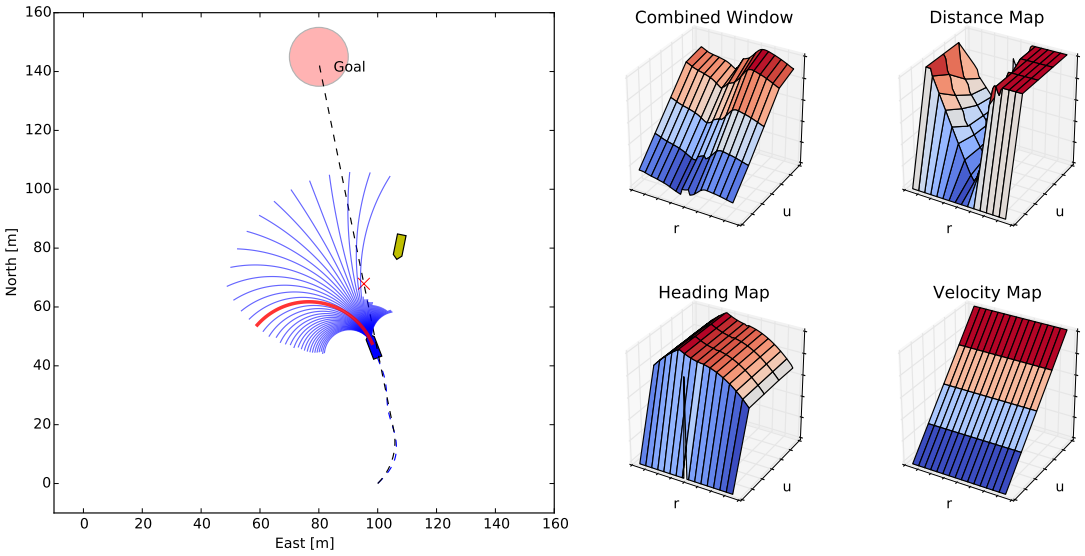


Figure 2.2: Finding the optimal velocities for the dynamic window. In this simulation hybrid-state A* is used to guide the DWA. The red cross is the current goal and the red line represents the optimal arc. Note that the blue vessel predicts the motion of the attacker by assuming it keeps a constant course and speed.

2.3 Global Methods

Global methods, in contrast to local methods, considers the full configuration space. They use a priori knowledge, *i.e.*, a map, to find a set of states describing in

a complete motion from the start to the goal. Several global methods are *complete*, meaning that if such a set of states exist, the algorithm will find it. In the literature they are sometimes known as *deliberate*. Considering a full set of motions naturally causes the global methods to be slower than their local counterparts, albeit some global methods may run in a fraction of a second at the cost of reducing the dynamical constraints or search heuristics. However, they are in general regarded as slow.

2.3.1 A* search

A star search, or simply *A**, first introduced in (Hart et al., 1968) and today one of the most popular path finding algorithms. It is a *best-first* method that applies heuristics to guide the algorithm.

The A* search is a very popular method for path planning, and with good reason. It is a computational effective and highly flexible algorithm and is used in a wide range of applications, from computer games to path planning for mobile robots. The flexibility lies in its use of heuristics and indifference towards how nodes are explored. The A* methodology has inspired many algorithms, such as D* (A. Stentz, 1994), Field D* (Ferguson & Anthony Stentz, 2005), Theta* (Daniel, Nash, Koenig, & Felner, 2010) and Hybrid-state A* Montemerlo et al. (2008), Dolgov et al. (2010).

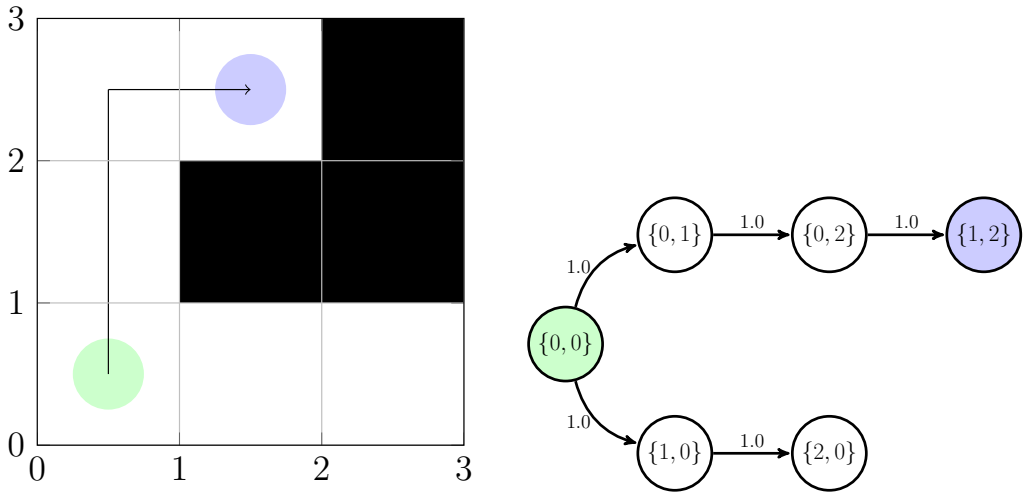
The algorithm searches through a set interconnected nodes, *i.e.*, a *graph*. The graph may be represented in many ways, but the simplest is to use a binary grid (Fig. 2.3).

A* is also a *complete* algorithm, *i.e.*, if there exist a path to the goal, the algorithm will find it. The pseudocode for A* is given in Algorithm 2.1.

Heuristics

One of the major advantages of A* is its use of heuristics. Heuristics, *or a heuristic function*, is a way of ranking alternatives in search algorithms. It gives an estimate the cost of reaching the goal from the given node. Specifically in the A* case, the heuristic function is used to determine which neighbor to explore.

A heuristic function $h(n)$ is said to be admissible if it does not overestimate the cost of reaching the goal. Furthermore $h(n)$ is said to be monotonic (or consistent)



(a) Grid representation of map (b) Corresponding graph

Figure 2.3: Grid representation of graph (or vice versa)

if for two adjacent nodes x, y with the length $d(x, y)$ between them we have

$$h(x) \leq d(x, y) + h(y) \tag{2.30}$$

It is possible to relax the admissibility criterion to speed up the search, at the expense of suboptimal paths.

2.3.2 Hybrid-state A* search

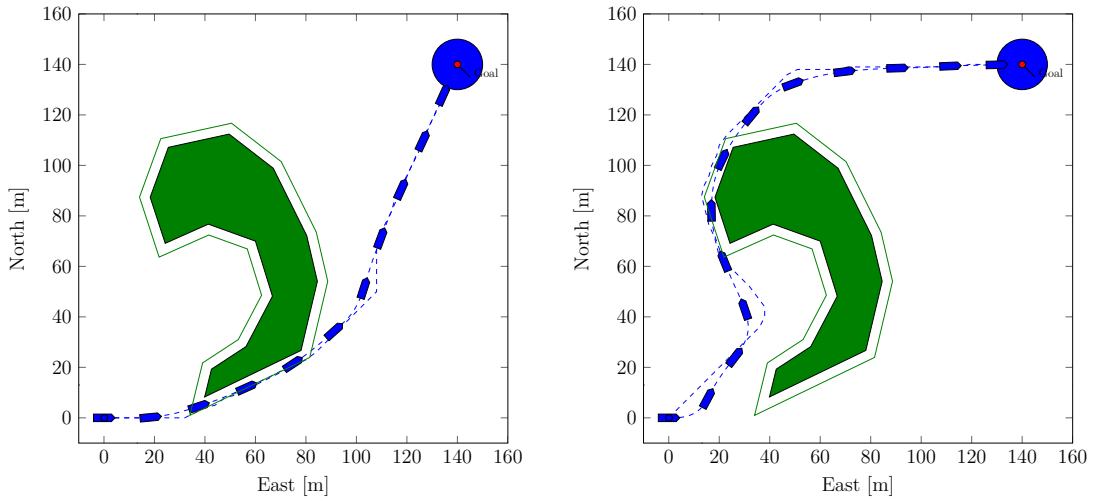
The *Hybrid-state A* search* algorithm was first presented in Montemerlo et al. (2008) and later in more detail in Dolgov et al. (2010). It builds upon the well-known A* algorithm.

Where the traditional A* normally explore in 4 or 8 directions, the hybrid-state A* search applies several steering actions and simulate the vessels motion. Furthermore, the hybrid-state A* associates each discretized cell with a continuous 3D state, $\mathbf{s}_i = (x, y, \psi)_i^\top$. Note that hybrid-state A* also searches in ψ, resulting in a three-dimensional search space.

By the restriction employed on the exploration of neighbors, finding the minimal-cost solution is abandoned, as is the algorithm completeness, but kinematic feasibility is guaranteed. However, the paths found often lie in the neighborhood of the optimal solution and may be further improved by smoothing techniques (Dolgov et al., 2010).

Algorithm 2.1 A* pseudo code

$\mathcal{O} \leftarrow start$ ▷ The open set, sorted by lowest estimated total cost
$cost(start) \leftarrow 0$
$est\_cost(start) \leftarrow cost(start) + heuristic(start, goal)$
$\mathcal{C} \leftarrow \emptyset$ ▷ The closed set
while $\mathcal{O} \neq \emptyset$ **do**
  $current \leftarrow \min(\mathcal{O})$ ▷ Extract the node with least estimated cost
  if *current* is *goal* **then**
    return get_path(*current*)
  end if
  for all neighbors, n, of *current* **do**
    $new\_cost \leftarrow cost(current) + dist(n)$
    if $n \notin \mathcal{C} \vee new\_cost < cost(n)$ **then**
      $cost(n) \leftarrow new\_cost$
      $est\_cost(n) \leftarrow cost(n) + heuristic(n, goal)$
      $\mathcal{O} \leftarrow n$
    end if
  end for
end while



(a) Admissible heuristic (b) Relaxed criterion

Figure 2.4: By relaxing the admissibility criterion one might speed up the search at the expense of sub-optimal paths.

Exploring the Neighbors

The neighboring nodes are found by applying several steering actions to the continuous state associated with the current node and generate child states by using

the kinematic model of the vehicle. For each of the continuous children states, the grid cell that they fall into is computed. If a node with the same grid cell is present on the A* open list and the cost of the new node is lower than the existing cost, the continuous state of the node is updated and it is re-prioritized on the open list.

Instead of simulating the system every time, pre-calculated vectors are rotated and translated according to the pose of the current state. Let $\boldsymbol{\eta}_k$ be the vehicle pose at node k, and the pre-calculated vector be defined as

$$\hat{\boldsymbol{\eta}}_i = \begin{bmatrix} l_s \cos \delta\theta_i \\ l_s \sin \delta\theta_i \\ \delta\theta_i \end{bmatrix} \tag{2.31}$$

The prediction distance l_s is proportional to the grid size and $\delta\theta_i \in [\delta\theta_{min},\ \delta\theta_{max}]$ is the resulting change in heading when applying steering commands $\theta_i \in [\theta_{min}, \theta_{max}]$. In this implementation, as in Dolgov et al. (2010), three cases are considered: hard port, no action and hard starboard, *i.e.*,

$$\theta_i \in \{\theta_{min},\ 0,\ \theta_{max}\} \tag{2.32}$$

The resulting vectors are then rotated and translated according to Eq. (2.33) for each neighbor explored.

$$\boldsymbol{\eta}_{n,i} = \boldsymbol{\eta}_k + \mathbf{R}_{z,\psi}\hat{\boldsymbol{\eta}}_i \tag{2.33}$$

Heuristics

The Hybrid-state A* algorithm may greatly benefit from guiding heuristics. Dolgov et al. (2010) describes two methods: *non-holonomic without obstacles* and *holonomic with obstacles*.

The first heuristic takes the non-holonomic nature of the vehicle into account, but neglects the obstacles. The shortest path from every point in some discretized neighborhood of the goal is computed and simply translated and rotated to match the current goal. This way, paths that approaches the goal from the wrong direction is assigned a high cost and thus not prioritized in the search. The heuristic may be fully precomputed offline. As one would expect, this heuristic yields best results in sparse environments and therefore is expected to give a significant performance boost in a marine application.

The second heuristic, holonomic-with-obstacles, ignores the holonomic nature of the vehicle, but takes obstacles into account. This guides the more expensive search away from dead-ends (*e.g.* U-shaped obstacles).

Analytic Expansions

In order to improve search speed and accuracy of the solution, the use of analytic expansions based on the Reeds-Shepp (Reeds & Shepp, 1990) model is proposed by Dolgov et al. (2010). For ships, changing direction of motion (forward/reverse) is a non-trivial task, thus Dubins paths (Dubins, 1957) is proposed instead.

The Reeds-Shepp model is equivalent to the Dubins model, except it allows reverse motion as well. The Dubins model, "Dubins car", is given in Eq. (2.34), using the notation from LaValle, 2006.

$$\begin{aligned} \dot{x} &= \cos\theta u_s \\ \dot{y} &= \sin\theta u_s \\ \dot{\theta} &= u_\phi \end{aligned} \tag{2.34}$$

where $u_\phi \in U = [-\tan\phi_{max}, \tan\phi_{max}]$ is the steering control action, u_s is the speed control action (assumed constant) and ϕ is the wheel angle. Dubins showed that the shortest path between two configurations $q_i = \{x, y, \theta\}_i$, is a sequence consisting of left (L), right (R) or straight ahead (S) motion. Furthermore, he showed that only six possible sequences are possible:

$$\{LRL,\ RLR,\ LSL,\ LSR,\ RSL,\ RSR\}. \tag{2.35}$$

The expansion works as follows: for every N node explored by the algorithm, generate a Dubins curve from the current node to the goal and check for intersections with the world. N decreases as a function of the distance to the goal.

2.4 Hybrid Methods

In several practical applications, a dual approach is proposed (Loe, 2008; Marder-Eppstein et al., 2010). The idea is to utilize the strong sides of both method types, *i.e.*, the completeness of the global methods and the fast reaction of the local methods.

2.4.1 Combining Local and Global Methods

A global method such as A* outputs a path, but local methods expect either a single goal configuration or a heading reference. Thus, a path tracking scheme is

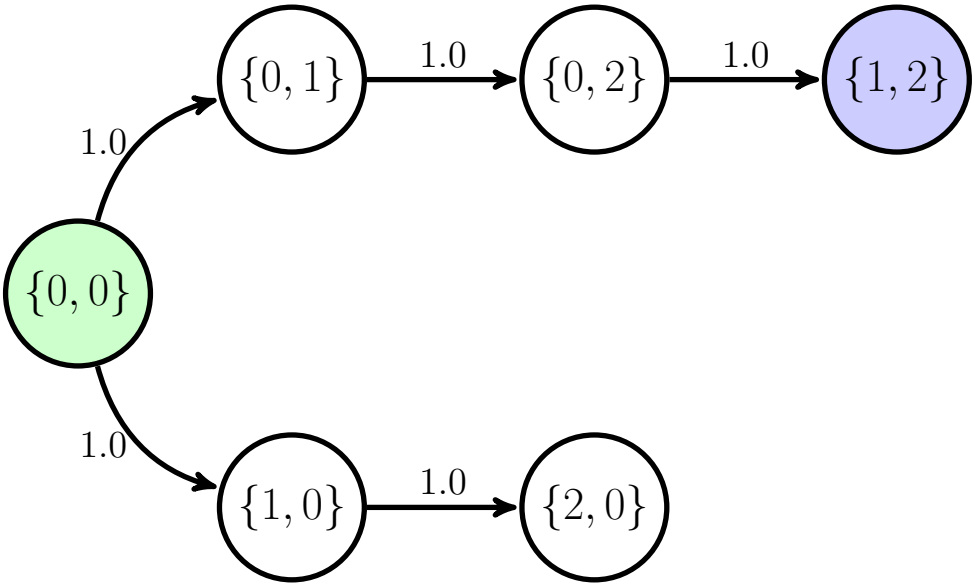


Figure 2.5: Example of a Dubins curve. The start and goal position is marked with a small circle. This is a LSR curve.

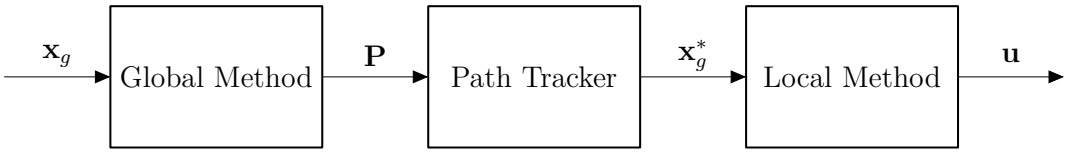


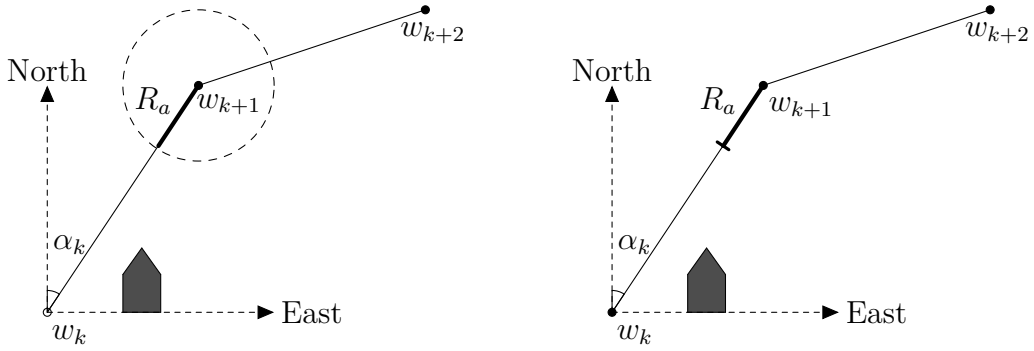
Figure 2.6: Control flow of hybrid methods. A path tracker is needed for interaction between the global and local method. Given a feasible goal configuration $\mathbf{x}_g$, the global method outputs a path $\mathbf{P}$ to reach the goal. A path tracker keeps track of how far along the path the vehicle has come and provides the local method with a temporary goal $\mathbf{x}_g^*$ or heading reference ψ_d. Finally, the local method steers the vehicle towards the current goal whilst avoiding any unforeseen obstacles.

needed for fusing the methods together (Fig. 2.6). One way of doing this is with a simple pure pursuit scheme (Fossen, 2011, p. 244) or the more advanced Line of Sight (LOS) method (Breivik & Fossen, 2004).

Another problem to be tackled is the waypoint switching criterion. Normally, with a pure pursuit or LOS scheme, waypoint switching is done with a *circle of acceptance* (Fig. 2.7a) switching criterion (Eq. (2.36)). Let $\mathbf{w}_{k+1} = (x_{k+1}, y_{k+1})$, then the switching criterion will be

$$(x_{k+1} - x)^2 + (y_{k+1} - y)^2 \leq R_a^2. \tag{2.36}$$

An advantage of this method is that the switching occurs before the waypoint is actually reached and the vehicle may therefore begin turning towards the next waypoint early – avoiding overshooting.



(a) Circle of acceptance method

(b) Progress along path method

Figure 2.7: 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.

This works very well in the case when every waypoint is to be visited, however with the hybrid approach the path is merely there as a guide for the local algorithm. Loe (2008, p. 62, Eq. (3.40)) therefore suggest a switching criterion based on progress along the path (Fig. 2.7b). Note that the proposed criterion is (assuming an error in the indexing as well)

$$(x(t) - x_{k+1}) \cos \alpha_k - (y(t) - y_{k+1}) \sin \alpha_k \leq Ra, \tag{2.37}$$

whilst the correct criterion is

$$(x_{k+1} - x(t)) \cos \alpha_k + (y_{k+1} - y(t)) \sin \alpha_k \leq Ra. \tag{2.38}$$

See Figure 2.8 for an illustration of the difference.

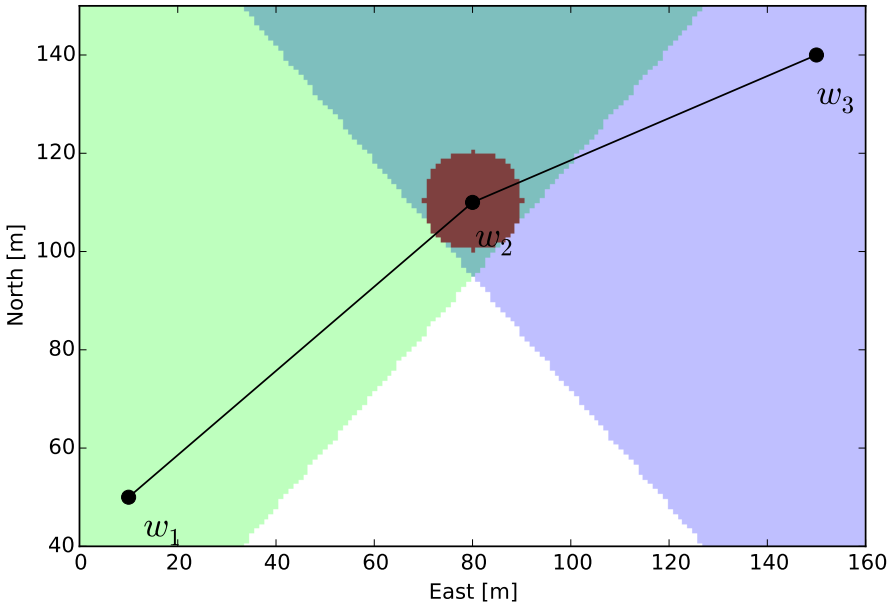


Figure 2.8: Illustration of the acceptance regions. With $\mathbf{w}_{k+1} = w_2$, the proposed region in Eq. 2.37 is in green, the corrected version from Eq. 2.38 is in blue and a circle of acceptance region is shown in brown.

Chapter 3

Simulation Results

3.1 Prerequisites

An important discussion yet to be had is *how to compare the different methods?* In this report, some parameters has been chosen for quantifying the performance of the algorithms presented, these are given in Table 3.1. However, quantifying the performance of planning algorithms is difficult as it may be highly dependent on factors such as implementation specifics or tuning. Therefore, the goal of the simulations presented in this chapter is to highlight the advantages and disadvantages with the individual methods. One parameter that is purposely left out is the CPU time used by the various algorithms, as this parameter is *very* prone to implementation specifics.

Table 3.1: Parameters used to compare algorithm performance

Parameter	Description
Success	Whether or not the vehicle reached its destination
Quality	The (subjective) quality of the chosen path
Length	Length of path traveled
Simulation time	Time taken to reach the goal

A summary of the platform used to run the simulations is given in Table 3.2.

Table 3.2: Summary of platform used for simulation

Computer	Dell Optiplex 990
Processor	Intel Core i5-2500 @ 3.30GHz × 4
Memory	Samsung DDR3 1333 MHz 4GB × 2
Operating System	Ubuntu 12.04 LTS
Program Versions	Python 2.7.3, Matplotlib 1.4.2, NumPy 1.6.1, SciPy 0.14.0

3.2 Results

3.2.1 Scenario 1 – Easy Mode

As the title may suggest, this scenario is regarded as easy and all algorithms should be able to complete it. The environment is purely static and contains no local minima. All of the algorithms are expected to navigate safely and relatively effective through this scenario. The simulation results are shown in Figures 3.1, 3.2 and in Table 3.3.

Table 3.3: Results from Scenario 1. The A* method performs best in terms of path length and simulation time, but due to oscillations in the start and extensive contour hugging, DWA and Hybrid-state A* gives better path quality.

Algorithm	Success	Quality	Distance (m)	Sim. Time (s)
Virtual Force Field	Yes	6/10	274,08	134,60
Dynamic Window	Yes	8/10	238,45	81,20
A*	Yes	7/10	223,80	76,35
Hybrid-state A*	Yes	8/10	214,05	73,15
A* + VFF	Yes	4/10	327,85	146,10
A* + DWA	Yes	8/10	233,68	79,60
HS A* + VFF	Yes	5/10	271,58	126,95
HS A* + DWA	Yes	8/10	229,76	78,30

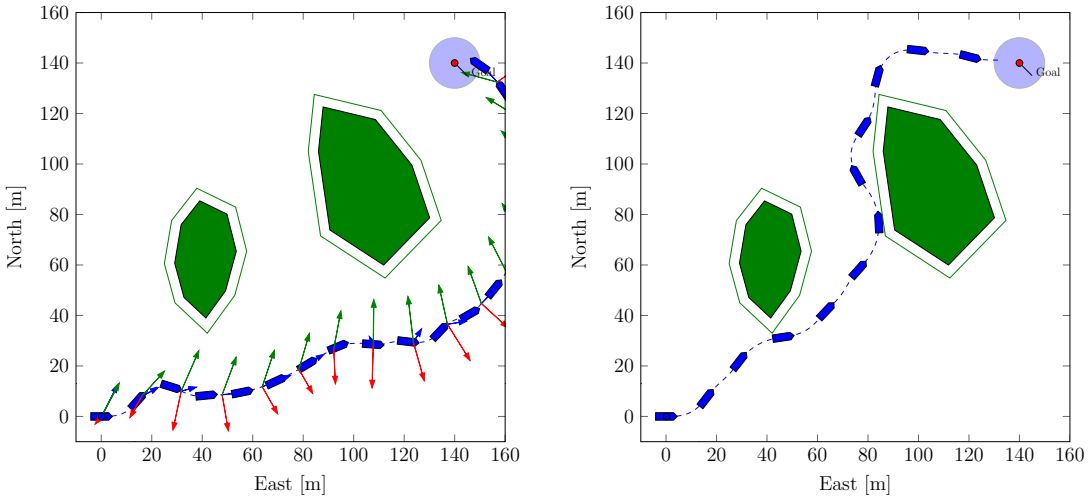
From Figure 3.1a an oscillatory response is clearly visible, nevertheless the virtual force field method steers the vehicle towards the goal without too much difficulty.

The dynamic window algorithm performs very well in this scenario (Fig. 3.1b). At one point it gets very close the second island, nevertheless the path is in general very smooth.

The A* search does not take the kinematics of the vehicle into account and as a result of this the vehicle struggles to follow the path at the start (Fig. 3.1c). In addition, the contour hugging (Sec. 4.1.3, p. 35) phenomena is clearly visible.

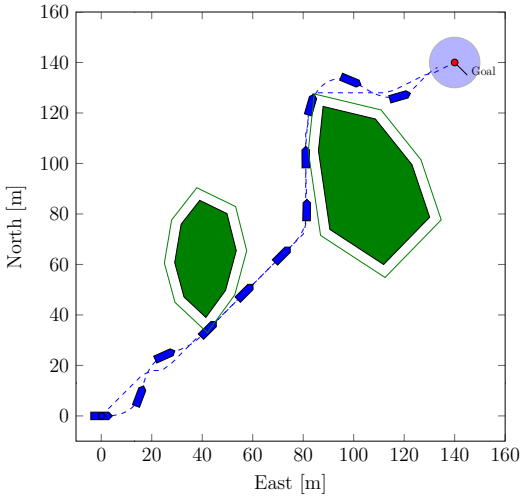
The hybrid-state A* search performs very well, although it passes the first island with little clearance (Fig. 3.1d).

When paired with dynamic window, both global methods performs well (Figs. 3.2b and 3.2d), however the little side clearance provided by the global paths cause problems for the virtual force field method. When the global methods force the virtual force field controller in between the islands severe oscillatory motion ensues (Figs. 3.2a and 3.2c).

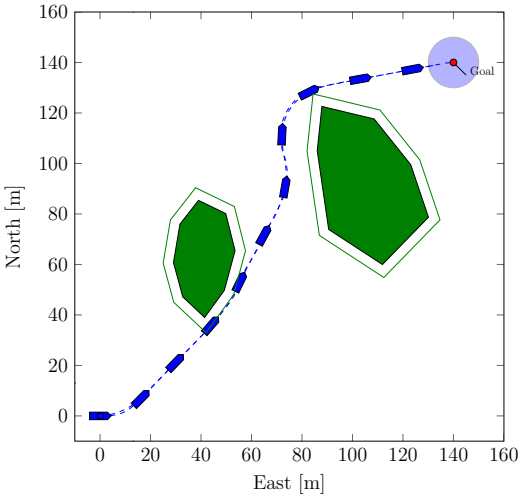


(a) Virtual Force Fields

(b) Dynamic Window

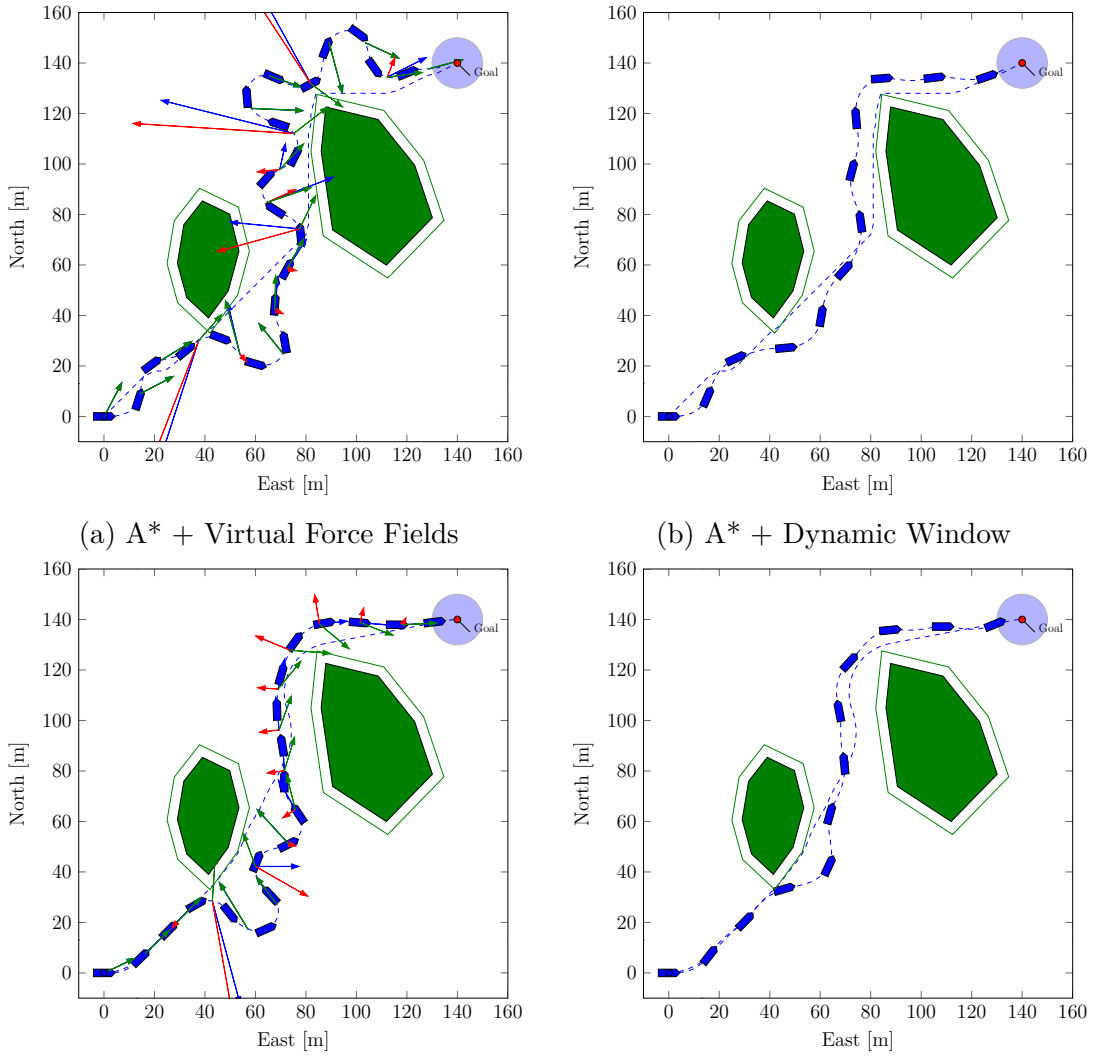


(c) A*



(d) Hybrid-state A*

Figure 3.1: Simulation results for Scenario 1. Overall good performance by all controllers, but the hybrid-A* provides the smoothest path.



(a) A* + Virtual Force Fields

(b) A* + Dynamic Window

(c) Hybrid-state A* + Virtual Force Fields

(d) Hybrid-state A* + Dynamic Window

Figure 3.2: Simulation results for Scenario 1 – hybrid methods. The vessels with dynamic window as local clearly outperforms the virtual force field.

3.2.2 Scenario 2 – Local Minima

This scenario was designed to illustrate the main weakness with local planners – they tend to get stuck in local minima. An U-shaped island represents the local minima. The simulation results are summarized in Figures 3.3, 3.4 and in Table 3.4.

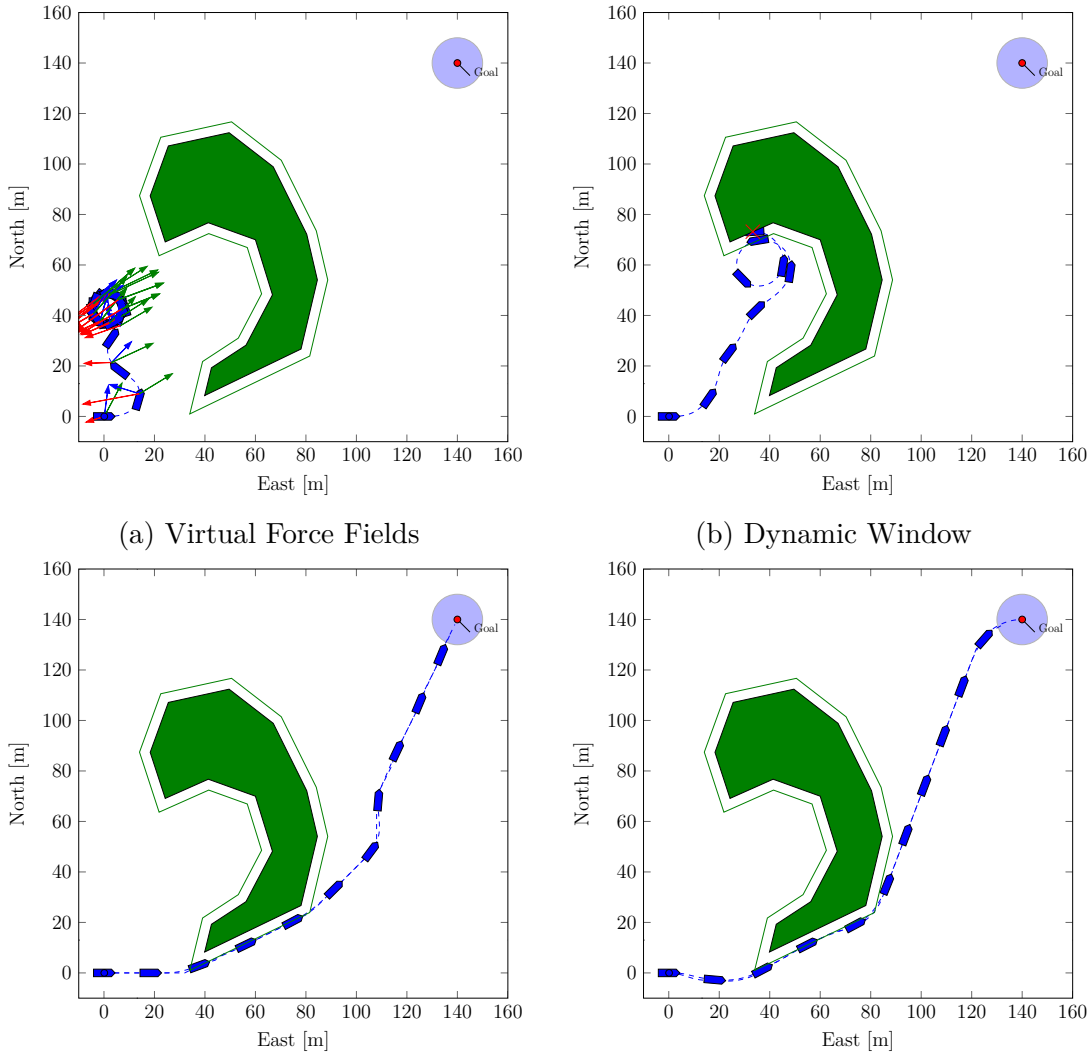
Table 3.4: Results from Scenario 2. Both local methods fail to escape the minima and therefore fails the task. The global methods succeed, albeit exhibiting extensive contour hugging in this scenario as well.

Algorithm	Success	Quality	Distance (m)	Sim. Time (s)
Virtual Force Field	No	0/10	306,15	150,00
Dynamic Window	No	0/10	161,27	56,05
A*	Yes	9/10	212,99	72,90
Hybrid-state A*	Yes	9/10	214,70	73,40
A* + VFF	Yes	6/10	265,89	119,45
A* + DWA	Yes	8/10	226,44	77,20
HS A* + VFF	Yes	6/10	283,69	127,85
HS A* + DWA	Yes	8/10	228,25	77,80

As expected, both the local methods (Figs. 3.3a and 3.3b) performs poorly and get stuck.

The global methods (Figs. 3.3c and 3.3d) easily circumvents the minima and successfully reaches the goal, although with little side clearance.

All hybrid combinations manages to find the goal, but yet again the virtual force field method has some severe oscillations when traveling close to the shore (Fig. 3.4).



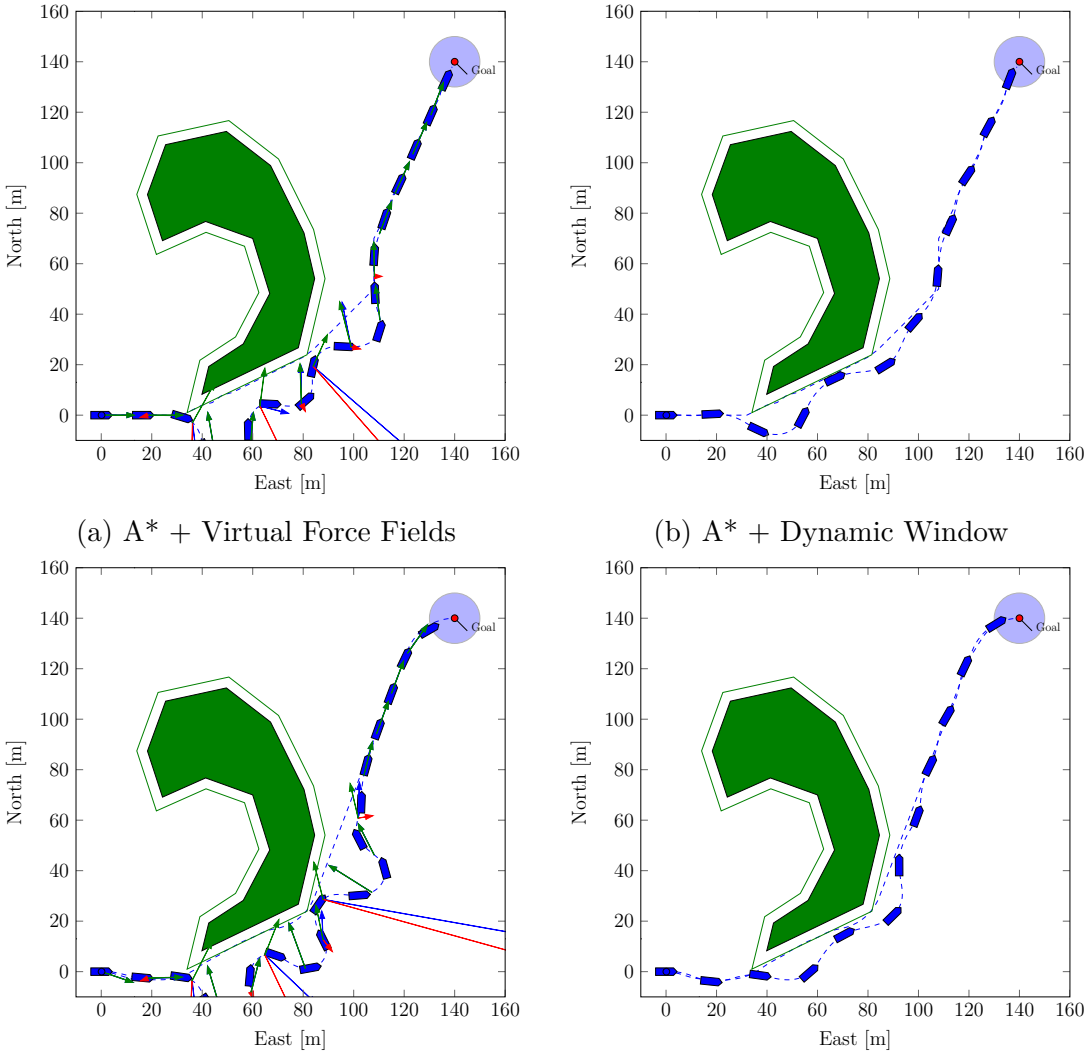
(a) Virtual Force Fields

(b) Dynamic Window

(c) A*

(d) Hybrid-state A*

Figure 3.3: Results for scenario 2. Both local methods fails to complete its task, but the global methods easily circumvents the local minima.



(a) A* + Virtual Force Fields

(b) A* + Dynamic Window

(c) Hybrid-state A* + Virtual Force Fields

(d) Hybrid-state A* + Dynamic Window

Figure 3.4: Simulation results for Scenario 2 – hybrid methods. Both sets of methods behave similar, with virtual force fields clearly worst performing. This is mostly caused by the A* based methods tendency of contour-hugging.

3.2.3 Scenario 3 – Under Attack

Global planners often outperforms the local planners in static environments, but in the presence of dynamic obstacles, *i.e.*, other vehicles, the lack of re-planning leads to problems. In this scenario, there are no static obstacles, but instead another ship, equipped with a *pure pursuit* controller (Fossen, 2011, p. 244), seeks to collide with our own, but its maximum speed is limited to 2,5 m/s. A summary of the results are found in Figures 3.5, 3.6 and in Table 3.5.

Table 3.5: Results from Scenario 3. Both local methods successfully completes the task, but the VFF method use more than three times the time to reach the goal. The lack of re-planning causes both global methods to fail.

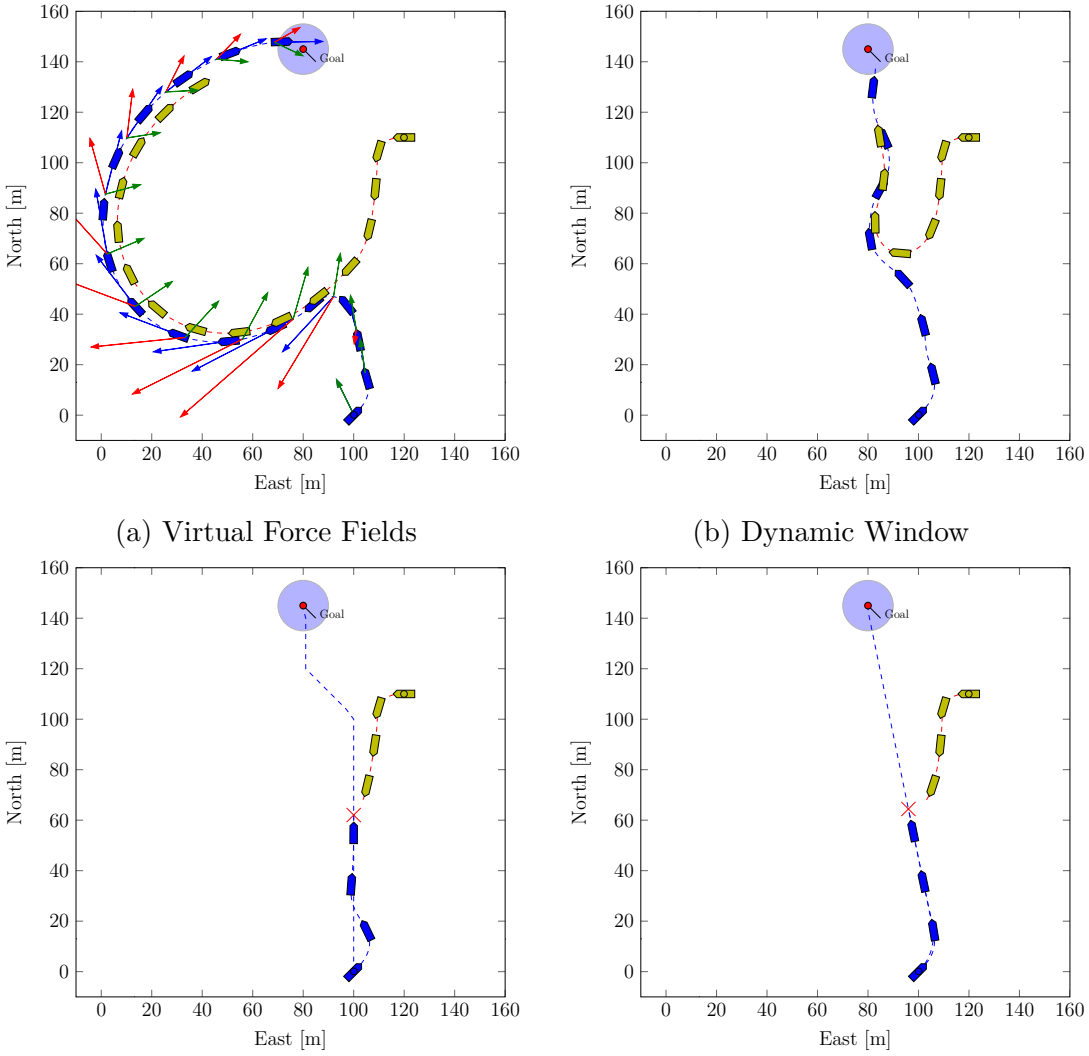
Algorithm	Success	Quality	Distance (m)	Sim. Time (s)
Virtual Force Field	Yes	6/10	288,63	105,70
Dynamic Window	Yes	9/10	150,52	51,90
A*	No	0/10	65,45	23,40
Hybrid-state A*	No	0/10	67,20	24,00
A* + VFF	No	0/10	58,55	26,45
A* + DWA	Yes	7/10	186,90	64,00
HS A* + VFF	Yes	5/10	257,37	96,70
HS A* + DWA	Yes	8/10	156,24	53,80

The virtual force field methods successfully avoids the other vehicle by "outrunning" it (Fig. 3.5a). However, if the attacking vehicle where to start at a more direct angle, our vessel would have difficulties escaping.

The dynamic window algorithm also successfully avoids the attacking vehicle, even more effective than the virtual force field method (Fig. 3.5b).

Both global methods fails to complete the task as they have no a priori information about the attacking vehicle does not re-plan their paths (Figs. 3.5c and 3.5d).

Again the global methods paired with the dynamic window local controller performs best, with a slight better performance by hybrid-state A* (Figs. 3.2b and 3.2d). The virtual force field controller fails when paired with A* (Fig. 3.2a) and barely succeeds when paired with hybrid-state A* (Fig. 3.2c), although this may very well be coincidental.



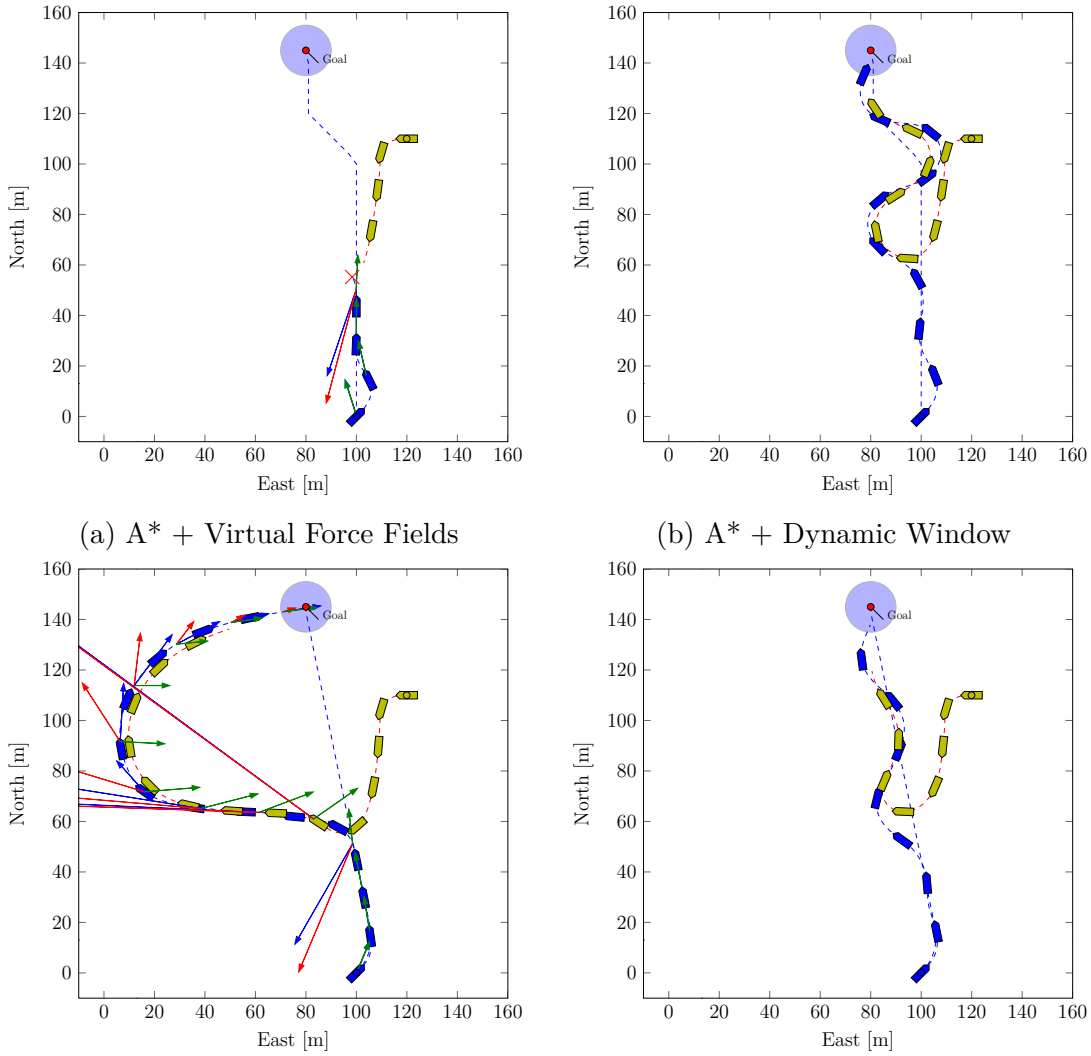
(a) Virtual Force Fields

(b) Dynamic Window

(c) A*

(d) Hybrid-state A*

Figure 3.5: Results for Scenario 3. Both local methods succeed, though the VFF method does so less elegant than the DWA. With the lack of re-planning, both global methods fail.



(a) A* + Virtual Force Fields

(b) A* + Dynamic Window

(c) Hybrid-state A* + Virtual Force Fields

(d) Hybrid-state A* + Dynamic Window

Figure 3.6: Simulation results for Scenario 3 – hybrid methods. Of the hybrid methods the best performing combination is hybrid-state A* and dynamic window.

Chapter 4

Discussion

The simulation results from Chapter 3 highlights the main qualities of the different approaches. What should be clear is that no single local nor global method completed all the scenarios, yet when the dynamic window controller was paired with a global method, it succeeded in all scenarios. In this chapter, the algorithm performance in light of the simulation results and the findings in other literature will be discussed.

4.1 Individual Algorithm Performance

4.1.1 Virtual Force Field

The overall performance of the virtual force field algorithm is poor. The advantage of potential field methods lies in their simplicity and elegance. Experiments with WMRs has shown sufficient results and although the motion of surface vessels is similar, their general lack of maneuverability renders this method unsuitable. Koren and Borenstein (1991) identifies four significant problems that are inherent to potential field methods, independent of implementation:

1. Trap situations due to local minima
2. No passage between closely spaced obstacles
3. Oscillations in the presence of obstacles
4. Oscillations in narrow passages

Phenomena one and three are clearly visible in Figures 3.3a and 3.1a, p. 27 and 24, respectively.

Optimistic heading set point may be the largest contributing factor to the observed oscillatory response of the algorithm (Fig. 3.1a, p. 24). The underactuated nature of the vehicle and its lack of agility (due to high inertia and induced sway motion when turning) in comparison to a WMR worsen this behavior significantly, almost acting as a time delay in a control system.

The method is originally designed for continuous control for (fast) WMRs with no a priori knowledge of the environment, therefore it misses out on a major advantage with navigation at sea: maps. To improve computational performance the static obstacle forces may be precomputed.

4.1.2 Dynamic Window

Of the local algorithms, the dynamic window approach performs best. The algorithm is today a very popular method for collision avoidance and used in, *e.g.*, the Robot Operating System (Open Source Robotics Foundation, 2014). By searching directly in the velocity space of the vehicle, the algorithm effectively considers the vehicle dynamics. However, like all local methods, it fails near local minima (Fig. 3.3b, p. 27).

There are several ways of modifying the algorithm to improve its performance for marine applications. In the original implementation (Fox et al., 1997) the velocity search space was derived from the motion equations for a synchro-drive wheeled mobile robot (WMR). The motion of a WMR is similar to that of a surface vehicle, but, *e.g.*, it does not have the same challenges with lateral (sway) motion. In addition, the algorithm requires an instant jump in velocities (u, r) to the arc velocities (u_i, r_i), which is infeasible. The larger the jump in velocities, the more inaccurate the predictions will be and more sway motion will be induced. In Loe (2008) vehicle accelerations and lateral speed estimation was included in the dynamic window prediction to mitigate these problems. The added features slowed the algorithm by 20%, but provided much better estimates of the vehicle trajectories. Another approach may be to use Fermat's spirals (Lekkas, 2014; Dahl, 2013) or clothoids in the transition to circular arcs to remove the discontinuity in curvature of the arcs (Wilde, 2009). This may reduce the estimation errors associated with large changes in yaw rate.

To increase side-clearance of obstacles, Fox et al. (1997) suggests using a filtering function $\sigma(\cdot)$. A two-dimensional averaging (uniform) filter was applied with success in Eriksen (2014) and is also used in this report. An example of a smoothed

objective function is shown in Figure 4.1.

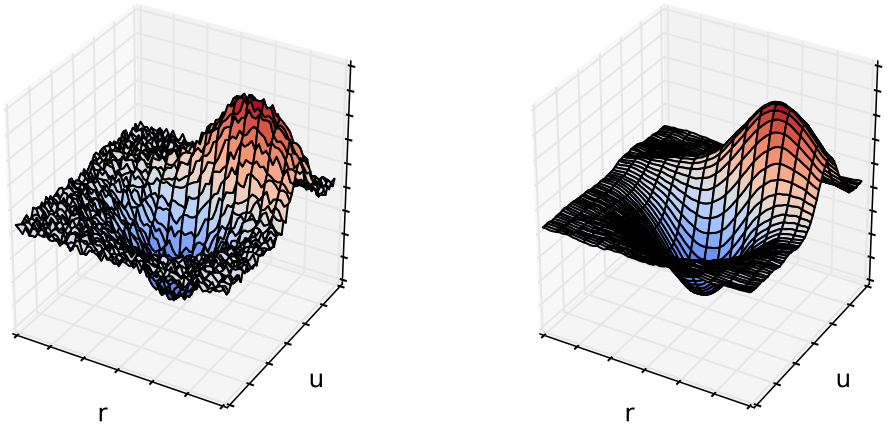


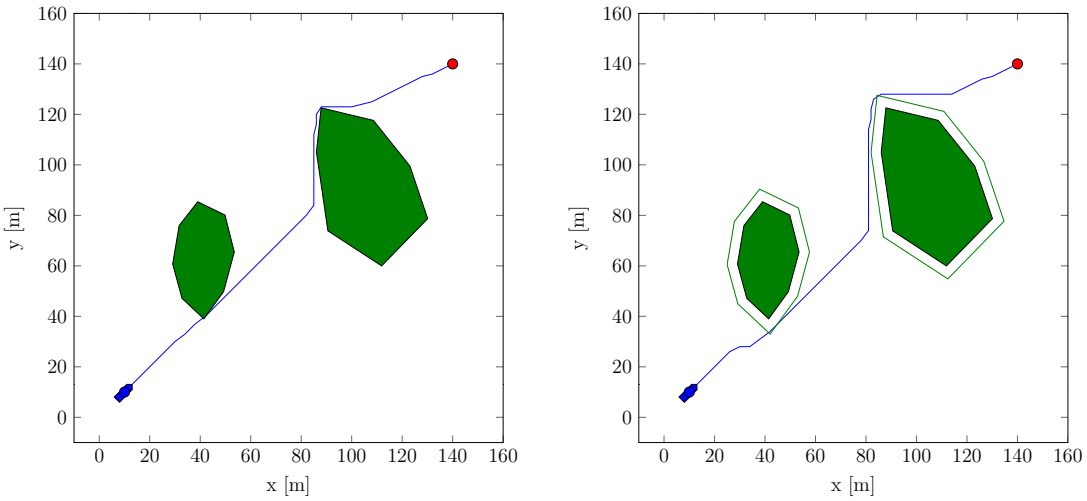
Figure 4.1: Two-dimensional smoothing. The function on the right is the result of a two-dimensional averaging (uniform) filter applied to the function on the left.

4.1.3 A* Search

From the results it is clear that with the basic implementation of A*, there are several issues that make this method less suitable for path planning for marine vessels. The paths found are in general infeasible as they may contain several 90 degree turns (Fig. 3.1c, p. 24). For low-speed WMRs, this is not a big problem, but for a underactuated ship such paths are impossible to follow. Some ad hoc methods may be used to improve the paths such as post processing with clothoid transitions and polynomial splines (Nord, 2010).

Furthermore, the algorithm has the problem of *contour hugging* (Fig. 4.2 and Fig. 3.1c). Contour hugging is the phenomena where the paths generated tend to "hug" the obstacles. This illustrates that the shortest path not necessarily is the best path. The most common mitigation is to use some form of safety region around the obstacles (Nord, 2010) (Fig. 4.2b). This is an easy and quick fix, albeit novel. Paths close to the shore is seldom advantageous and goes against best practices for navigation at sea. Other approaches suggest using methods based on Voronoi diagrams as guiding heuristics (see Sec. 4.1.4).

For guiding other algorithms, *e.g.*, RRT (Loe, 2008) or DWA (Marder-Eppstein et al., 2010), it can be quite effective.



(a) A* search without safety region

(b) A* search with safety region

Figure 4.2: A* search with and without safety region. An added safety region mitigates the dangerous paths generated, however it is a novel solution.

4.1.4 Hybrid-state A* Search

In this project, only a minimal implementation of the path planning scheme proposed in Dolgov et al. (2010) was implemented, nevertheless it shows very promising results. Apart for the obvious shortcomings with no re-planning in this implementation, it generates overall good paths, even without the trajectory smoothing algorithm proposed.

The basic hybrid-state A* shares the some of the same challenges as the classical A*, such as contour hugging (Sec. 4.1.3). Dolgov et al. (2010) proposes the usage of a Voronoi field to penalize proximity to obstacles. The advantage of using such a field over the more common potential field (Khatib, 1986) is these create high cost potential areas in narrow passages (Koren & Borenstein, 1991). Given a point $(x, y) \in \mathbb{R}^2$, the distance to nearest obstacle $d_{\mathcal{O}}$ and the distance to the nearest edge of the Generalized Voronoi Diagram (GVD) $d_{\mathcal{V}}$, the potential is defined as

$$\rho_V(x,y) = \begin{cases} \frac{\alpha}{\alpha + d_{\mathcal{O}}(x,y)} \cdot \frac{d_{\mathcal{V}}(x,y)}{d_{\mathcal{O}}(x,y) + d_{\mathcal{V}}(x,y)} \cdot \frac{(d_{\mathcal{O}} - d_{\mathcal{O}}^{\max})^2}{(d_{\mathcal{O}}^{\max})^2}, & d_{\mathcal{O}} \leq d_{\mathcal{O}}^{\max}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

This type of potential field is especially interesting for marine navigation as it normally is preferred to travel far from the shore. Moreover, it may be possible to utilize the field to achieve paths such that the Voronoi edges is at the port side of the vessel (Fig. 4.3).

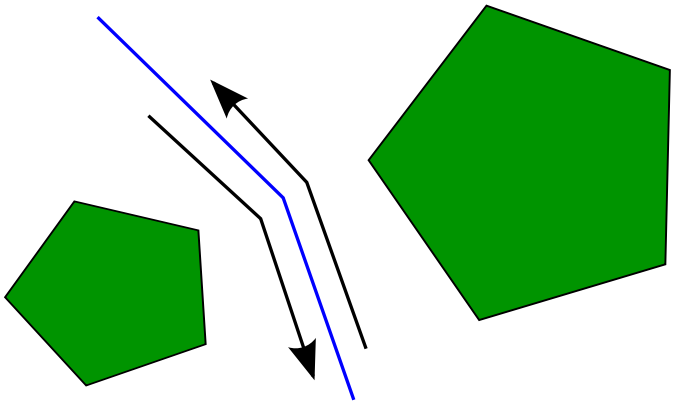


Figure 4.3: Using Voronoi fields for safe navigation at sea. By maintaining the Voronoi edge at the port side of the vessel, best practice navigation is ensured and the vessel motion is predictable to other (human) traffic.

4.2 Alternative Approaches

The number of approaches to the planning problem is vast, thus only a limited set of these is examined in detail in this report. However, some of the alternative methods deserve some discussion and would likely been implemented if not for the limited time available.

4.2.1 Local Methods in a Global World

Inherent to local methods is the problem of getting trapped in a local minima. The common solution to the minima trap is to apply some sort of heuristic or ad hoc method. But first, one need to identify the trap situation. In an ideal scenario this trap would be identified by simply monitoring the speed of the vehicle, which would come to a halt at the equilibria. Borenstein and Koren (1989) suggest a more practical solution by monitoring the direction of travel. If the direction of travel is more than 90 degrees off-target, the vehicle is very likely about to get trapped. The method proposed to escape this trap is wall-following (Borenstein & Koren, 1989). This however may result in vastly ineffective paths if the “wrong” direction along the wall is chosen (Fig. 4.4).

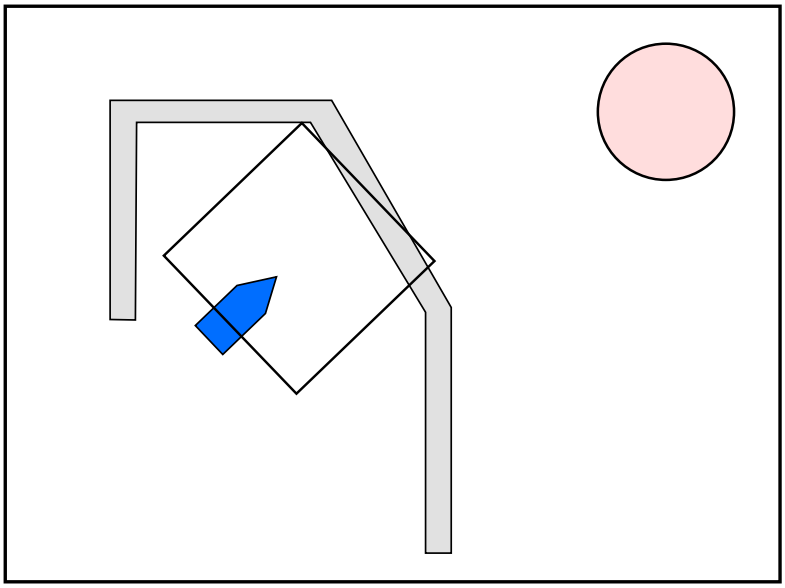


Figure 4.4: Choosing the wrong direction. If a local controller should choose the wrong direction (*i.e.* starboard), the path to the goal becomes much longer.

4.2.2 Global Methods for Collision Avoidance

One may argue that if re-planning was introduced for the global methods, they might work for collision avoidance as well. In the A* case, its infeasibility may cause oscillatory response as in the start of Scenario 1 (Fig 3.1c). Given such a response for each change of path is clearly undesirable. Hybrid-state A* on the other hand would perform much better in such a scenario. In fact, in the original implementation Montemerlo et al. (2008), Dolgov et al. (2010) it is used as the sole path planner during free navigation.

Another issue when using global methods for collision avoidance is that they are often too slow. For reliable collision avoidance, the their run time should be a fraction of a second, however most global methods use several seconds or even minutes, depending on the size and complexity of the configuration space. Using guiding heuristics and reusing data from previous searches, the computational time of A* based methods may be severely reduced. A variable resolution search may also speed up the search significantly. Other methods, such as using RRTs (see Section 4.2.3) are more difficult to speed up and still ensure reliable performance, due to their probabilistic nature. Dolgov et al. (2010) claims a run time in the range of 50-300 ms for a full re-planning cycle.

4.2.3 Rapidly-exploring Random Trees

First introduced in LaValle (1998), a planning method for handling nonholonomic constraints and high degrees of freedom. It may be used for global path planning.

In short, it explores a metric space X by considering a random state x_{rand} for which the nearest vertex x_{near} already present in the tree $\mathcal{T}$ is found. Then, the input $u \in U$ that minimizes some distance metric ρ is selected. Finally, by applying u to the system a new state x_{new} is generated and added to $\mathcal{T}$. In Figure 4.5 the result from a novel implementation of the RRT method is shown.

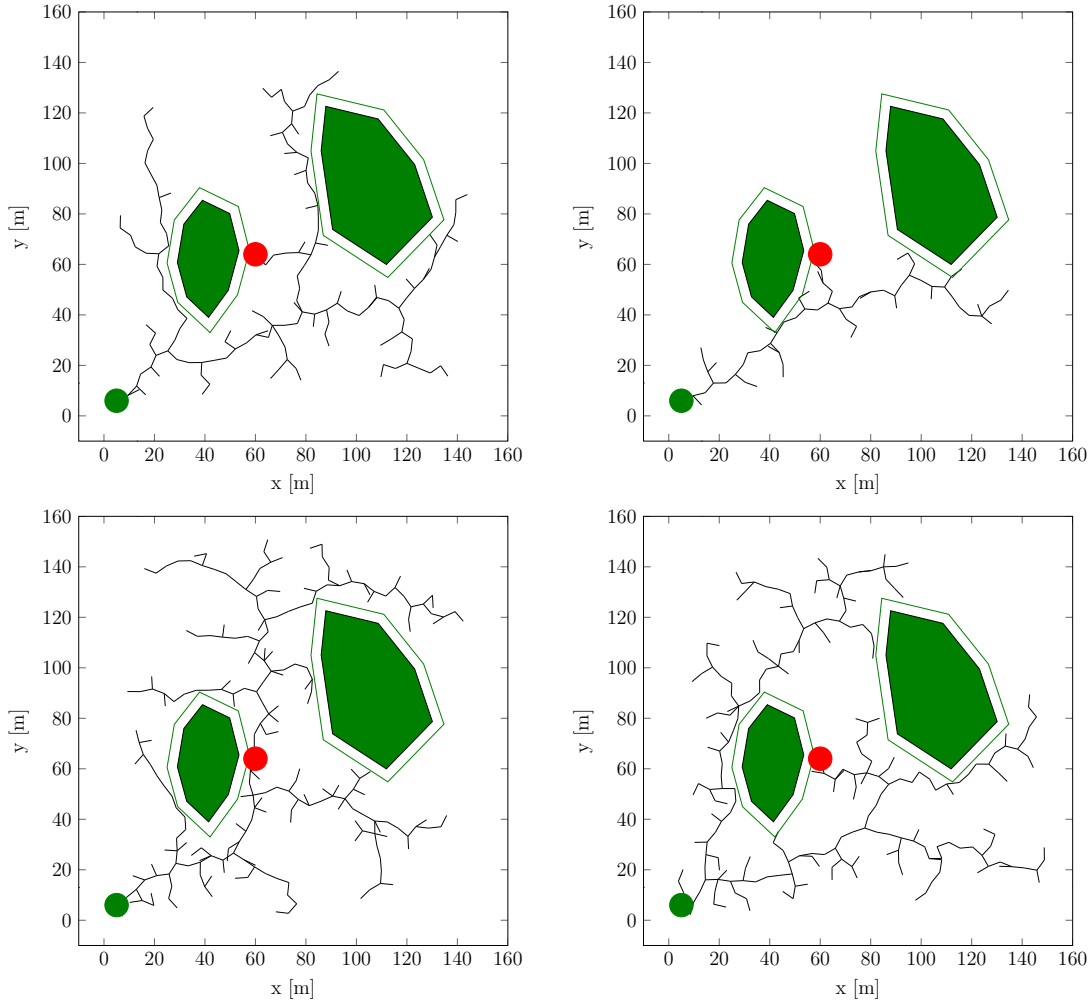


Figure 4.5: A simple Rapidly-exploring Random Tree implementation. The four different plots represents four simulations with the same initial conditions.

In Loe (2008) an A* guided RRT method was used for global navigation with

success. However, one may argue that there are other methods that are more suited and performs better for this purpose. The main arguments from Loe (2007) is that while the classical A* search finds a path quicker than RRT, it is in general more unsafe (mostly due to contour hugging). But, this is no guarantee that the RRT paths will be safe as well unless distance from obstacles is added as term in the cost function when selecting the path through the tree. Its flexibility and the fact that it provides dynamically feasible paths is perhaps not enough to outweigh its downsides. It is in general much slower than an A* based method, especially if heuristics and variable resolution searching is applied to speed up the search.

4.3 Other Considerations

The discussion in this chapter has for the most part considered an idealized world, however if a guidance system where to be implemented and used in a realistic setting, several considerations would have to be made.

The first aspect to consider when dealing with autonomous navigation at sea is adhering to the rules and regulation at sea (International Maritime Organization, 2003). Loe (2008) provides this functionality through the use of collision cones (Chakravarthy & Ghose, 1998; Benjamin, Leonard, Curcio, & Newman, 2006; Colito, 2007) and modifying the objective function of DWA as well as for RRT to penalize trajectories not adhering to COLREGS.

Another aspect is the degree of autonomy. Sheridan (1992) defines ten degrees

1. Computer offers no assistance; human does it all.
2. Computer offers a complete set of action alternatives.
3. Computer narrows the selection down to a few choices.
4. Computer suggests a single action.
5. Computer executes that action if human approves.
6. Computer allows the human limited time to veto before automatic execution.
7. Computer executes automatically then necessarily informs the human.
8. Computer informs human after automatic execution only if human asks.
9. Computer informs human after automatic execution only if it decides to.
10. Computer decides everything and acts autonomously, ignoring the human.

When designing an ASV it is important to keep this scale in mind. A typical ASV application may be in the range of 6-10 on this scale, dependent on the application.

Chapter 5

Conclusion

In this project thesis a flexible and modular simulation environment has been developed in the Python programming language, known algorithms (the virtual force field method, the dynamic window approach and A*) has been implemented in addition to the newer hybrid-state A* search. Unsurprisingly, the dynamic window approach performed best of the local approaches. In the case of the global methods, the hybrid-state A* outperforms the classical A*.

When considering the results in this report and the results obtained by others (Loe, 2007, 2008; Montemerlo et al., 2008; Dolgov et al., 2010; Larson et al., 2006, 2007), a natural candidate for a complete guidance system emerges: a hybrid approach with the hybrid-state A* search as the global method and the dynamic window approach as the local method. This is a system architecture that has proven itself robust in several practical settings Larson et al. (2006, 2007), Marder-Eppstein et al. (2010).

Appendix A

Simulator

As a part of the project a simulator for comparing planning algorithms was implemented in Python using NumPy, SciPy and Matplotlib. The focus has been to implement a modular and flexible program that is easy to extend. The source code for the simulator can be found at https://github.com/thomsten/project-thesis-simulator.

In the scientific community, especially in the control systems field, Matlab is by far the preferred framework for numerical programming. Therefore, a valid question is: *why Python?* There are many advantages with using Matlab:

- Good and fast vector mathematics
- Quality built-in functions
- Good plotting environments

However, not all is great with Matlab. It is a proprietary software that works well within its own ecosystem, but when one are to interface *other* systems (*e.g.* other programs or hardware) things get more difficult as it is not a general purpose programming language. It works very good for minor functions or scripts, but larger programs quickly get unstructured and difficult to maintain. Python on the other hand, is a general purpose programming language that is good for rapid prototyping. In conjunction with NumPy, SciPy and Matplotlib all of the functionality of Matlab is covered, except for Simulink (which there is no good alternative for).

A.1 Simulator Structure

The simulator is organized in a hierarchical manner (Fig. A.1). By implementing core modules as separate classes, it is believed that through this strict separation modularity and customizability is retained.

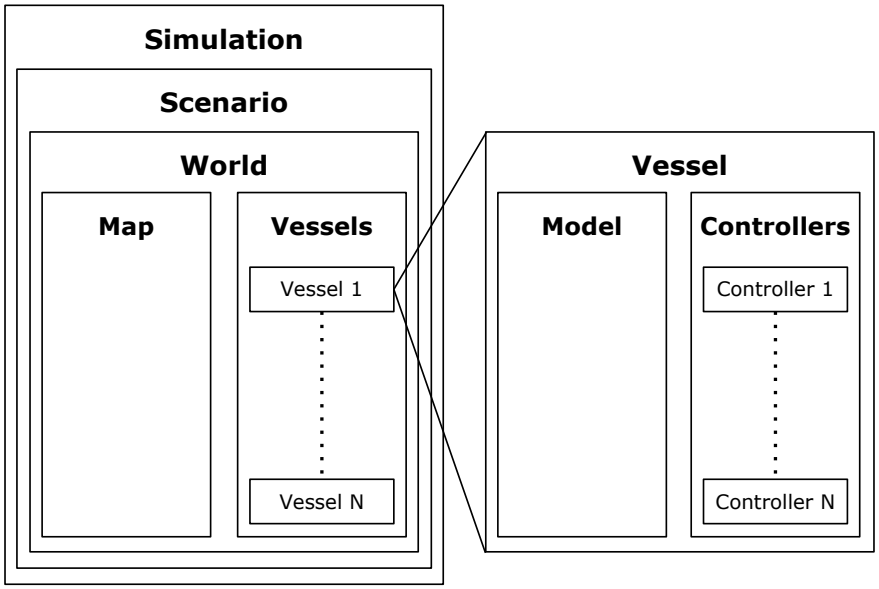


Figure A.1: General overview of simulator structure. The boxes represent classes which are organized in a hierarchical manner.

A.1.1 Simulation and Scenario Classes

At the topmost level lies the `Simulation` and `Scenario` classes. The `Simulation` class essentially performs the numerical integration necessary to simulate the system. It updates the `World` (which then updates its `Vessels`) and performs collision detection. If given figure and axes handles from Matplotlib it will perform a stepped simulation, *i.e.*, pause the simulation every n^{th} time step. The `Simulation` class also has the functionality of saving the simulation data (vessel paths, etc.).

The `Scenario` class is used to set up vessels with controllers, the world and simulation parameters in an easy way.

A.1.2 Map Class

The `Map` class implements a general purpose map. In the simulator a map consists of a set of `Polygons` (islands) with an optional safety region. A discretized version of the map can also be generated given a grid resolution. The discretization is used to speed up the planning algorithms as they tend to test a great deal of points and the algorithm for testing if a point is in a polygon is much slower than an array access. The discretization method is based on Finley (2007a), originally a polygon fill algorithm. A point-in-polygon algorithm was implemented according to Finley (2007b).

To efficiently generate a safety region around a `Polygon`, an algorithm for extruding the `Polygon` was developed. The approach chosen was to extrude each edge of the `Polygon` a given distance. The algorithm is summarized in Algorithm A.1. The algorithm parameters are illustrated in Figure A.2.

Algorithm A.1 Polygon extruding algorithm

Require: Polygon P, extruding length ℓ

for all Vertices $v_i \in P$ **do** ▷ Calculate direction vectors and angles for each vertex

$\vec{a} \leftarrow (v_{i-1} - v_i)/\|v_{i-1} - v_i\|$

$\vec{b} \leftarrow (v_{i+1} - v_i)/\|v_{i-1} - v_i\|$

$c_i \leftarrow (\vec{a} + \vec{b})/\sqrt{2}$

$\theta_i \leftarrow \angle(\vec{a}, \vec{b})$

end for

for all Vertices $v_i \in P$ **do** ▷ Extrude polygon

$v_{new,i} \leftarrow v_i - \frac{\sqrt{2}\ell}{\sqrt{1-\theta_i}}\vec{c}_i$

if $v_{new,i} \in P$ **then**

$v_{new,i} \leftarrow v_i + \frac{\sqrt{2}\ell}{\sqrt{1-\theta_i}}\vec{c}_i$

end if

end for

return $P_{new} \leftarrow \{v_{new,0}, \ldots, v_{new,N}\}$

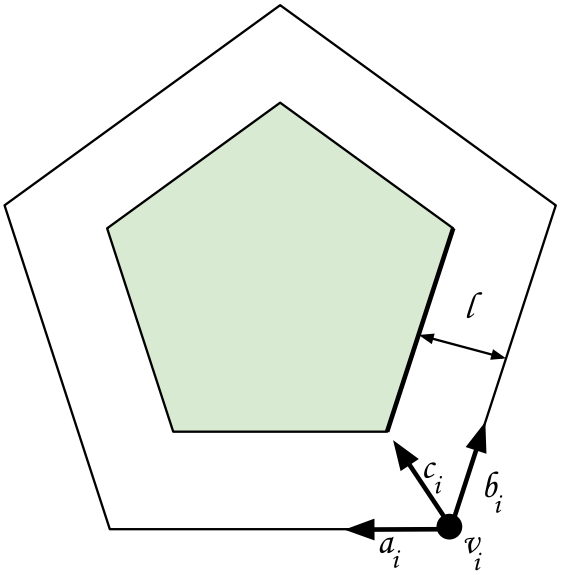


Figure A.2: Polygon extruding

A.1.3 Vessel Class

The `Vessel` class is an implementation of a general surface vehicle. It provides functionality such as drawing, animating, etc., and contains an instance of a `VesselModel` class. The `VesselModel` class implements a general 3-DOF non-linear vessel model and performs the necessary numerical integration to simulate the vessel motion. It defaults to the parameters used in Loe (2008), given in Table A.1.

A.1.4 Controller Class

Each *controller* or planning algorithm is implemented as a subclass of the `Controller` superclass. The implementation of the Virtual Force Field and Dynamic Window is somewhat based on Loe (2007).

The following controllers are implemented in the simulator:

- Virtual Force Field
- Dynamic Window
- A*
- Hybrid-state A*

Table A.1: Vessel parameters for the Viknes 830. Note that it is assumed $\mathbf{M}_A = \mathbf{C}_A = 0$, which yields a slightly unrealistic model, however it captures the vessel dynamics good enough.

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

- A novel Rapidly-exploring Random Tree implementation
- Line of Sight (LOS) guidance
- Pure Pursuit (PP) guidance
- Constant Bearing (CB) guidance

A.2 Other Features

Other features of the simulator include direct exporting of plots to Ti*k*Z/PGF, animation of simulations and exporting these as movie files and much more. For examples of animated simulations, see: https://www.youtube.com/watch?v=MeUAwhdm6WY&list=PLCb52_fHWPzwJ7bbfdDRcSZeVITlK8g4

Bibliography

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. doi:10.1002/rob

Borenstein, J. & Koren, Y. (1989, September). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, *19*(5), 1179–1187. doi:10.1109/21.44033

Breivik, M. & Fossen, T. I. (2004). Path following for marine surface vessels. In *Oceans'04. mtts/ieee techno-ocean'04* (Vol. 4, pp. 2282–2289). IEEE. doi:10.1109/OCEANS.2004.1406507

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. doi:10.1109/3468.709600

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). Retrieved from https://digital.lib.washington.edu/researchworks/handle/1773/3112

Dahl, A. R. (2013). *Path Planning and Guidance for Marine Surface Vessels* (Master's thesis, Norwegian University of Science and Technology).

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. Retrieved from http://www.aaai.org/Papers/AAAI/2007/AAAI07-187.pdf

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271. doi:10.1007/BF01386390

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. doi:10.1177/0278364909359210

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

Dubins, L. E. (1957, July). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, *79*(3), 497–516. doi:10.2307/2372560

Eriksen, B.-O. H. (2014). *Horizontal Collision Avoidance for Autonomous Underwater Vehicles* (Project Report, Norwegian University of Science and Technology).

Ferguson, D. & Stentz, A. [Anthony]. (2005). Field D*: An interpolation-based path planner and replanner. In *Proceedings of the international symposium on robotics research (isrr)*. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-48113-3_22

Finley, D. R. (2007a). Efficient Polygon Fill Algorithm With C Code Sample. Retrieved December 19, 2014, from http://alienryderflex.com/polygon_fill/

Finley, D. R. (2007b). Point-In-Polygon Algorithm – Determining Whether A Point Is Inside A Complex Polygon. Retrieved December 19, 2014, from http://alienryderflex.com/polygon/

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

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. Retrieved from http://cheonji.kaist.ac.kr/ee683/lecture_note_2013/Dynamic%20Window%20Approach.pdf

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

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. doi:10.1109/TSSC.1968.300136

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

Kavraki, L., Svestka, P., Latombe, J.-C., & Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE*

Transactions on Robotics and Automation, *12*(4), 566–580. doi:10.1109/70.508439

Khatib, O. (1986, March). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, *5*(1), 90–98. doi:10.1177/027836498600500106

Koren, Y. & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the 1991 ieee international conference on robotics and automation* (pp. 1398–1404). Sacramento, California: IEEE Comput. Soc. Press. doi:10.1109/ROBOT.1991.131810

Larson, J., Bruch, M., & Ebken, J. (2006). Autonomous navigation and obstacle avoidance for unmanned surface vehicles. *Proceedings of SPIE*, *6230 I*, 17–20. doi:10.1117/12.663798

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. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA475547

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. Retrieved from http://143.132.8.23/robotics/CSC539-14S/Papers/06%20RRT.pdf

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511546877

Lekkas, A. M. (2014). *Guidance and Path-Planning Systems for Autonomous Vehicles* (PhD thesis, Norwegian University of Science and Technology). Retrieved from http://www.diva-portal.org/smash/record.jsf?pid=diva2:727945

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). Retrieved from http://www.diva-portal.org/smash/record.jsf?pid=diva2:347606

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*. Retrieved from http://wiki.ros.org/Papers/ICRA2010_Marder-Eppstein

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. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/rob.20258/abstract

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 December 9, 2014, from http://www.ros.org/

Reeds, J. & Shepp, L. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, *145*(2), 367–393. Retrieved from http://msp.org/pjm/1990/145-2/p06.xhtml

Sheridan, T. B. (1992). *Telerobotics, Automation, and Human Supervisory Control*. MIT press. Retrieved from http://www.google.no/books?hl=no&lr=&id=eu41_M2Do9oC&pgis=1

Simmons, R. & Henriksen, L. (1996). Obstacle avoidance and safeguarding for a lunar rover. In *Proceedings from the aiaa forum on advanced developments in space robotics*. Madison, WI. Retrieved from https://www.cs.cmu.edu/afs/cs/usr/reids/www/home/papers/AIAAobsAvoid.pdf

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. Retrieved from http://books.google.no/books?id=sZ_bOwAACAAJ

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

Stentz, A. [A.]. (1994). Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 ieee international conference on robotics and automation* (pp. 3310–3317). IEEE Comput. Soc. Press. doi:10.1109/ROBOT.1994.351061

Wilde, D. K. (2009, October). Computing clothoid segments for trajectory generation. In *2009 ieee/rsj international conference on intelligent robots and systems* (pp. 2440–2445). IEEE. doi:10.1109/IROS.2009.5354700