ISSN 0280-5316
ISRN LUTFD2/TFRT--5717--SE

Optimizing Yacht Routes using Dynamic Programming

Mikael Lindberg

Department of Automatic Control
Lund Institute of Technology
December 2003

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
January 2004

Document Number
ISRN LUTFD2/TFRT--5717--SE

Author(s)
Mikael Lindberg

Supervisor
Prof. Rolf Johansson
Anders Robertsson

Sponsoring organisation

Title and subtitle
Optimizing Yacht Routes using Dynamic Programming (Optimering av segelbåtsrutter med dynamisk programmering)

Abstract

A method for generating minimum time yacht routes for short distances while avoiding stationary obstacles and taking into account wave and water current disturbances is sought. The method should be feasible for use in an autopilot type construction. A solution is found using an iterative forumulation of the dynamic programming algorithm on a grided state space and an implementation in ANSI C is produced. Several other approaches are also examined. Finally, some aspects of constructing an autopilot based on the algorithm are discussed.

Key words
Optimization, Yacht Routing, Dynamic Programming

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
36

Recipient's notes

Security classification

Optimizing Yacht Routes using Dynamic Programming

Mikael Lindberg

Revision : 1.11

February 16, 2004

Contents

1. Background

Controlling a sailing vessel is in many ways unlike controlling motorized water vehicles. Though the two tasks share properties related with currents, wave disturbances and inertia, the control performance of the sailing vessel is perhaps most limited by its dependence on wind. The speed and direction of the wind determines not only the possible speed of the sailing vessel, but also what trajectories that are feasible.

Sailing vessels cannot travel directly towards the wind. Combined with the fact that the maximum attainable speed is not only determined by the speed of the wind but also depends nonlinearly on the angle between the wind and the heading of the boat, this makes choosing a good trajectory non-trivial and choosing an optimal one difficult. Other complicating factors are disturbances, caused by waves and water currents, as well as the the limited braking capabilities of the sailing vessel. A skilled navigator can do fairly well on common sense and experience but since computers started to become more and more powerful, it is not uncommon for professional yacht racers to make heavy use of computerized optimization for helping out. Much of this software is proprietary and expensive, forcing most hobbyist navigators to cope without them. This thesis is my attempt in bringing some help to those of us without large corporate sponsors that still would like to win a race or two.

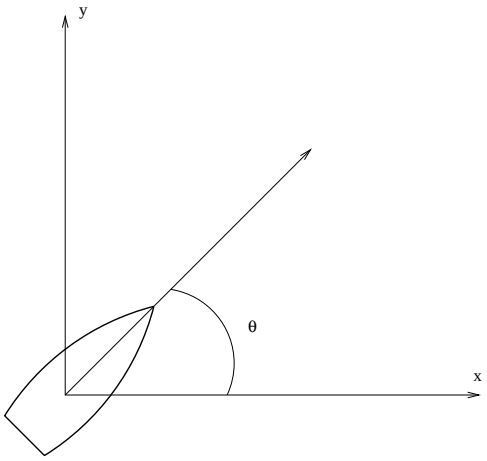

Figure 1 The boat in a Cartesian coordinate system with the heading θ

2. Problem Formulation

This thesis aims to answer how to compute a minimal time path for taking a sailing boat between two predefined points given boat dynamics and wind conditions and while avoiding stationary obstacles. A secondary objective is to prototype an autopilot able to steer a sailing boat along this course. As such, the solution must be possible to implement on a computer or device that can reasonably be fitted on the boat.

3. Modeling a Sailing Boat

A sailing boat is subject to a lot of non-linear dynamics. A commonly known example of this is that the boat cannot sail directly towards the wind. More precisely, the maximum speed of the boat in any given direction depends nonlinearly on the relative angle between the boat heading and wind direction, but also quite logically on the wind speed. A simple and intuitive model of this in state space form would be

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v(\theta)\cos(\theta) \\ v(\theta)\sin(\theta) \\ f(u) \end{pmatrix} \tag{1}$$

where x and y denote the Cartesian coordinates of the boat and θ is the heading of the boat as shown in Figure 1.

From here we can use for instance forward difference approximation of the derivatives, which will give us the discrete time model

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x(k) + hv(\theta(k))\cos(\theta(k)) \\ y(k) + hv(\theta(k))\sin(\theta(k)) \\ \theta(k) + hf(u(k)) \end{pmatrix} \tag{2}$$

where h is the sampling period and k_h some constant depending on h. Given the focus of this thesis and the typical update frequency of a normal

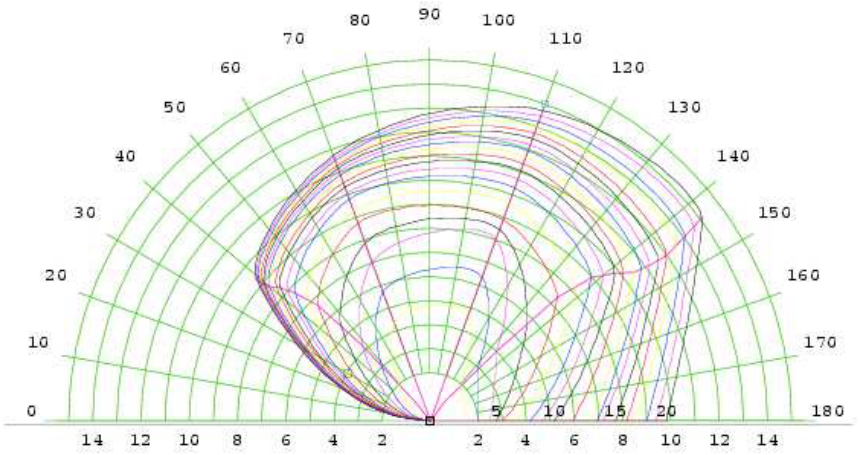

Figure 2 A plot of a series of measured maximum speeds for one given boat and wind speed. The diagram was borrowed with permission from [3].

GPS unit, the sampling period will be 1 second unless another value is given specifically.

3.1 Modeling Wind Dependent Dynamics $v(\theta)$

The complex dynamics of the interaction between the wind and the boat will make calculating the function $v(\theta)$ from theoretical models very difficult. Today, many professional boat builders use advanced software to estimate the maximum boat speed from properties such as hull shape and rig setup. These programs (Velocity Prediction Programs or VPPs) are expensive and many sailors resolve to the more traditional practice of empirical measuring. The resulting data are commonly presented in a polar plot called the *speed curve*. Figure 2 shows an example of this curve for a typical boat.

Estimates of the speed curve in between the data points can then be calculated using several different techniques.

Linear Interpolation The tabular form of the gathered data makes linear interpolation an attractive choice. It is robust for non-continuous phenomena and computationally cheap. The most obvious drawback is that it is tricky to handle for analytical computations.

Polynomial Approximation Polynomial approximation makes analytical calculations much simpler and can be even more computationally effective than linear interpolation. Its biggest drawback is that making a good fit can require substantial user interaction in choice of polynomial degree and handling non continuous phenomena. Figure 5 shows what can happen when using a cubic spline approximation.

Periodic Expansion Calculating the relative angle between boat and wind can involve a lot of non differentiable operations such as modulo π and $abs()$. An interesting way to handle this would be to make an even periodic approximation using the cosine half period expansion. For an explanation

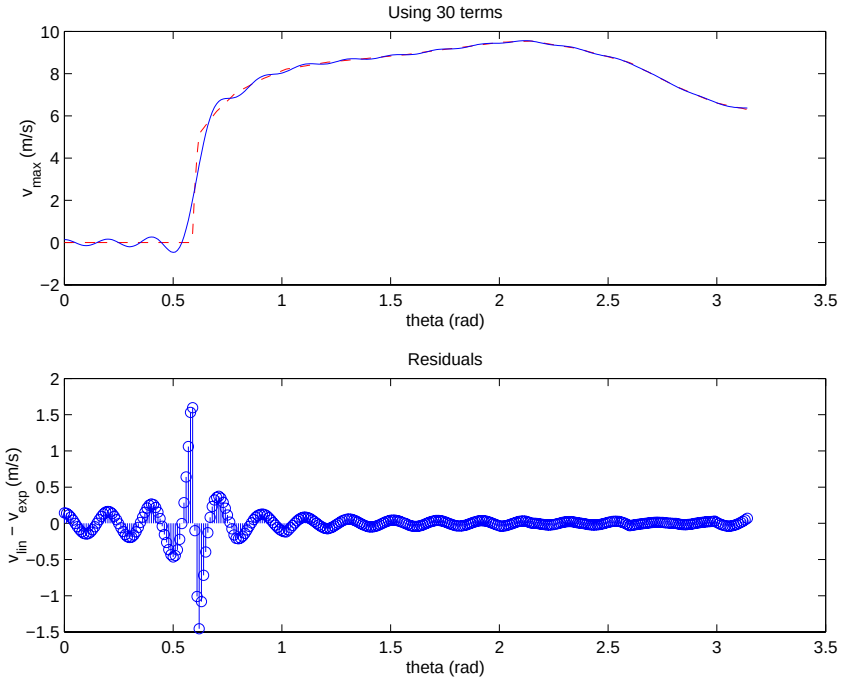

Figure 3 Large periodic residuals due to discontinuities. The dotted line represents the curve from the linear interpolation.

of the theory see e.g. [5]. The approximation of $v(\theta)$ will here be given by:

$$v(\theta) = \frac{a_0}{2} + \sum_{k=1}^{N} a_k \cos(\frac{k\pi(\theta - \theta_v)}{L}) \tag{3}$$

$$L = \pi \tag{4}$$

which gives

$$v(\theta) = \frac{a_0}{2} + \sum_{k=1}^{N} a_k \cos(k(\theta - \theta_v)) \tag{5}$$

where θ_v is the direction of the wind.

The most apparent advantage of this approach is that it gives a differentiable approximation while on the other hand being much more computationally expensive. We can see from Figure 6 that this approach suffers from problems similar to those of the spline approximation, although of a smaller magnitude. The ripples (commonly known as Gibb's phenomenon) are to be expected when approximating discontinuous functions in this manner. The effect decreases when using more coefficients but is never eliminated. Some improvements might be gained from modifying the speed curve so that it is smoother. In Figure 3 we can see large periodic errors when comparing the linear interpolation and the expansion (this time for 30 coefficients) while in Figure 4 the effect is reduced. Depending on the specific problem and choice of optimization routine, this may or may not affect the results.

Using a large number of coefficients might pose a performance problem in numerical optimizations, but since the computations are based on sums

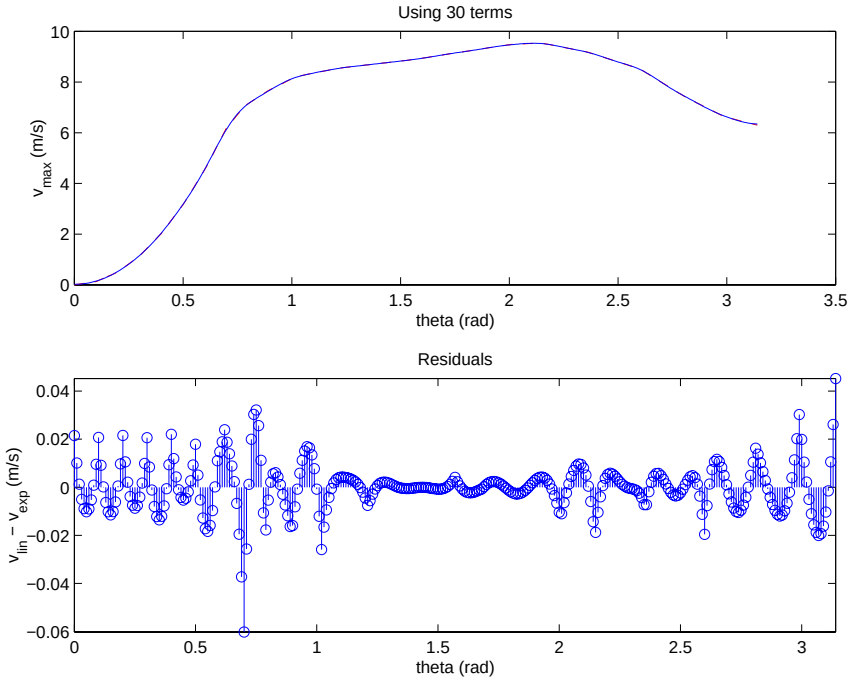

Figure 4 The residuals are reduced considerably by smoothing the curve

much could be done in parallel. If applied to a gridded state space, precalculated values for all angles in the grid would also increase speed. Many moderns CPUs support so called SIMD-instructions[1] which could perhaps be used to achieve even better performance.

Dealing with Stochastic Wind While it is common to regard the wind as a parameter that changes slowly over time, typically being of constant direction and speed for hours, in reality it behaves much more chaotically and the usual measurements should be considered a time average. Often used as a text book case for stochastic prediction models, a lot could be said on this subject. In this thesis we will disregard the faster dynamics and consider the wind to be constant in direction and speed over time, but may vary from one place to another.

3.2 Turning Dynamics

The turning dynamics of the boat modeled by the function $f(u)$ are also non-linear. The maximum rate at which the boat can turn is limited by a number of factors including:

- The size and shape of the hull
- The current speed of the boat
- The maximum rudder angle (usually 90° relative to the heading of the boat)

[1]Single Instruction, Multiple Data-Instructions. Typically used in geometry and signal processing applications where convolutions and other sum-based operations make up a large part of the computational burden.

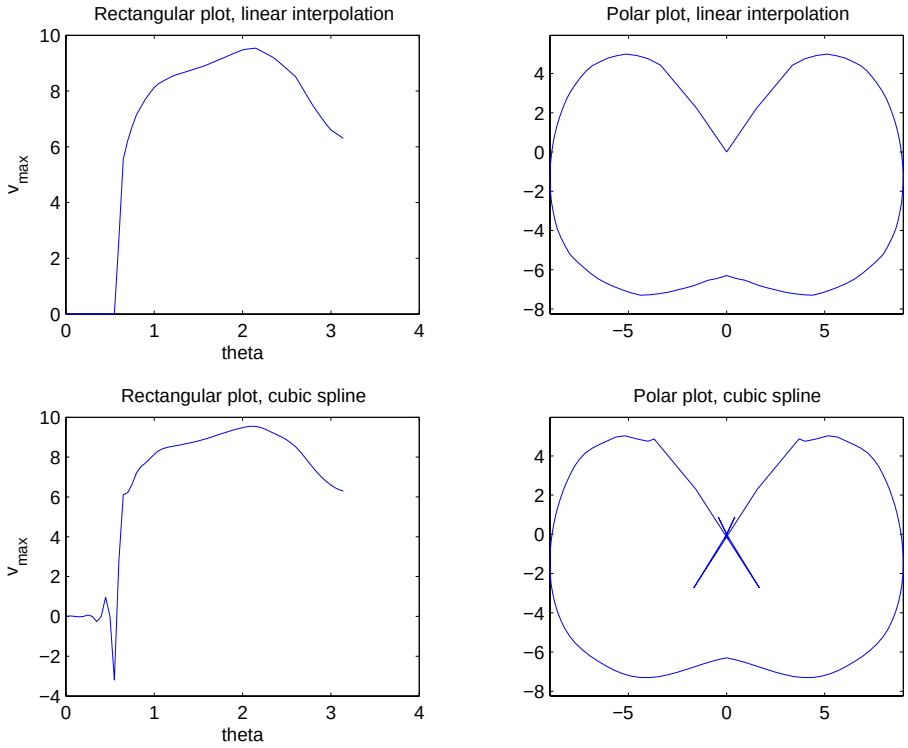

Figure 5 Comparison between different approximation methods

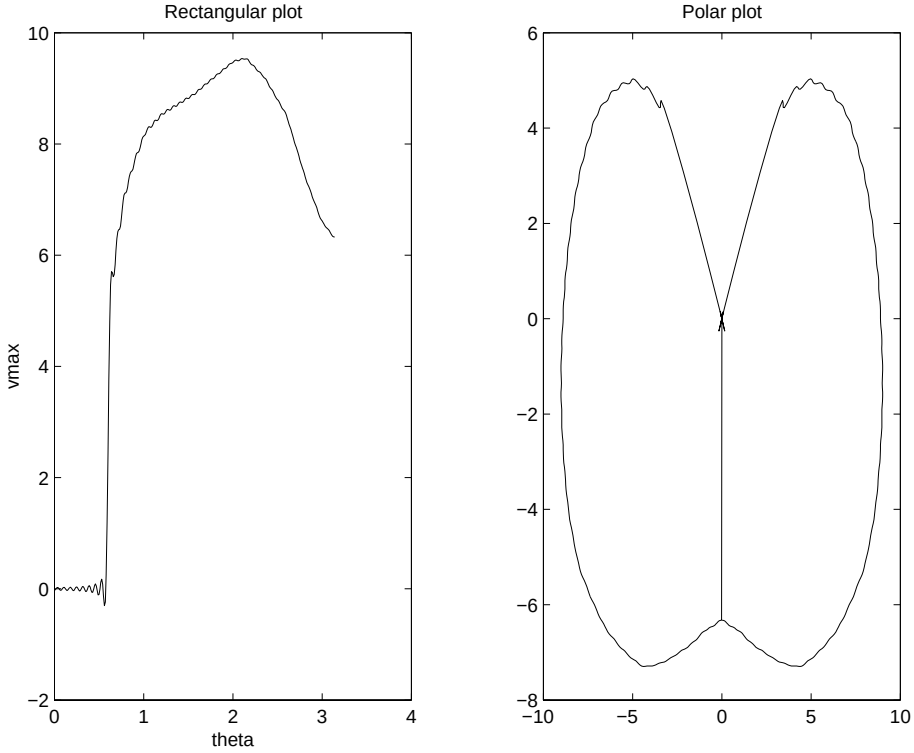

Figure 6 The cosine half-period expansion using 100 coefficients

In this thesis we will approximate $f(u)$ with an ordinary saturation under the following assumptions

- The boat will always move with maximum speed. A typical boat can reach maximum speed within seconds of changing course.
- The boat will make small changes to its course except when tacking. A turn usually takes only a few seconds and inertia will keep the boat going while moving over a slow section of the speed curve.

The inclusion of turning dynamics is a matter of scale. For optimization over long distances, the significance of turning decreases while it is important when trying to navigate in tight spaces and short time horizons.

3.3 Disturbances

When considering the boat dynamics over shorter distances one has to take into account two significant disturbances, namely water currents and waves.

Water Currents Water currents are typically constant over a long period of time but can vary quite a bit from one location to another. Their contribution to the dynamics will be modeled as stationary disturbances to x and y that vary with the current location. Currents would also be significant for calculationg longe distance routes.

Wave Disturbances Riding up and down the waves will cause the boat to turn and we will therefore have to consider their influence when trying to stay on course. The waves are periodic in nature and a skilled sailor will quickly adapt to their rhythm and compensate with the rudder. An autopilot has no easy way of seeing the waves and can only use the measurement of the current boat heading when determining the wave disturbances. This can be modeled using an ARMAX-model as follows:

$$\theta(k+1) = \theta(k) + hbu(k) + C(z^{-1})e(k) \tag{6}$$

where b is a constant gain determined by linearization of $f(u)$, $e(k)$ is the innovations process and the $C(z^{-1})$-polynomial will have to be determined for each occasion. From this it is a straight forward task to determine a linear predictor (e.g., a Kalman filter) for determining the wave influences. See Chapter 10 for more on this.

3.4 The Final Model

Combining the discrete time model with the disturbance models we get

$$\begin{pmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{pmatrix} = \begin{pmatrix} x(k) + hv(\theta(k))\cos(\theta(k)) + d_x(x,y) \\ y(k) + hv(\theta(k))\sin(\theta(k)) + d_y(x,y) \\ \theta(k) + hbu(k) + C_w(z^{-1})e(k) \end{pmatrix} \tag{7}$$

In this thesis I will study a hypothetical boat with the speed curve shown in Figure 7.

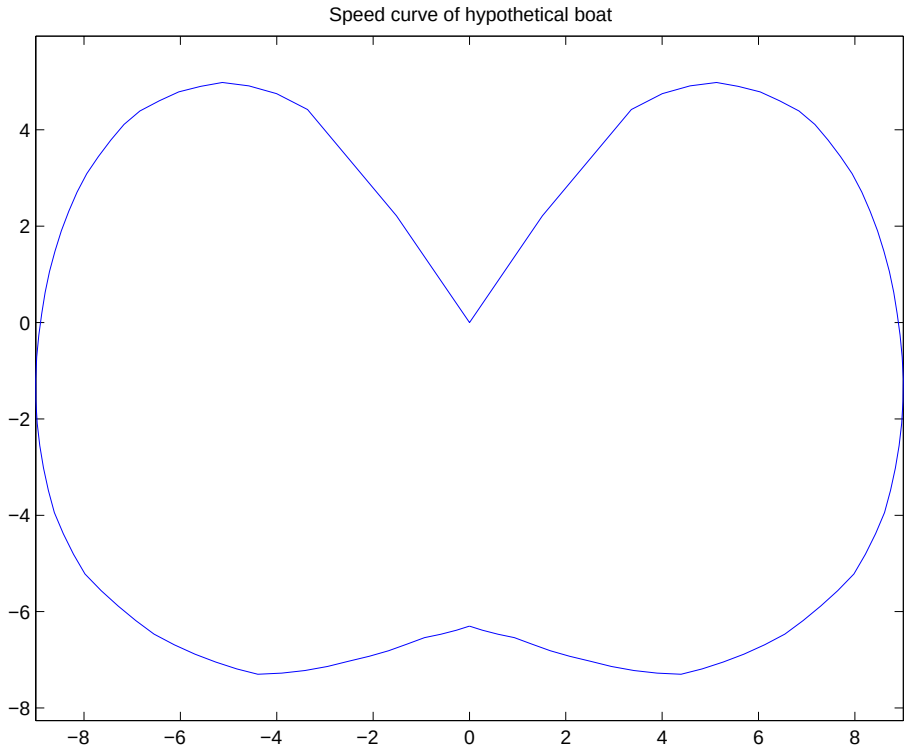

Figure 7 The speed curve of the hypothetical boat used in this thesis

4. Hamiltonian optics and sailing

This chapter is mostly a summary of [2]. The figures are also from that article.

4.1 Approach

Light travels on time optimal paths from one point to another. This is assured by Fermat's principle and is used in Hamiltonian optics. By reformulating this into the language of sailing, an interesting application arises. While the sailing boat has its speed curve, light moving through and anisotropic material has an *indicatrix*, which is a polar plot conceptually much like the speed curve, but ellipsoidal in shape and without the slow parts.

Consider a boat which after starting out always will sail in a time optimal way. The possible locations for this boat after some time t would be described by a curve analogous to the light wavefront described by Hamilton's time space function $T(\mathbf{r}) = t$. The evolution of the wavefront and the paths of the boat to reach a point on the curve is governed by Huygens' principle and the shape of the speed curve.

If $\Delta t = t_1 - t_0$ is sufficiently small, Huygens' principle tells us that we can acquire the next evolution of the wavefront by placing the speed diagram at all points along $T(\mathbf{r}) = t_0$ and scaling them by Δt (so to change the speeds into appropriate distances). The envelop curve touching the outer edges of the speed diagrams will then be $T(\mathbf{r}) = t_1$.

If you want to advance as fast as possible in a specific direction $\mathbf{p}$, you draw a tangent to the speed curve that is perpendicular to $\mathbf{p}$. You then

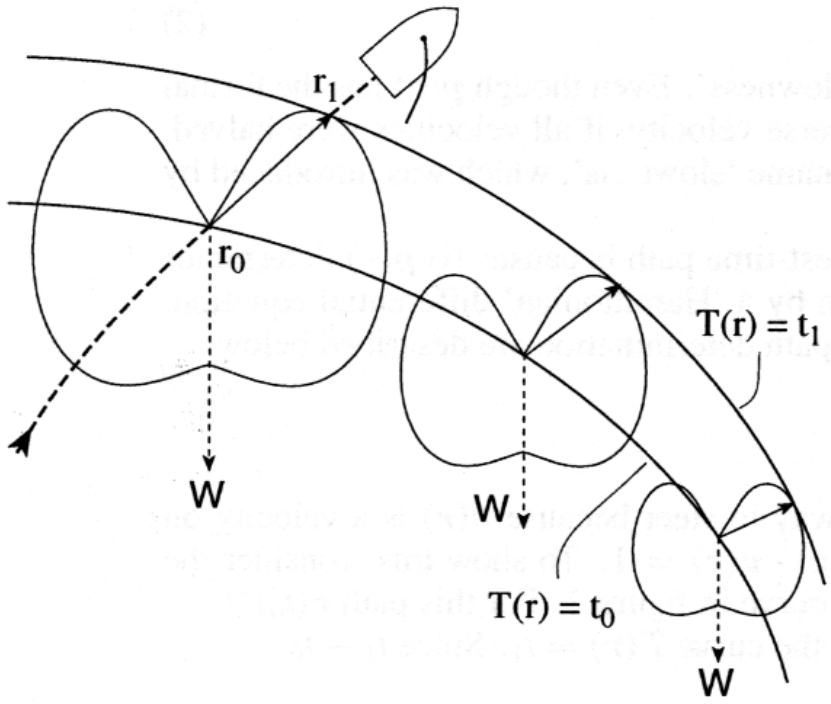

Figure 8 Two curves showing the possible position of the boat at times t_0 and t_1. The dotted curve shows the time-minimizing path for a sailboat which passes though r_0 at time t_0.

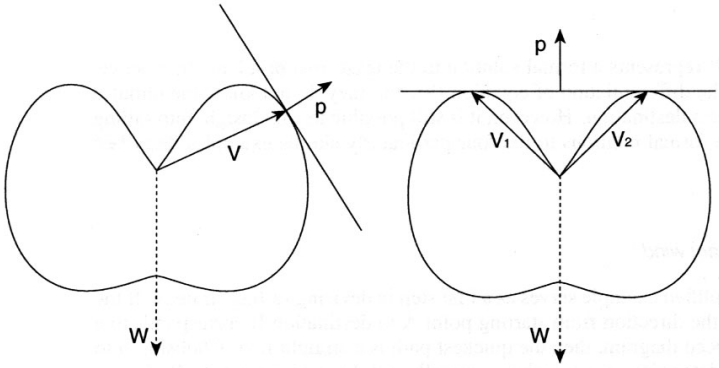

Figure 9 How to find the direction **v** to sail when you want to advance as fast as possible along the direction **p** given the wind direction **W**. Note that in general, **p** and **v** are not parallel

draw a vector from the origin of the speed curve to the point where the tangent touches. This vector will point in the most advantageous direction. Note that this is in general not parallel to **p**. Two examples of this is shown in Figure 9.

4.2 Results

The way to advance fastest in any direction can be found by applying a tangent to the speed curve perpendicular to the desired direction. The point at which the tangent touches the speed curve represents the direction in which to sail. The speed curve would of course have to be reorientated in order to reflect the direction of the wind.

4.3 Conclusion

This is an elegant theory and despite its many limitations not entirely without uses. In this form it cannot answer how to get to any specific point but it is usable for other situations, like getting to the shore as fast as possible or crossing a large body of water. Tacking costs or trajectory constraints cannot be easily introduced and although the article does show how to deal with time and position dependent wind, you need deterministic information in order to make the computations.

5. The RIOTS Toolbox

RIOTS[2] is a Matlab toolbox by Adam Schwartz for solving optimal control problems. It solves a large class of finite-time problems and allows for a variety of constraints and cost functions. It assumes that the control signal(s) can be expressed as spline functions and optimizes the coefficients in these functions.

RIOTS allows dynamics to be expressed in continuous time, but it is not clear how (or if) it could be used to handle stochastic disturbances as the version available at the time of writing only supports optimization of one goal function.

5.1 Approach

The objective is to calculate an optimal trajectory and the corresponding optimal control sequence using RIOTS. Should the performance of the calculations be good enough, that is if they can be completed fast enough using limited hardware, it would be possible to base an autopilot on this software. Since speed and memory usage are an issue, the boat model is implemented in C rather than as Matlab functions for better results.

Disturbances We disregard the water currents and the wave disturbances for the moment.

The handling of free final time RIOTS is built around a fixed final time problem formulation, but suggests that free final time can be handled by augmenting the system dynamics with an additional state (two for non-autonomous problems). RIOTS can then be configured to see that state as a free choice variable and try to find the appropriate time interval that would fit the optimal trajectory. More on this can be found in Section 2 of the RIOTS manual.

With the augmented state t_f the model becomes:

[2]RIOTS stands for Recursive Integration Optimal Trajectory Solver. More on RIOTS can be found in [4]

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{t_f} \end{pmatrix} = \begin{pmatrix} t_f v(\theta)\cos(\theta) \\ t_f v(\theta)\sin(\theta) \\ t_f u \\ 0 \end{pmatrix} \tag{8}$$

5.2 Results

A few possibly non-optimal trajectories could be computed, while most optimizations failed. No apparent connection between initial values, the number of decision points or initial guess for control signal and optimization success was found. No improvement could be observed when using user supplied gradient expressions instead of finite differences.

5.3 Conclusions and Remarks

The results from RIOTS seem unreliable. Often the optimization fails with error messages like 'linear search failed' or 'maximum number of iterations reached'. Changing the initial guess of the control sequence and the number of decision points sometimes helps but I have not been able to find any pattern in this. When the optimization actually finishes normally, the solutions are often found to be suboptimal.

For instance, in an example with the wind angle $\pi/4$ and starting in (-10, 0), RIOTS computed the trajectory that is seen in Figure 10, which is quite obviously suboptimal. In this case the initial guess of the control sequence was $[0\,0\,...\,0](=u_1)$ and the final time was 12.74 seconds. With the initial control signal u_2 chosen as $-u_1$ RIOTS gives the solution found in Figure 11. The final time was 12.73, that is slightly better. This suggests that there are many local minima close to the optimal solution that will make it very difficult to find the actual minimum.

I have tried switching from finite difference approximation of the gradients to user supplied gradients but without any improvement.

The supplied gradient expressions are:

$$\nabla f(x,u) = \begin{pmatrix} 0 & 0 & \frac{\partial \dot{x}}{\partial \theta} & v(\theta)\cos(\theta) \\ 0 & 0 & \frac{\partial \dot{y}}{\partial \theta} & v(\theta)\sin(\theta) \\ 0 & 0 & 0 & u \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{9}$$

where

$$\begin{aligned} \frac{\partial \dot{x}}{\partial \theta} &= t_f \left(\sum_{k=1}^{n} -a_k k \sin(k(\theta-\theta_v))\right)\cos(\theta) - \\ &\quad t_f \left(\frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(k(\theta-\theta_v))\right)\sin(\theta) \\ \frac{\partial \dot{y}}{\partial \theta} &= t_f \left(\sum_{k=1}^{n} -a_k k \sin(k(\theta-\theta_v))\right)\sin(\theta) + \\ &\quad t_f \left(\frac{a_0}{2} + \sum_{k=0}^{n} a_k \cos(k(\theta-\theta_v))\right)\cos(\theta) \end{aligned} \tag{10}$$

(θ_v is the direction of the wind) and for the final cost function $g(t, z_0, z_f)$:

$$g(t, z_0, z_f) = x_f^2 + y_f^2 \tag{11}$$

$$\frac{\partial g}{\partial x_f} = 2x_f, \frac{\partial g}{\partial y_f} = 2y_f, \frac{\partial g}{\partial \theta_f} = 0, \frac{\partial g}{\partial t_f} = 0 \tag{12}$$

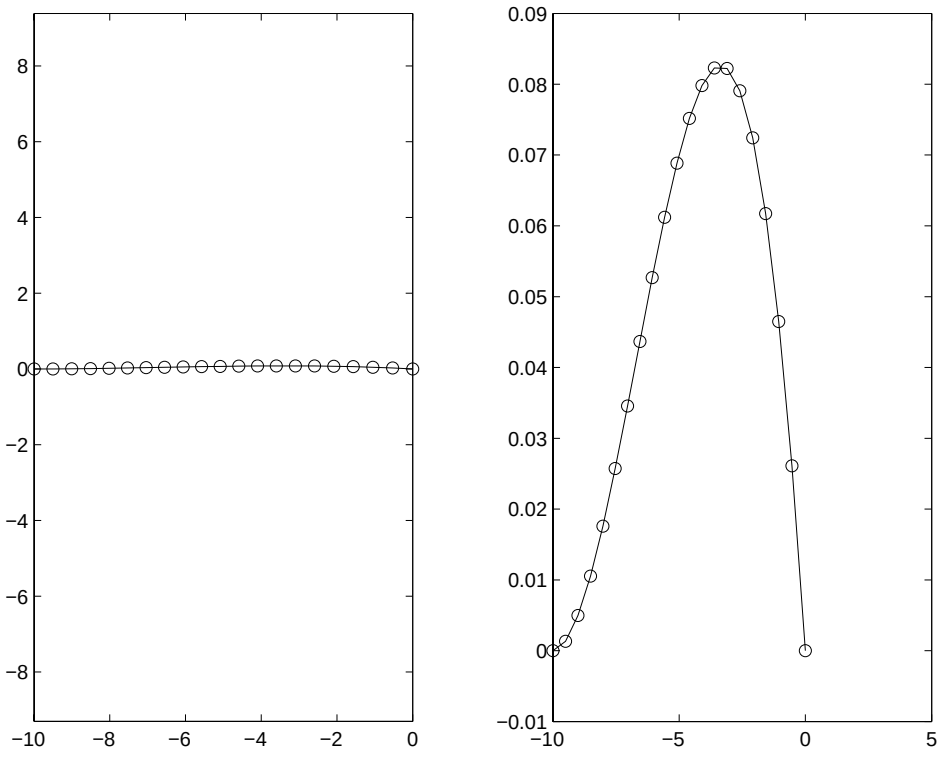

Figure 10 Suboptimal trajectory as the wind direction is $\pi/4$. The time to complete the route is 12.74 seconds

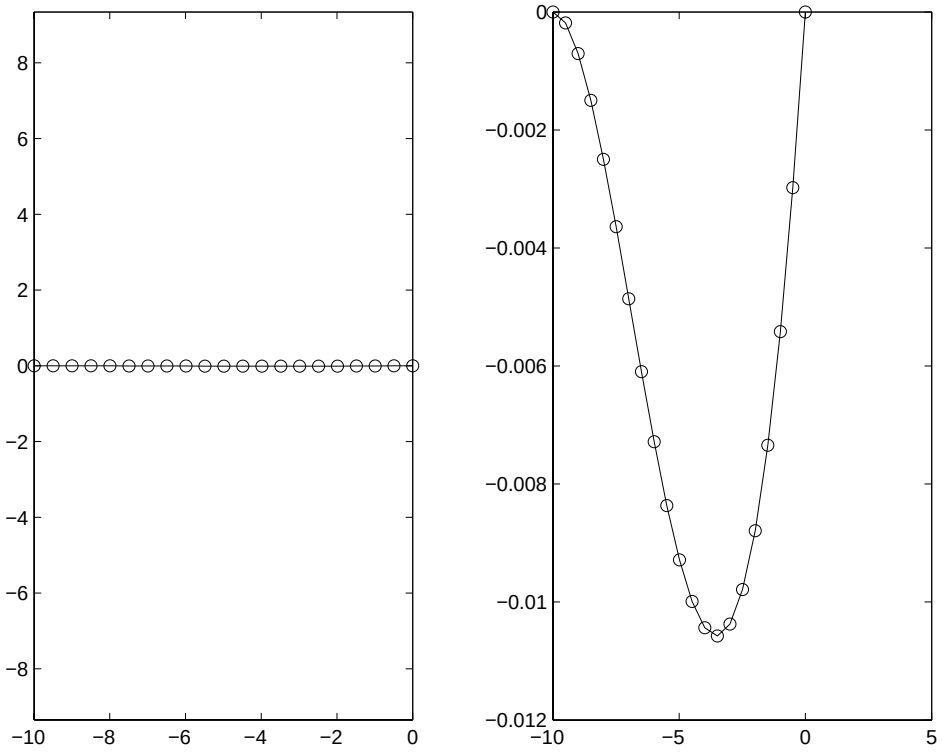

Figure 11 Less suboptimal trajectory. Wind direction is still $\pi/4$ and the time to complete the route is 12.73 seconds

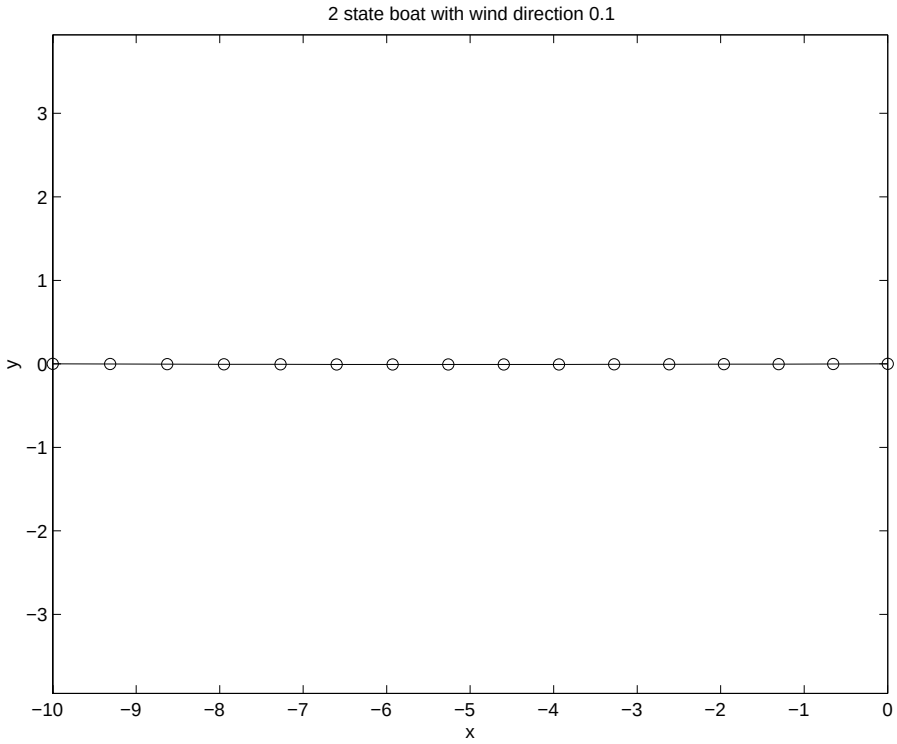

Figure 12 Non optimal trajectory found by RIOTS for a two-state-boat model

There could be some error in my calculations or the implementation of the gradient calculations in the boat dynamics. RIOTS test functions show very small differences between the results given by the above calculations and the gradient estimation by the finite differences. This suggests that the calculations are correct and that the error lies elsewhere.

Using the reduced order, 2 state model of the boat where the turning dynamics have been removed, yields no better results. Figure 12 shows an trajectory from RIOTS where θ_v is 0.1.

6. Static optimization of simple trajectories

6.1 Approach

We assume that the possible trajectories are going to be triangular, that is we change tack at most once. Research done with dynamic programming suggests that this is a reasonable assumption. See for example [3] for an example. We also assume that the boat can turn sufficiently fast in order to make the sharp turn and keep fairly close to the triangle.

Figure 13 shows how the course from A to B via C can be defined by two parameters, the angle α and the distance a. From these it is possible to calculate the angle β and the distance b, which gives the following expression for the course time:

$$T = \frac{a}{v(\alpha)} + \frac{b}{v(-\beta)} \tag{13}$$

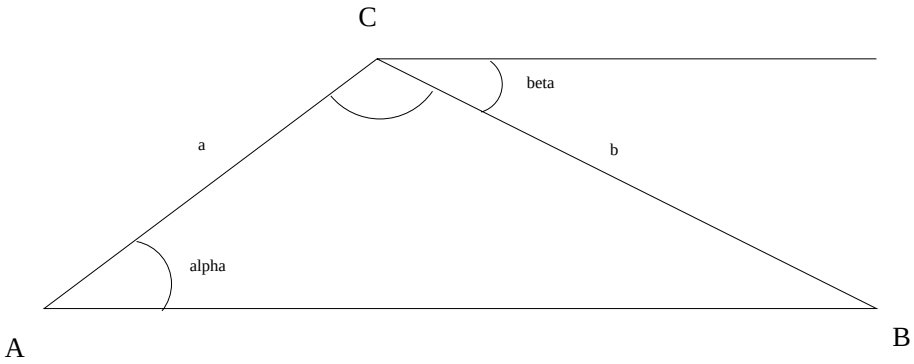

Figure 13 Simple parameterized trajectory

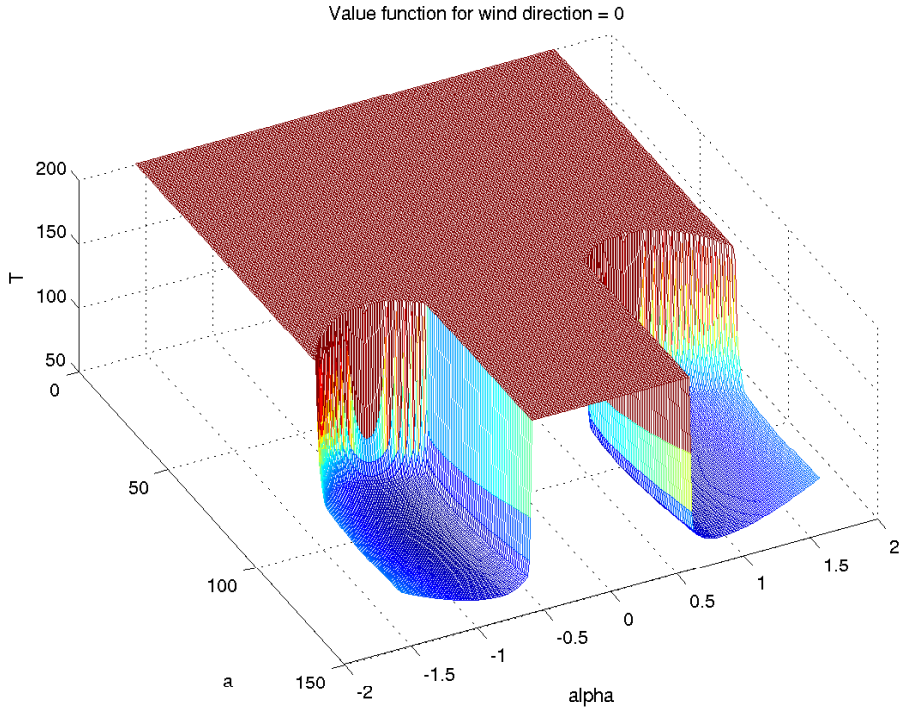

Figure 14 Value function for triangle trajectory when sailing towards the wind ($\theta_{wind} = 0$)

The expressions for b and β are

$$b = \sqrt{a^2 + c^2 - 2acos(\alpha)} \tag{14}$$

$$\beta = \arcsin(\frac{a\sin(\alpha)}{b}) \tag{15}$$

It is then a matter of static optimization. We minimize T with respect to a and α using for instance the Matlab function `fminunc`.

It is important to select the optimization starting point carefully when sailing upwind as the value function is undefined for many values of a and α, which can be seen in Figure 14.

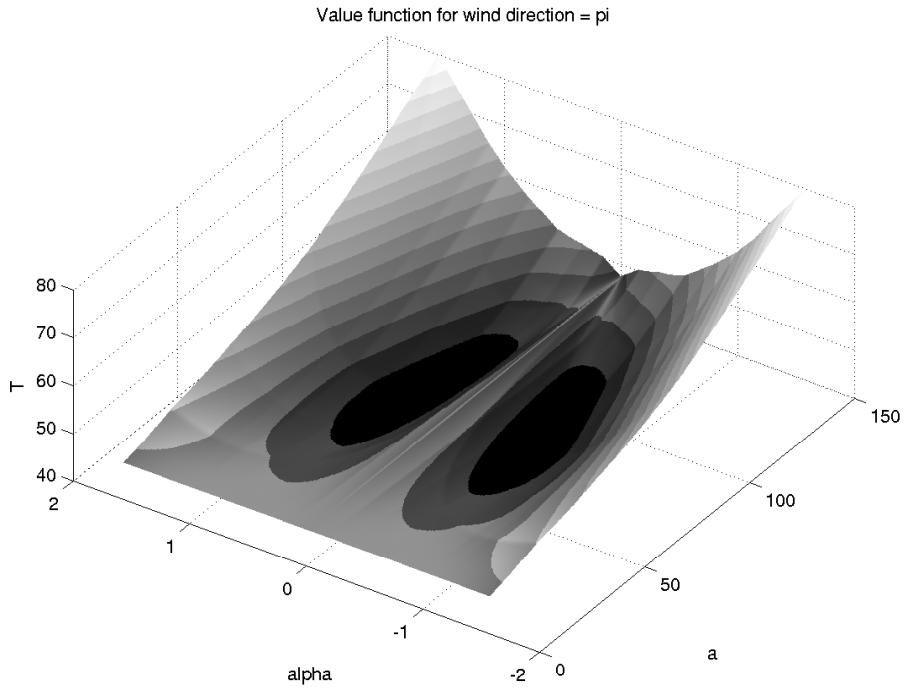

Figure 15 Value function for triangle trajectory when sailing downwind ($\theta_{wind} = \pi$)

Sailing downwind is much easier, as the value function allows for any choice of initial values except for $\alpha = \theta_{wind}$. Many times this will also work, but in the case of sailing with the wind exactly towards the target point, the optimization could stall. A plot of the value function for $\theta_{wind} = \pi$ is shown in Figure 15.

6.2 Results

A time optimal path can be calculated provided you only need to make one turn or less. Experiments with dynamic programming suggest this would be the case for all unconstrained trajectories if we disregard the disturbances.

Sailing upwind Using the method for an upwind situation with C 150 meters away you get the trajectory shown in Figure 16 with $a = 106, \alpha = -0.7854$. The time to complete the route is 60 seconds.

Sailing downwind An interesting result appears when sailing with the wind to the back. Figure 17 shows the optimal trajectory which is clearly not a straight line. The method suggests $a = 88, \alpha = 0.5424$ and the time to complete the route is 41 seconds, as compared to 47 seconds for the shortest route.

6.3 Conclusion and Remarks

This method has proven quite reliable, as long as the starting point is chosen reasonably. Although quick and robust, it is not suitable as a solution to the complete problem formulation (including obstacles) as as multiple turn

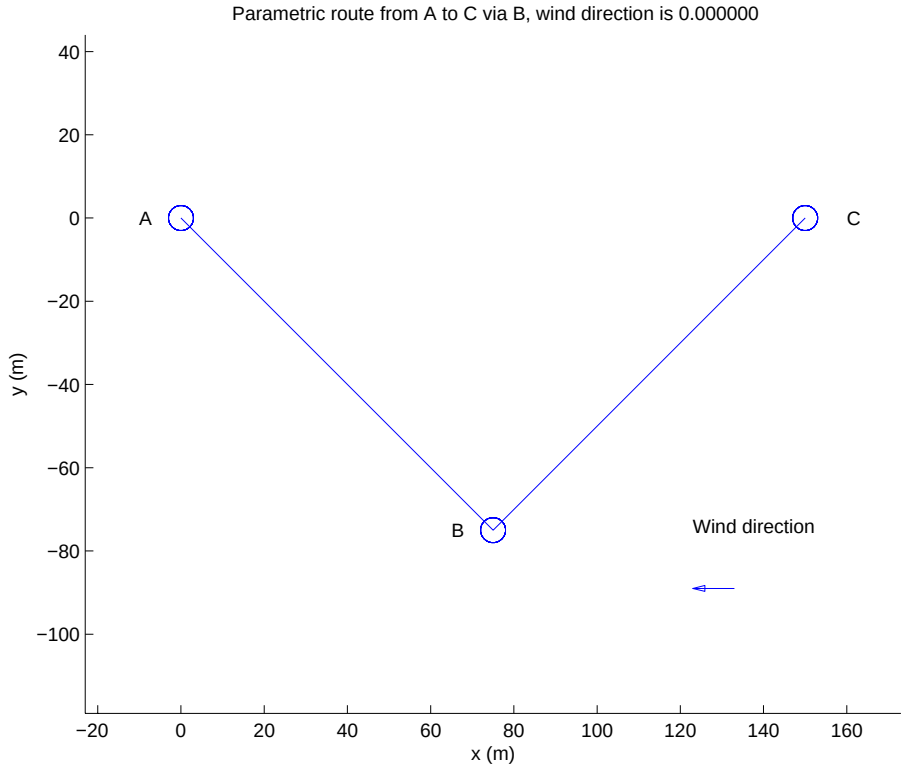

Figure 16 The optimized trajectory when sailing from A to C with $\theta_{wind} = 0$

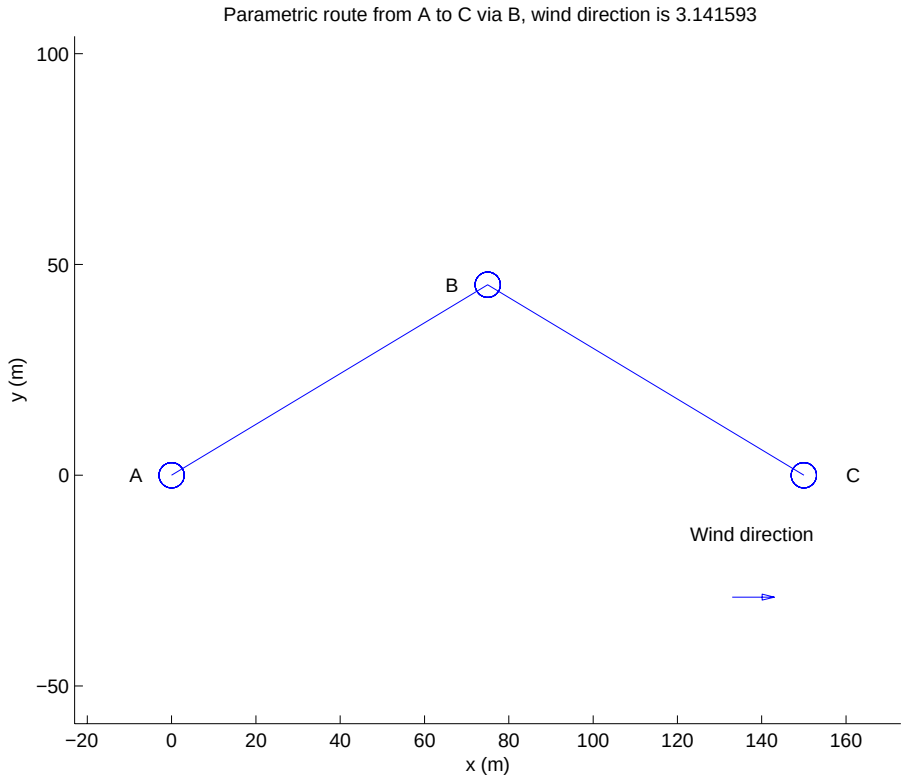

Figure 17 The optimized trajectory when sailing from A to C with $\theta_{wind} = \pi$

trajectories will yield very complex trigonometric expressions and quite possibly difficult optimization problems. Handling stochastic disturbances could also be a problem. It is however useful for quickly generating an initial guess for more numerically expensive calculations, such as dynamic programming and for long range and unconstrained sailing.

7. Analytical calculations

Optimal control theory suggests an analytical way to compute the optimal control signal by ways of dynamic optimization.

7.1 Approach

Minimize the cost function

$$J = \int_0^{t_f} dt \tag{16}$$

subject to

$$z = \begin{pmatrix} x & y & \theta \end{pmatrix}^T \tag{17}$$

$$\dot{z} = \begin{pmatrix} v(\theta)cos(\theta) \\ v(\theta)sin(\theta) \\ u \end{pmatrix} \tag{18}$$

$$z(0) = \begin{pmatrix} x_0 & y_0 & \theta_0 \end{pmatrix}^T \tag{19}$$

$$x(t_f) = 0 \tag{20}$$

$$y(t_f) = 0 \tag{21}$$

The Hamiltonian H then becomes

$$H = 1 + \lambda(t)^T \begin{pmatrix} v(\theta)cos(\theta) \\ v(\theta)sin(\theta) \\ u \end{pmatrix} \tag{22}$$

$\lambda(t)$ is defined as $(\lambda_1(t)\,\lambda_2(t)\,\lambda_3(t))^T$ which gives

$$H = 1 + \lambda_1(t)v(\theta)cos(\theta) + \lambda_2(t)v(\theta)sin(\theta) + \lambda_3(t)u \tag{23}$$

$$\dot{\lambda}(t)^T = -\frac{\partial H}{\partial z} \tag{24}$$

$$\dot{\lambda_1}(t) = \dot{\lambda_2}(t) = 0 \tag{25}$$

$$\dot{\lambda_3}(t) = -\lambda_1(\frac{dv}{d\theta}\cos(\theta) - v(\theta)\sin(\theta)) - \lambda_2(\frac{dv}{d\theta}\sin(\theta) + v(\theta)\cos(\theta)) \tag{26}$$

$$\theta(t) = \theta_0 + \int_0^t u(\tau)d\tau \tag{27}$$

This primitive is very hard to find and currently beyond this thesis.

8. The CDP Toolbox

CDP[3] is a Matlab toolbox developed by Sven Hedlund at the Department for Automatic Control, Lund Institute of Technology for solving optimal control problems for hybrid systems. The underlying theory is based on converting the non-linear problem into a larger linear programming form. It uses a discretized state space and can be configured to use a number of different linear programming solvers. In this thesis I have used the built-in Matlab solver `linprog`. More on the theory CDP is based on can be found in [1].

As CDP uses a numerical discretization of the state space, memory constraints limits precision when using lots of states. Using many states also increase the computational complexity. For these reasons, the number of states has been reduced to two, discarding the turning dynamics. Initially, we also disregard the disturbances.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} v(u)\cos(u) \\ v(u)\sin(u) \end{pmatrix} \tag{28}$$

Using this model, `cdplowm` was run with the following arguments:

```
uv = [0:pi/4:2*pi]';
O = [-20 -20 20 20 ]';
xmin = [-10 -10]';
xmax = [10 10]';
XQ = [xmin  ; xmax ; 1 ];
xqf = [0 0 1 ]';
N = [11 11]';
tb = 0;
```

Subsequently, `cdpctrl` is used to calculate a control matrix determining the optimal control signals for different points in the state space.

8.1 Results

I have not yet been able produced a usable control law with CDP given the current problem formulation.

8.2 Remarks

The problems using CDP may well be associated with memory problems. I have tried solving basic problems with small linear systems (one or two states) and have found that a too small grid can cause the optimization to fail or yield obviously wrong results. Since increasing the number of grid points to more than 11x11 required more memory than the available computer was equipped with (384 Mb + 0.5 Gb swap), I decided this approach was not suited for autopilot implementation.

It should be noted that the massive memory usage could be the result of using `linprog`. Unfortunately, I haven't had access to other solvers (e.g. PCx or similar) so that possibility remains unchecked.

[3] Short for Convex Dynamic Programming

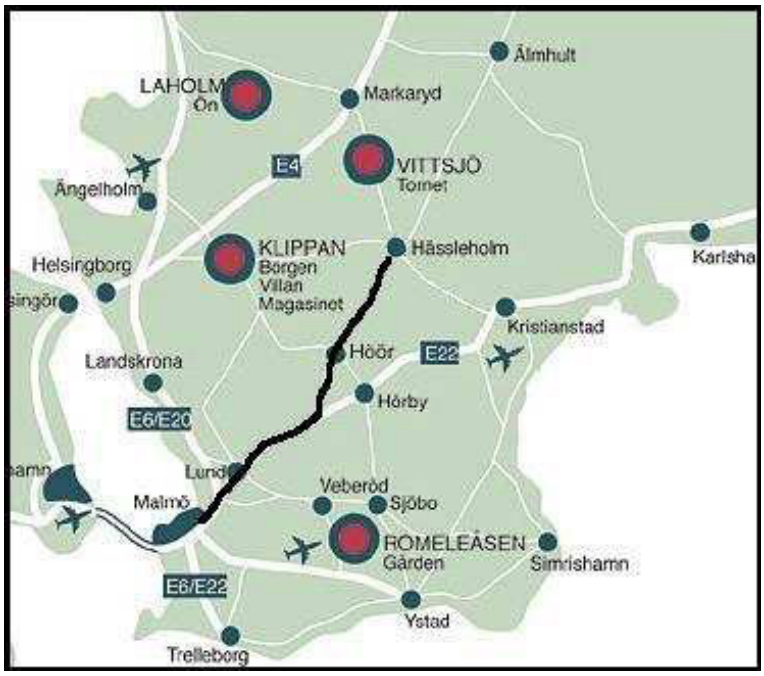

Figure 18 The black line represents the shortest route between Hässleholm and Malmö.

9. Dynamic Programming

Dynamic programming is a principle for solving dynamic optimization problems by ways of using partial results. It has many close relatives in graph algorithms (such as Dijkstra's algorithm) and can be applied to a large class of problems, ranging from optimal control to game theory and production planning.

9.1 Brief Introduction to Dynamic Programming

The fundamental principle is very simple in itself and best illustrated by an example. Imagine three cities, let's call them Malmö, Lund and Hässleholm. We place ourselves in Hässleholm and in some way we manage to calculate the time optimal way of driving from Hässleholm to Malmö. By chance, this happen to take us through Lund. If later we find ourselves in Lund and would want to go to Malmö as fast as possible, the dynamic programming principle would tell us that this would take us along the last part of the optimal route from Hässleholm to Malmö. The route is shown in Figure 18.

Let's expand the example by adding a few other cities, say Landskrona, Trelleborg and Sjöbo. For some reason, you already know the optimal way of getting from all these three places to Malmö. If you then happen to find yourself somewhere in Skåne (the area of Sweden shown in Figure 18) without knowing how to get to Malmö as fast as possible, you can perhaps figure out which one of these known cities is easiest to reach. Rephrased in another way, you need to figure out which one of these cities might lie on the optimal route to Malmö and then use the partial result of how to getting from that city and to your final destination.

Mathematically speaking, imagine an optimal discrete time trajectory

$z_0, z_1 \dots z_f$ given the transition cost function $g(z_1, z_2)$ and a set of admissible control signals U. We can now define the cost-to-go function $V(z)$ that gives the remaining cost of moving along the optimal trajectory from z to z_f as

$$V(z_k) = \begin{cases} 0 & if\, z_k = z_f \\ \min\{g(z_k, z_{k+1}) + V(z_{k+1})\} & otherwise \end{cases} \tag{29}$$

The method of solving for the optimal trajectory from any given state will be to start out in z_f and then recursively go backwards until you reach the desired initial state. Observe that since we are in discrete time (with typically uniformly sampled systems) we cannot know the number of steps needed for a specific initial state if the object is to achieve a minimal time trajectory.

9.2 Approach

The number of recursions in each case will be limited by the number of admissible control signals (i.e. the number of elements in U) but will soon grow very large unless restricted. In a continuous state space, there is an infinite number of possible states so our first step will be to sample each state uniformly. We can now also introduce a memory so that the algorithm is able to recognize a state is has visited before.

On its current recursive form, the algorithm will still require a large number of recursions for any normal sized problem. Assuming we sample each second and that we wish to travel 100 m along a straight line in the direction where the boat is fastest, we could still need some 30 decisions for a typical situation. Using a conservatively discretized control signal with 3 admissible choices (max turn left, max turn right and straight ahead) we would then end up with $3^{30} \approx 10^{14}$ recursions which obviously is too much.

Approximate Iterative Algorithm Using that the dynamic programming algorithm relies on partial results we can reform the recursive algorithm in an iterative way. We now introduce the following

$$X = (X_0, X_1 \dots X_{m-1}), \left|X_i - X_j\right| = \Delta x | i \neq j, i, j \in [0, m-1] \tag{30}$$
$$Y = (Y_0, Y_1 \dots Y_{n-1}), \left|Y_i - Y_j\right| = \Delta y | i \neq j, i, j \in [0, n-1] \tag{31}$$
$$\Theta = (\Theta_0, \Theta_1 \dots \Theta_{k-1}), \left|\Theta_i - \Theta_j\right| = \Delta \theta | i \neq j, i, j \in [0, k-1] \tag{32}$$

as a sampled region of the continuous state space. We now compute an estimation $\hat{V}(z)$ of $V(z)$ iteratively. Let $\hat{V}_k(z)$ be the estimate of $V(z)$ after k iterations. For compact notation, we will write the system dynamics as $z_{t+1} = \Phi(z_t, u)$ and the state space region defined by X, Y, Θ as S.

$$\hat{V}_k(z) = \begin{cases} 0 & k = 0, z = z_f \\ \infty & k = 0, z \neq z_f, z \in S \\ \infty & z \notin S \\ \min_{u \in U} \left\{ g(z, \Phi(z,u)) + \hat{V}_{k-1}(\Phi(z,u)) \right\} & otherwise \end{cases} \tag{33}$$

A problem with this formulation is that it is unlikely that you will move exactly the distance between two points in S during one discrete time step. This can be solved in two ways.

Variable Step Time If we allow each step to take a varying amount of time, we could calculate $\min\left\{g(z,\Phi(z,u))+\hat{V}_{k-1}(\Phi(z,u))\right\}$ by selecting a number of states close to z (e.g., within a given radius) and calculating the cost to reach them, then selecting the choice with the least cost. Let $z_{new}=(x_{new}\ y_{new}\ \theta_{new})^T$ denote one such state. Then $g(z,z_{new})$ could be expressed as

$$g(z,z_{new})=\begin{cases}\infty & z_{new}\ \text{unreachablefrom}z\\ \frac{\sqrt{(x-x_{new})^2+(y-y_{new})^2}}{v(\theta_{new})} & otherwise\end{cases} \tag{34}$$

The problem here would be to figure out if z_{new} is unreachable. Also one must take care to allow for a sufficiently large radius when choosing candidate states as a too small circle would explore too few directions and possibly miss a local maxima on the speed curve. For this to work, one must also assume that any control signal u that is within the span of U would be allowed.

This approach also makes it difficult to handle disturbances, as you would have to figure out how to counter the disturbance in order to get to the state space point in the end.

Linear Approximation in between Points A less complicated way is to allow z_{new} to be a point in between the grid points of S. $\hat{V}_{k-1}(z_{new})$ could then be determined as a linear interpolation of $\hat{V}_{k-1}(z_{new})$ for the points in S around z_{new}.

The method to calculate $g(z,z_{new})+\hat{V}_{k-1}(z_{new})$ here would be to try all possible control signals in U and selecting the one that would give the least cost. It is important to note that in order for this to work, the function $V(z)$ must be sufficiently smooth for the linear interpolation to be exact enough. A formal proof of this is beyond this thesis.

Since we are using linear interpolation, allowing $\hat{V}_k(z)$ to approach ∞ is not practical and instead we choose a large value, preferably some orders of magnitude larger than the optimal time.

This method allows for easy handling of the water current disturbances, as they (if they are known or estimated) can just be incorporated into the system dynamics function $\Phi(z,u)$.

Convergence Given a sufficiently smooth $V(z)$, the estimation converges for any initial guess. A better initial guess should improve convergence speed. This is an important property as it allows the algorithm to adapt to changing system dynamics faster. For this application, measurement of difference between two iterations will be based on

$$\Delta V=\sum_{z\in S}\left|\hat{V}_k(z)-\hat{V}_{k-1}(z)\right| \tag{35}$$

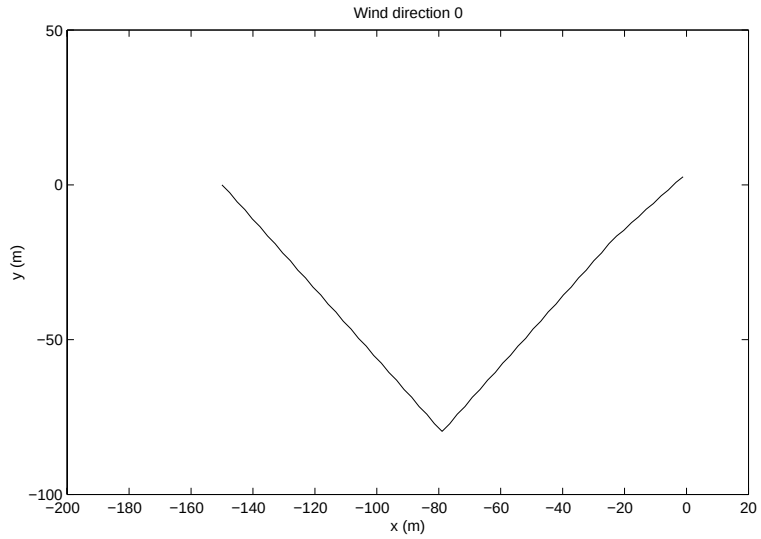

Figure 19 Sailing against the wind from (-150,0) to the origin. Boat starts out heading towards the wind.

Handling Stationary Obstacles Stationary obstacles can be handled adding special cases to the cost-to-go function similar to how one handles states outside of S. All points within the obstacle will have a high constant cost which will make it "expensive" to maneuver through them. In this thesis, we will limit the experiment to a single, rectangular obstacle defined by the four coordinates $x_{min}, x_{max}, y_{min}, y_{max}$.

Navigating by the Cost-to-go Function After having obtained an estimate for $V(z)$ we can use it for feedback control. By doing essentially a one step iteration from the current state, we select the control signal that drives us in the direction where the cost-to-go function decreases most rapidly, in essence a steepest decent method.

It is important to note that the cost-to-go function is calculated in a batch operation. As long as you navigate inside S and with constant wind and current conditions, there will be little calculation left to do.

9.3 Results

The above described algorithm was implemented using ANSI C on a Linux PC.

We use a state space discretization with 101 x 101 x 17 points and 8 control signals. X, Y stretches between -200 and 200 while U spans between $-\pi/4$ to $\pi/4$. The boat reaches the point (-1.113476, 2.612883) after 62 seconds. Calculating the cube used for the simulation took about 1440 seconds (75 iterations) on a Pentium 3, 600 MHz, running a common desktop Linux OS. Running the actual simulation takes less than 0.01 seconds.

Switching to sailing downwind reveals some interesting results.

Sailing this route takes 41 seconds, as compared to 47 when sailing the shortest route. This shows that some boats will benefit from not always taking the shortest route between two points. The gain is quite substantial (a 12% reduction in sailing time) but for these short distances it might be more practical to take the slower but simpler route as turning introduces

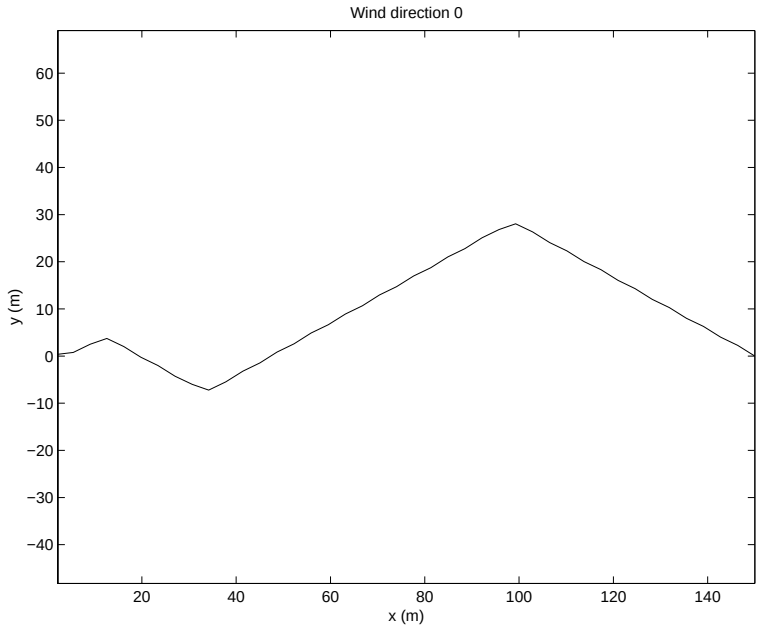

Figure 20 Sailing downwind from (150,0) to the origin. Boat starts out heading away from the wind.

a lot of manual work. Over longer distances the drawbacks of having to turn will decrease.

Adapting to Changing Wind Changing the wind direction to 0.3 radians and then allowing the algorithm to recalculate until it converges again takes 1260 seconds (or 64 iterations), which suggests there is not a lot to gain by using the adaptive properties in this way. It is possible that the time it takes for the estimation to converge is more dependent on the number of grid points and the speed curve than the initial guess.

Handling obstacles Adding obstacle handling to the algorithm does not seem to change the convergence rate significantly. The object in the experiment was to maneuver around an object between the starting position at (50, 0) and the origin, see Figure 20. The object is rectangular and stretches from (20, -20) to (40, 20). As before, the wind direction will be 0. We here use a substantially smaller state space grid, mostly because it gives a cost-to-go function that is easier to display.

Water Currents We now add a steady water current with $d_x = d_y =$ -1 m/s. This is a quite strong current, but not uncommon in some waters. We use the same obstacle as before and the same wind direction.

If we take a look at the cost-to-go function for two situations where the boat starts out close to the obstacle on the right side we can see in Figure 24 that the cost quickly rises to infinity when starting out facing towards the obstacle while not so when starting out facing away from it (Figure 25). This is a sign that the boat will not be able to avoid the obstacle from that starting situation.

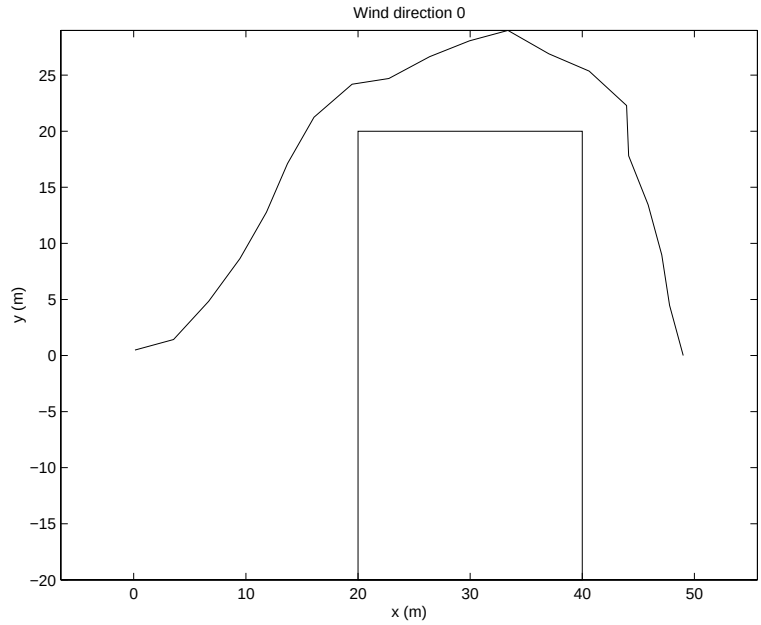

Figure 21 Sailing around an obstacle, starting in (-50, 0) with the boat facing $\pi/2$.

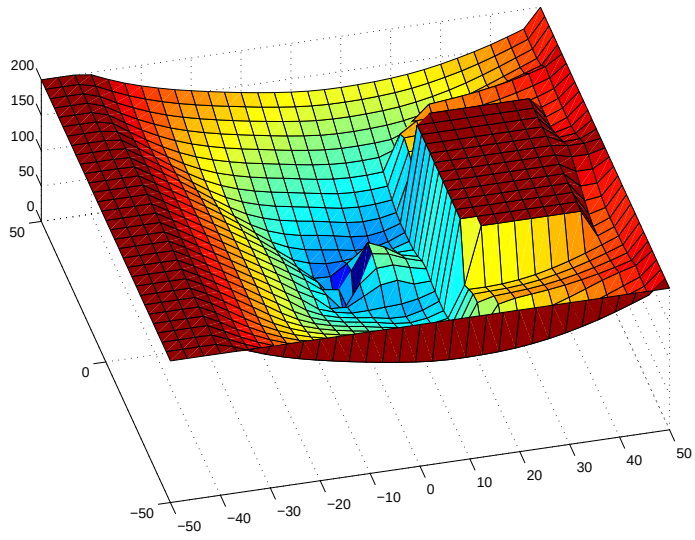

Figure 22 A slice of the corresponding cost-to-go function where the boat starts out at the angle π. The obstacle is visible in the form of a very high cost region of the cost function

9.4 Conclusion and Remarks

The algorithm shows much promise as it can handle all the difficulties except the stochastic wave disturbances. Calculating the cost-to-go function can be done in advance and since the resulting data require fairly little storage space, it can be pre-calculated for a large number of possible wind and current conditions.

Since all the individual calculations in an iteration only depend on the

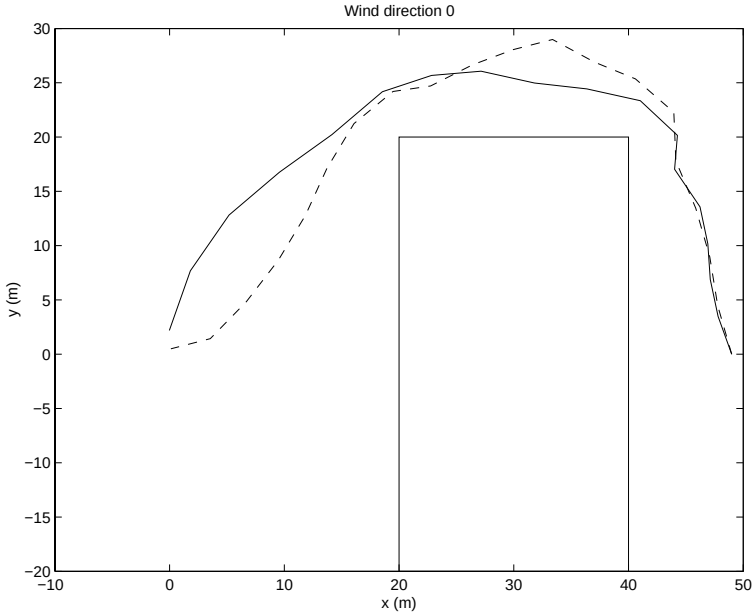

Figure 23 The current $d_x = d_y = -1$ is used here. Also plotted with a dashed line is the previous trajectory when we had no current.

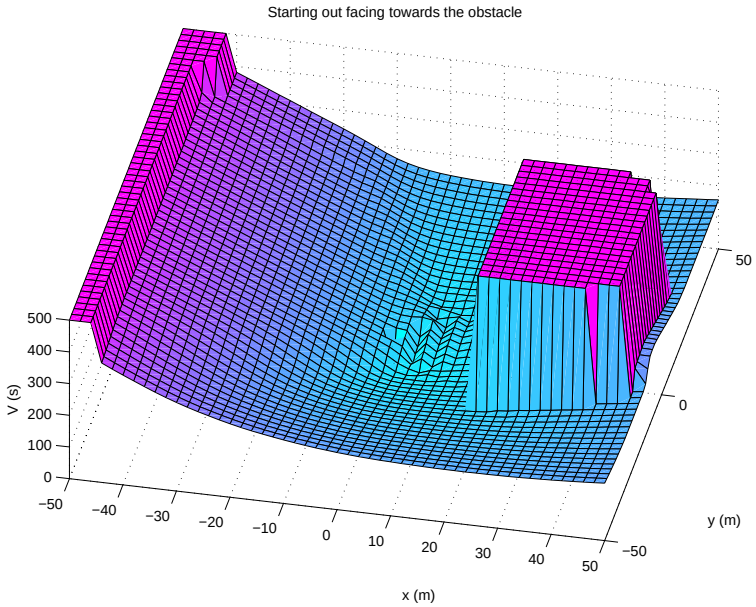

Figure 24 Cost-to-go function when starting out in (48,0) facing towards the obstacle. Wind direction is 0 and currents along (-1,-1)

previous and not on each other, the algorithm is well suited for parallelization if several CPUs are available. Theoretically it would scale approximately linearly until the number of threads approach the number of discrete state space points, which seems unlikely for a ship board mounted solution (see Chapter 10).

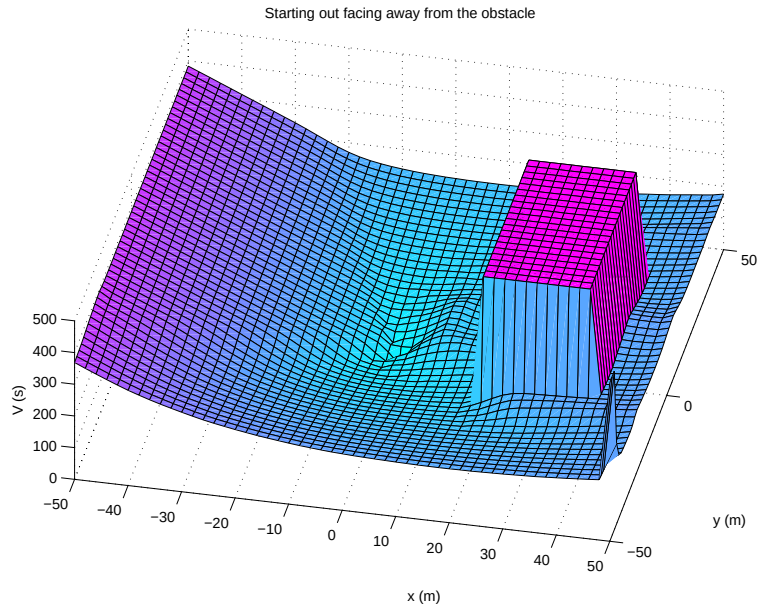

Figure 25 Cost-to-go function when starting out in (48,0) facing away from the obstacle. Wind direction is 0 and currents along (-1,-1)

10. Autopilot Design

There are several issues to address before the algorithm developed in Chapter 9 can be used on a sailing boat. This chapter will not provide a finished design for an autopilot, but will try to address a few of the problems involved with such an attempt.

10.1 Hardware Considerations

Equipment that should be taken on board must be resilient to force and water. It must not consume too much power as the batteries of the boat must power such things as the GPS and the ignition for the on board engine. At the same time, you need sufficient computing power to re-calculate the cost-to-go function in reasonable time should the need arise.

The dynamic programming algorithm is dependent on position and heading information which are both easily obtained from the GPS unit commonly found on board.

A plausible solution to these requirements would be to use a laptop computer connected to the GPS unit. If present, it could also be connected to the rudder servo. A laptop bought today would be much faster than the computer used for the experiments presented in this report but would probably still need many minutes to complete a high resolution optimization such as the first example in Chapter 9.3. There are several options for improving performance.

Software Optimization The program as it is written today is completely unoptimized. It is still many times faster than the Matlab implementation produced to prototype the algorithm, but some optimizations are evident. Running gprof[4] gives the following output shown in Table 1.

[4]A profiler that comes with gcc, the c-compiler used for this thesis. See the online manual

Table 1 The output from running gprof on the algorithm

```
Flat profile:

Each sample counts as 0.01 seconds.
   %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 17.65     12.33    12.33 88363392     0.00     0.00  vfunk3
 14.71     22.61    10.28 11045424     0.00     0.00  cube_interp...
 10.96     30.27     7.66 682657041    0.00     0.00  dynassert
 10.76     37.79     7.52 99944832     0.00     0.00  cyclemod
  9.76     44.61     6.82      104    65.58   671.35  dynitteration3
  9.53     51.27     6.66 24610560     0.00     0.00  linterpol
  8.92     57.50     6.23 24610560     0.00     0.00  findindex
  4.67     60.76     3.26 12305280     0.00     0.00  findcube
  4.02     63.57     2.81 12305280     0.00     0.00  findcyclicindex
  3.33     65.90     2.33 12305280     0.00     0.00  step3
  2.93     67.95     2.05 24610560     0.00     0.00  speed
  0.70     68.44     0.49 24610560     0.00     0.00  fcyclemod
  0.62     68.87     0.43 24610560     0.00     0.00  fmin
  0.53     69.24     0.37 14004328     0.00     0.00  in_obstacle3
  0.47     69.57     0.33 12215944     0.00     0.00  xyoutside_cube
  0.36     69.82     0.25  1538160     0.00     0.00  vectormin
  0.07     69.87     0.05      905     0.06     0.06  MatrixWrite
  0.00     69.87     0.00      105     0.00     0.00  set_xycost
  0.00     69.87     0.00       53     0.00     0.94  CubeWrite
  0.00     69.87     0.00        4     0.00     0.00  makedvector
  0.00     69.87     0.00        2     0.00     0.00  MatrixRead
  0.00     69.87     0.00        2     0.00     0.00  makedcube
  0.00     69.87     0.00        1     0.00     0.00  save_cube
  0.00     69.87     0.00        1     0.00     0.00  set_defaultcost
```

This means we can reduce the computation time by 10% by removing the `dynassert` calls (only meaningful during development). `vfunk3` is fairly unoptimized as it is implemented and could perhaps be speeded up by choosing a smarter data structure for handling the cube that holds the values of $\hat{V}_k(z)$. A speed gain could possibly result from better use of the CPU cache. `cyclemod` is a function created to handle the circular nature of the θ state and could possibly be optimized with in-line assembler. All in all, a speed gain of more than 20-30% is unlikely and large optimizations would still be problematic to handle in real time.

Algorithm Optimization Given knowledge of how the algorithm works, there are several possibilities for improving the performance.

- Given how the approximation "grows" from the origin a possible optimization would be to just do the calculations on the area to which the approximation has spread. Since the growth speed is constant this would reduce the execution time by roughly 50%.
- If the wind and currents are constant for all state space points, the resulting relative step would be the same for all (x, y). It can then

pages for gprof for a detailed description

be shown that the resulting cost for a step in a given direction can be computed for all states at the same time using a 3-dimensional convolution. Present day hardware has good functions for computing convolutions rapidly which could be exploited in this case.

Specialized Hardware It was mentioned earlier that the algorithm is well suited for parallelization. Few laptops are fitted with multiple CPUs so custom hardware would have to be developed. This option clearly offers the largest performance improvement, but also the one that would be the most expensive. CPUs are cheap but require a motherboard to work. Motherboards for more than 2 CPUs are typically expensive and probably not an option for the amateur this thesis is aimed at. If implemented, the algorithm would scale linearly for each CPU, a property that comes from that each calculation only depends on already finished results.

10.2 Following a Course

Having computed an optimal route we are immediately faced with the problem of following it. The wave disturbances are routinely handled by the sailor but practical experience has shown that autopilots have trouble keeping the boat on course as they can't see the waves and as such won't discover the disturbance until it has already caused an error.

Kalman filter Estimation of Wave Disturbances We modeled the turning dynamics together with the noise caused by the waves as an ARMAX-model

$$\theta(k+1) = \theta(k) + u(k) + C(z^{-1})e(k) \tag{36}$$

The polynomial $C(z^{-1})$ would have to be obtained from studying the wave dynamics. In Figure 26 we see wave data collected from an oil platform in the North Sea. The majority of the frequency content is concentrated between 0.1 and 0.4 Hz (shown in Figure 27), implying that if we continue to sample the process at 1 Hz, the FIR-filter would need to be at least of length 10, possibly much higher to obtain sufficient dampening.

The sample rate cannot be made much slower since me must keep above the Nyquist frequency. Instead, I suggest a state space representation. Introduce the state vector

$$v = \begin{pmatrix} v_1 & v_2 & \dots v_n \end{pmatrix}^T \tag{37}$$

and the two systems

$$\theta(k+1) = \theta(k) + u(k) + w(k) \tag{38}$$

$$v(k+1) = \Phi_v v(k) + \Gamma_v e(k) \tag{39}$$

$$w(k) = C_v v(k) \tag{40}$$

This can be written in block form like

$$\begin{pmatrix} \theta(k+1) \\ v(k+1) \end{pmatrix} = \begin{pmatrix} 1 & C_v \\ 0 & \Phi_v \end{pmatrix} \begin{pmatrix} \theta(k) \\ v(k) \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & \Gamma_v \end{pmatrix} \begin{pmatrix} u(k) \\ e(k) \end{pmatrix} \tag{41}$$

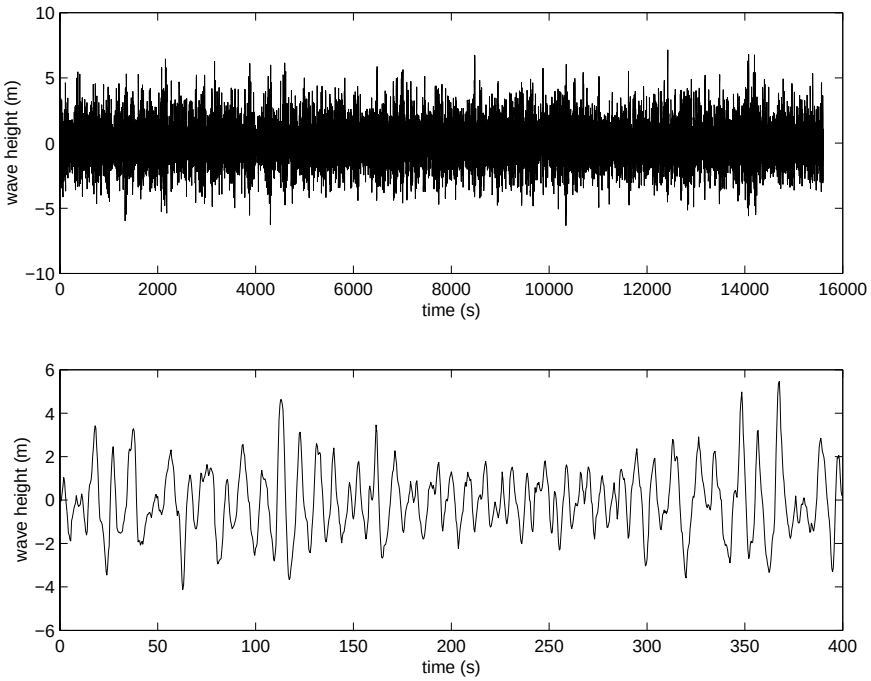

Figure 26 The wave data was measured on December 24th 1989 at the Gullfaks C platform in the North Sea from 17.00 to 21.20. The data was graciously provided by Jesper Ryden from the Department of Mathematical Statistics at Lund Institute of Technology

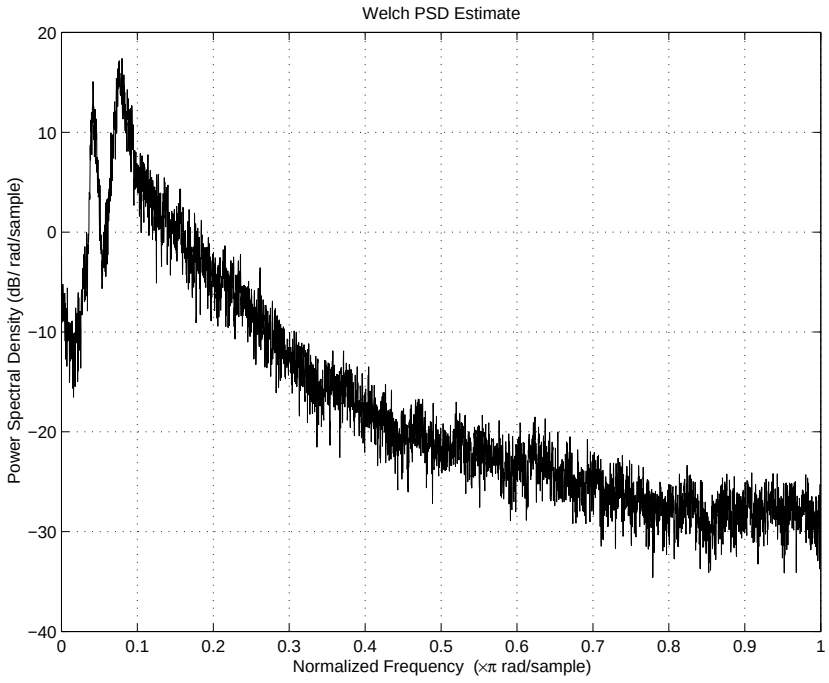

Figure 27 A power spectrum estimation using Welch's method

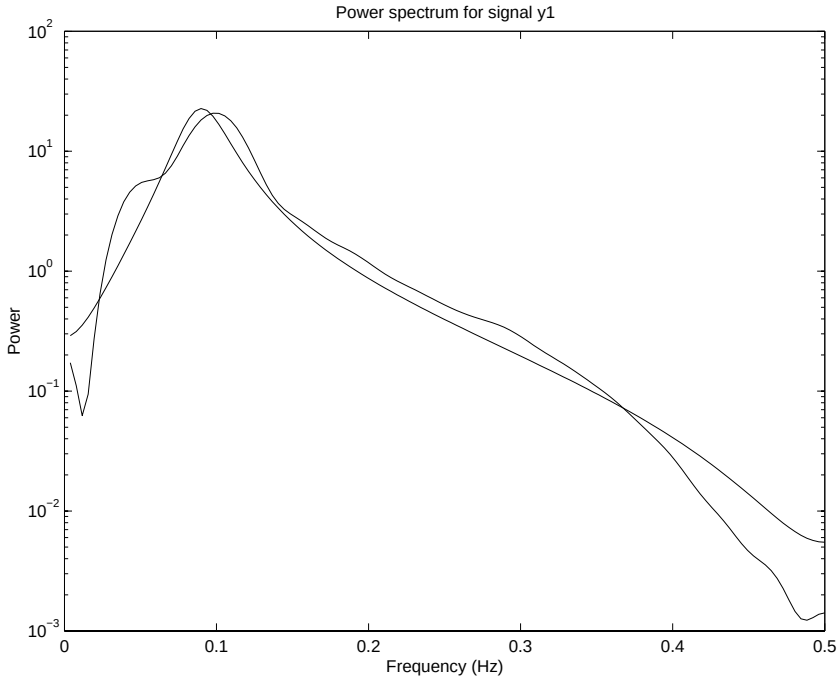

Figure 28 Comparing the estimated model with the data. The two spectrums are reasonably alike with the smoother line representing the model spectrum. The estimated spectrum from the wave data has been estimated after downsampling to 1 Hz

$$y_\theta(k) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} \theta(k) \\ v(k) \end{pmatrix} \tag{42}$$

Now we can use system identification methods to obtain the matrices Φ_v, Γ_v and C_v. Applying first `idresamp` to down sample the wave data to 1 Hz and then using `n4sid` to identify a 2nd order model.

The one thing that remains is determining how the surface elevation affects the boat heading. It seems reasonable that it is the slope of the wave, not the actual hight, that will cause the disturbance. Exactly how much deviation this slope then causes is dependent on the hull (size and shape) but also from which direction the wave hits the hull. Intuitively, one would guess that a wave hitting the boat in the stern wouldn't affect the heading while one hitting it from the side would.

We introduce a differentiating filter and a scaling factor k_w

$$
\begin{aligned}
\begin{pmatrix} \theta(k+1) \\ v(k) \\ v(k+1) \end{pmatrix} &= \begin{pmatrix} 1 & -1 & k_w C_v \\ 0 & 0 & k_w C_v \\ 0 & 0 & \Phi_v \end{pmatrix} \begin{pmatrix} \theta(k) \\ v(k-1) \\ v(k) \end{pmatrix} + \\
&\quad \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & \Gamma_v \end{pmatrix} \begin{pmatrix} u(k) \\ e(k) \end{pmatrix} \qquad (43) \\
y_\theta(k) &= \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \theta(k) \\ v(k-1) \\ v(k) \end{pmatrix}
\end{aligned}
$$

From here it is a straight forward task to compute a minimum-variance controller by using e.g. the `dkalman`, `dlgr` and `lqgreg`. As validating the results from such a construction requires a "true" model of the wave to θ dynamics or an actual boat, this will not be covered in this thesis. An explanation of the calculations required to obtain the estimator can be found in [6].

10.3 Estimating Wind and Currents

The GPS unit makes it relatively simple to estimate the wind and currents. The boat is typically fitted with both some kind of wind indicator and a speed meter. The wind indicator will tell the wind direction relative to the boat and a recent model might even tell the wind speed. In order to get a usable estimate, the indicator signal must be low-pass filtered and compensated for the velocity of the boat (given by the GPS). Similarly, the speed meter will tell the boat speed relative to the water in the direction the boat is heading. Subtracting this from the absolute speed (given by the GPS) will yield a local estimate of the water currents.

11. Conclusions

We have seen that it is possible to compute routes around obstacles taking into account the specific wind properties of the boat as well as currents. This has been done on hardware sufficiently cheap and low on power consumption to be used on board an ordinary sailing boat. Stochastic wave disturbances has not been solved, but a strategy that seem likely to work has been prototyped.

Although able to handle many different problems, it is important to note that the mathematical properties of the linear approximation based version has not been examined. The fact that it gives the same solution as the static optimization approach from Chapter 6 for unconstrained problems gives some indication of its validity.

12. Further work

An important step in continuing to investigate the iterative dynamic algorithm using the linear interpolation approximation would be to construct a valid proof of its correctness.

The algorithm can easily be expanded to handle any number of states, control signals and obstacles, making it useful for many other optimal control problems. Handling such things as generic trajectory constraints can be introduced by explicit manipulation of the value function as well as by introducing checks while deciding on the best control signal similar to those used now for obstacle avoidance.

Additionally, introducing stochastic properties into the dynamics would also be desirable and would possibly allow for inclusion of wave disturbance handing into the main algorithm.

13. References

[1] Sven Hedlund. *Computational Methods for Optimal Control of Hybrid Systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, May 2003.

[2] J C Kimball and Harold Story. Fermat's principle, Huygens' principle, Hamilton's optics and sailing strategy. 1997.

[3] Andy Philpott and Andrew Mason. Optimizing yacht routes under uncertainty. 2000.

[4] A. Schwartz and E. Polak. *RIOTS Manual*. Department of Electrical Engineering and Computer Sciences, University of California, 1996.

[5] Gunnar Sparr and Annika Sparr. *Kontinuerliga system*. Studentlitteratur, 2000.

[6] Karl Johan Åström and Björn Wittenmark. *Computer Controlled Systems*. Prentice Hall, 1997.

A. Source Code and Downloads

Refer to the webpage

`http://www.control.lth.se/articles/article.pike?artkey=5717`

for an electronic copy of the latest version of this thesis as well as a downloadable copy of the source code used for producing the results. Note that this software is provided "as is" and is not supported by neither the author nor the Department of Automatic Control.