Michael Levy
Prit Chovatiya

Gain Scheduling and Short-Term Path Optimization for Autonomous Sailboats

Based on:
"A gain-scheduling control strategy and short-term path optimization with genetic algorithm for autonomous navigation of a sailboat robot"
By Davi Henrique dos Santos and Luiz Marcos Garcia Goncalves

Importance of the Problem

With oceans composing 71% of our planet, it is imperative society understands how they evolve over time. One method of collecting data on our oceans is the use of unmanned surface vehicles (USVs). Deploying a large fleet of USVs will decentralize measurement of weather changes, tidal levels, animal migration, ocean acidification, and oil spills. All of which are incredibly important to improve the fidelity of evolving climate change models. Large scale deployment of USV fleets will enable remote data relay, support expanded ocean charting, and monitor ocean traffic. Demand for sustainable technologies is readily increasing, and USVs offer noninvasive means of measuring environmental data and climate change indicators.

In order to scale a fleet of USVs sustainably, full autonomy and high mission-endurance must be achieved. USVs must leverage the ocean's abundant supply of wind energy to attain multi-year mission endurance without human intervention. Sailboats are the most suitable candidate for sustainable rollout of USVs given their ability to harness wind energy. A sailboat USV has the difficult challenge of being able to navigate a very non-linear environment: the ocean! Being able to build an effective, robust, high, and low-level controller for the sailboat has proven to be a non-trivial task.

Background

Traditionally, low-level control of an autonomous sailboat has utilized static parameters for a proportional-integral (PI) controller for the rudder and sail angle. This controller can easily be implemented, however, is not optimal for the varied weather conditions the sailboat can find itself in. For the high-level path planning controller previous optimization methods found reachable tacking waypoints for a sailboat to follow to get to a target. However, no previous method considered waypoint optimization with respect to time or resource constraints on the sailboat (such as energy or lateral space). When an autonomous sailboat is trying to sail upwind specifically, the path planning controller has traditionally not given an optimal solution and often humans would need to interact with the system [1].

Sailboats have a specific zone when sailing "upwind" or as some say "beating" that they cannot harness power from. This area where the boat cannot harness power is called the "no go zone." Generally, a sailboat can point as close to 45 degrees from the true wind direction. This angle

varies from boat-to-boat and is often characterized by a specific boat's "Polar Diagram" (shown in the image below). A polar diagram shows the angle at which the sailboat can sail and the velocity is plotted corresponding to the angle. When a sailboat is sailing towards a target that is directly upwind it needs to zig-zag to move towards the specific target. A "tack" is when a sailboat changes directions and crosses through the "no go zone." Since there is an infinite amount of paths the boat can take to get to the target by zig-zagging the traditional static PI controller does not perform optimally. Especially when there are restrictions on where the sailboat can sail, time minimization, or resource constraints.

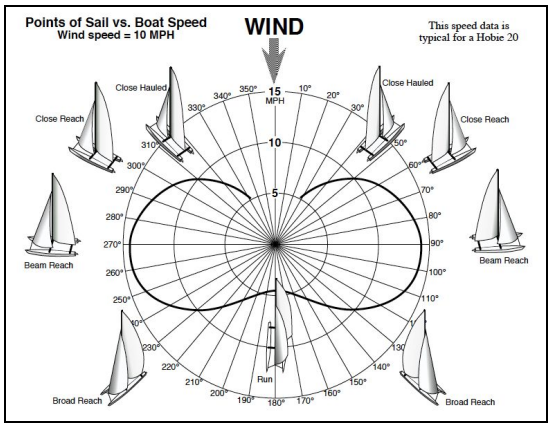


Figure 1: No-go-zone of a sailboat

Sailing upwind calls for a dynamic control law and consequently a path planning optimization technique in order for the sailboat to successfully complete a mission. Therefore, a gain-scheduling control strategy and short-term path optimization algorithm proposed and assessed in this paper [3].

Mathematical Model

Previous work done by the University of South Denmark set out to construct a static PI controller to change the rudder and sail angle of a sailboat to keep it tracking in a straight line. Due to how complex controlling a sailboat is based on different environmental conditions the authors wanted to increase the fidelity of the control by solving two problems. First, they wanted to change the low-level control of the sailboat by changing the static control to a Gain Schedule PI controller. Second, they wanted to find the optimal path for a sailboat to sail specifically in constrained environments, and while the boat was sailing upwind. These are two of the hardest environments an autonomous sailboat could find itself.

To model the dynamics of the autonomous sailboat, a challenge considering the ocean's non-linear environment, the authors utilized the previous work of University of South Denmark [2]. In this model, a four degree of freedom (DOF) system is utilized to model the non-linear

Michael Levy
Prit Chovatiya

sailboat dynamics. They set up two frames, a body-fixed frame to the sailboat frame and an inertial frame for the sailboat. These two frames, as well as the forces on the sail, are shown in figure 1 and 2. The body-fixed frame has angular velocity $w = [p, q, r]^T$ relative to the inertial frame. Due to the ocean environment, the boat can pitch up and down due to the waves, roll due to the force of the sail, and yaw due to the force of the rudder or current. The dynamic model included the roll of the sailboat, which was crucial to ensure that it is as accurate as possible. This is because when a sailboat hull rolls, the angle of the keel, rudder, and sail are all affected. All of these foils have a varied lift, and drag forces that change direction due to the roll of the sailboat.

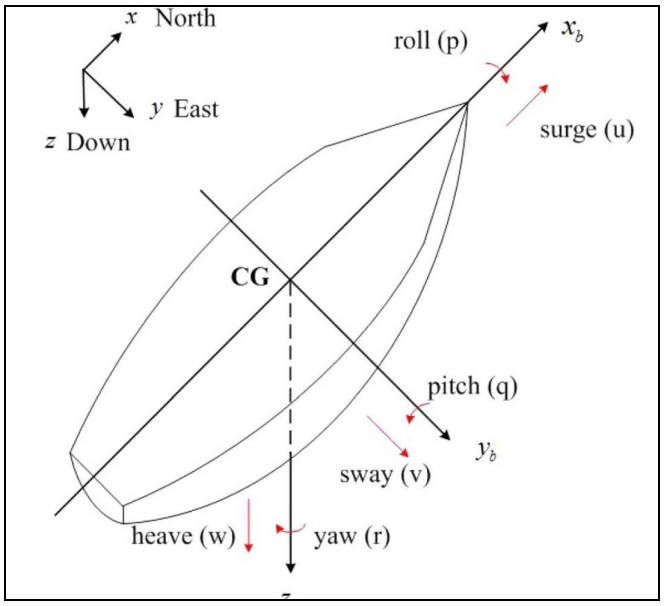


Figure 2: Inertial Frame of the sailboat

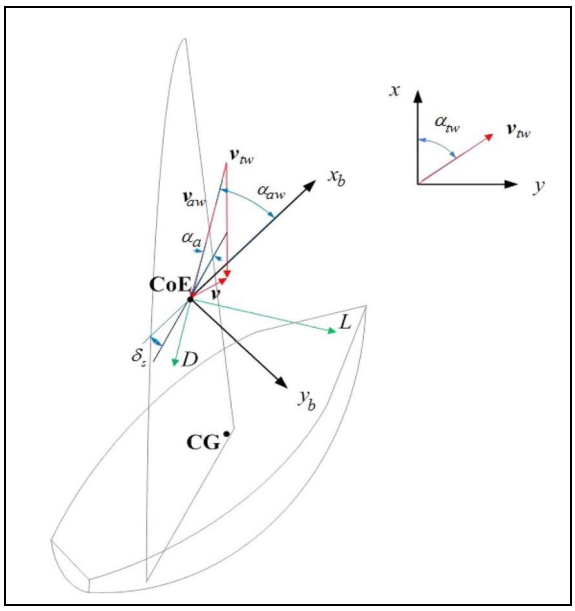


Figure 3: Forces on the sailboat

The model is then used via MATLAB and the Simulink toolbox to approximate the dynamics of the sailboat. The equation of motion for the sailboat is given by the following state-space equation:

$$\boldsymbol{M\dot{\nu}} + \boldsymbol{C(\nu)\nu} + \boldsymbol{D(\nu,\eta)} + \boldsymbol{g(\eta)} = \boldsymbol{\tau}$$

where the '*M'* is the sailboat mass inertia matrix, '*v'* is the velocity, *'C(v)'* is the system Coriolis-centripetal matrix, *'D'* is the system vector of damping and *'g(n)'* represents the vector of restoring forces. Consequently, the mass *'M'* is given by the sum of the rigid-body mass inertia matrix 'M_{RB}' and the added-mass inertia matrix 'M_A' and are given below:

Michael Levy
Prit Chovatiya

$$\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_{A}$$

$$\boldsymbol{M}_{RB} = \begin{bmatrix} m\boldsymbol{I}_{2\times 2} & \boldsymbol{0}_{2\times 2} \\ \boldsymbol{0}_{2\times 2} & \boldsymbol{I} \end{bmatrix}, \quad \boldsymbol{I} = \begin{bmatrix} I_{xx} & -I_{xz} \\ -I_{xz} & I_{zz} \end{bmatrix}$$

$$\boldsymbol{M}_{A} = -\begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{p}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{p}} & Y_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{p}} & K_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{p}} & N_{\dot{r}} \end{bmatrix}$$

Furthermore, the system Coriolis-centripetal matrix *'C(v)'* arises as the sum of *'C_{RB}(v)'* rigid-body Coriolis-centripetal matrix and *'C_A(v)'* added-mass Coriolis-centripetal matrix and are shown below:

$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C}_{RB}(\boldsymbol{\nu}) + \boldsymbol{C}_{A}(\boldsymbol{\nu})$$

$$\boldsymbol{C}_{RB}(\boldsymbol{\nu}) = \begin{bmatrix} 0 & -\mathrm{M}_r & 0 & 0 \\ \mathrm{M}_r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{C}_{A}(\boldsymbol{\nu}) = \begin{bmatrix} \boldsymbol{0}_{2\times 2} & \boldsymbol{C}_{A12}(\boldsymbol{\nu}) \\ \boldsymbol{C}_{A21}(\boldsymbol{\nu}) & \boldsymbol{C}_{A22}(\boldsymbol{\nu}) \end{bmatrix}$$

The system vector of damping *'D'* is given from the sum of the keel, hull, and added heel and yaw damping matrices given below:

$$\boldsymbol{D}(\boldsymbol{\nu}, \boldsymbol{\eta}) = \boldsymbol{D}_{\mathrm{k}}(\boldsymbol{\nu}) + \boldsymbol{D}_{\mathrm{h}}(\boldsymbol{\nu}, \boldsymbol{\eta}) + \boldsymbol{D}_{\mathrm{heel}}(\boldsymbol{\nu}) + \boldsymbol{D}_{\mathrm{yaw}}(\boldsymbol{\nu}, \boldsymbol{\eta})$$

$$\boldsymbol{D}_{\mathrm{k}}(\boldsymbol{\nu}) = \begin{bmatrix} -L_k \sin\alpha_{ak} + D_k \cos\alpha_{ak} \\ -L_k \cos\alpha_{ak} - D_k \sin\alpha_{ak} \\ (-L_k \cos\alpha_{ak} - D_k \sin\alpha_{ak})|z_k| \\ (L_k \cos\alpha_{ak} + D_k \sin\alpha_{ak})|x_k| \end{bmatrix}$$

$$\boldsymbol{D}_{\mathrm{h}}(\boldsymbol{\nu}, \boldsymbol{\eta}) = \begin{bmatrix} F_{rh}(\upsilon_{ah}) \cos\alpha_{ah} \\ -F_{rh}(\upsilon_{ah}) \sin\alpha_{ah} \cos\phi \\ (-F_{rh}(\upsilon_{ah}) \sin\alpha_{ah} \cos\phi)|z_h| \\ -F_{rh}(\upsilon_{ah}) \sin\alpha_{ah} \cos\phi|x_h| \end{bmatrix}$$

$$\boldsymbol{D}_{\mathrm{heel}}(\boldsymbol{\nu}) + \boldsymbol{D}_{\mathrm{yaw}}(\boldsymbol{\nu}, \boldsymbol{\eta}) = \begin{bmatrix} 0 \\ 0 \\ c\dot{\phi}|\dot{\phi}| \\ d\dot{\theta}|\dot{\theta}| \cos\phi \end{bmatrix}$$

Michael Levy
Prit Chovatiya

The lift and drag *'L'* and *'D'* are given by the conventional equations:

$$L = \frac{1}{2}\rho A v_a^2 \, \mathrm{C}_L(\alpha), \quad D = \frac{1}{2}\rho A v_a^2 \, \mathrm{C}_\mathrm{D}(\alpha)$$

The lift and drag forces are both found with a lookup table based on the apparent velocity and angle of the fluid flowing over it for the sail, keel, rudder, and hull. These values were acquired from the University of Auckland in their experimental facilities [3]. In order to calculate the angle the equation below is used:

$$\boldsymbol{v}_{\mathrm{aw}} = \boldsymbol{v}_{\mathrm{tw}}^b - \boldsymbol{v} - \boldsymbol{\omega} \times [x_s, y_s, z_s]^T$$

$$\alpha_{\mathrm{aw}} = \arctan 2(v_{awv}, -v_{awu}).^2$$

Where the true wind velocity is shifted into the body fixed from the inertial using two Euler angle rotations. The velocity of the boat is then subtracted from it and the moment created by the roll of the sailboat is incorporated as well. This equation is not only used for the sail, but it is also used for the rudder, hull, and keel. All with their respective fluids and angles. After computing the apparent fluid angle, the angle of attack used for lift and drag coefficient values is shown below, it incorporates the trim angle of the sail. The same method is used for the rudder. And with the keel and hull just the apparent angle is used.

$$\alpha_s(\boldsymbol{\eta}, \boldsymbol{\nu}, \delta_s, v_{\mathrm{tw}}, \alpha_{\mathrm{tw}}) = \alpha_{\mathrm{aw}}(\boldsymbol{\eta}, \boldsymbol{\nu}, v_{\mathrm{tw}}, \alpha_{\mathrm{tw}}) - \delta_s$$

The restoring force vector *'g(n)'* is given by the following vector. The coefficients a and b would be found by an inclining test for the specific sailboat design:

$$\boldsymbol{g}(\boldsymbol{\eta}) = \begin{bmatrix} 0 \\ 0 \\ a\phi^2 + b\phi \\ 0 \end{bmatrix}$$

As seen above, the restoring forces vector is a function of the orientation and position *'n'* which is with respect to the frame of reference *'W'* and is given by:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu}$$

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \boldsymbol{J}_1(\boldsymbol{\eta}) & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \boldsymbol{J}_2(\boldsymbol{\eta}) \end{bmatrix}$$

where '$J_1(n)$' ,'$J_2(n)$' and velocity *'v'* are given by the following equations respectively:

Michael Levy
Prit Chovatiya

$$\boldsymbol{J}_1(\boldsymbol{\eta}) = \begin{bmatrix} \cos\theta & -\sin\theta\cos\phi \\ \sin\theta & \cos\theta\sin\phi \end{bmatrix}$$

$$\boldsymbol{J}_2(\boldsymbol{\eta}) = \begin{bmatrix} 1 & 0 \\ 0 & cos\phi \end{bmatrix}$$

$$\dot{\boldsymbol{\nu}} = -\boldsymbol{M}^{-1}\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{M}^{-1}\boldsymbol{D}(\boldsymbol{\nu},\boldsymbol{\eta}) - \boldsymbol{M}^{-1}\boldsymbol{g}(\boldsymbol{\eta}) + \boldsymbol{M}^{-1}\boldsymbol{\tau}(\boldsymbol{\eta},\boldsymbol{\nu},\delta_r,\delta_s,v_{tw},\alpha_{tw})$$

Finally, the resulting state-space equations describing the sailboat dynamics are given by equations of change in η and the above equation, where "T" represents the resulting propulsive forces of the sailboat that is the sum of the sail and the rudder forces "T_S" and "T_R" respectively:

$$\boldsymbol{\tau}(\boldsymbol{\eta},\boldsymbol{\nu},\delta_r,\delta_s,v_{tw},\alpha_{tw}) = \boldsymbol{\tau}_s(\boldsymbol{\eta},\boldsymbol{\nu},\delta_s,v_{tw},\alpha_{tw}) + \boldsymbol{\tau}_r(\boldsymbol{\nu},\delta_r)$$

$$\boldsymbol{\tau}_s(\boldsymbol{\eta},\boldsymbol{\nu},\delta_s,v_{tw},\alpha_{tw}) = \begin{bmatrix} L_s\sin\alpha_{aw} - D_s\cos\alpha_{aw} \\ L_s\cos\alpha_{aw} + D_s\sin\alpha_{aw} \\ (L_s\cos\alpha_{aw} + D_s\sin\alpha_{aw})|z_s| \\ -(L_s\sin\alpha_{aw} - D_s\cos\alpha_{aw})x_{sm}\sin\delta_s \\ +(L_s\cos\alpha_{aw} + D_s\sin\alpha_{aw})(x_m - x_{sm}\cos\delta_s) \end{bmatrix}$$

$$\boldsymbol{\tau}_r(\boldsymbol{\nu},\delta_r) = \begin{bmatrix} L_r\sin\alpha_{ar} - D_r\cos\alpha_{ar} \\ L_r\cos\alpha_{ar} + D_r\sin\alpha_{ar} \\ (L_r\cos\alpha_{ar} + D_r\sin\alpha_{ar})|z_r| \\ (-L_r\cos\alpha_{ar} - D_r\sin\alpha_{ar})|x_r| \end{bmatrix}$$

This results in the sailboat state-vector given by:

[x; y; ϕ; θ; u; v; p; r]

where (x, y) are position coordinates, (u, v) are velocity components, (phi, theta) are Euler's angles, (p, r) are angular velocities in the w-frame.

Assumptions

The dynamical model used in some cases large assumptions that are important to recognize. First, the heaving and pitching motion was assumed to be zero. This essentially assumes that the boat is sailing in very calm waters (no current or waves). Additionally, in the context of the added mass calculation, the rolling motion is neglected and thus, the added mass coefficients are held constant. The sailboat is essentially assumed to be upright for the calculations. The forces of the sail, keel, rudder, and hull are all decoupled and considered independently of each other. Additionally, the authors applied a single lift and drag force along the center of effort for the keel, hull, rudder, and sail.

Techniques Used

Gain Scheduling PI Controller (Low-Level Control)

The low-level control of a sailboat consists of controlling the angle of attack of the sail(s) and the rudder angle. With the low-level controller, path planning is not considered, the goal is to have the boat follow a desired motion. For sailing, the control of the sail is often largely dependent on the angle of the apparent wind. Due to this simplicity, the sail angle is controlled by looking at the apparent wind direction given by the wind sensors onboard and adjusting the angle accordingly. The rudder movement is quite complex in comparison to the sail adjustments. This is due to how rudder control requires smooth steady input in order to control the heel, speed, and point of sail of the boat. This is why a static PI controller has been chosen by previous investigations on autonomous sailboats. A PI controller offers fast, simple implementation and is quite robust. Normally the K_p and K_i gains for the static PI controller are found by identifying the worst-case scenario for the sailboat to maneuver in (the scenario where the controller performs the worst) and tuning the constants accordingly for the best performance. These constants are then set for all scenarios.

The authors set about to change the low-level controller from a static PI controller to a gain scheduling (GS) PI controller. This method offers the same robust PI control as previous works, however, the sailboat uses the optimal K_p and K_i gains for every scenario it is in. Both field experiments and simulations were used in order to find the best K_p and K_i gains. From actual field testing, the authors found that the initial orientation, θ (the difference between North and the boat's heading), and the apparent wind direction (α) had the largest impact on the dynamics of the sailboat. Intuitively this makes sense as the loads on the rudder change as the apparent wind angle evolves. Additionally, the rudder needs to be handled differently when the sailboat is headed towards an upwind target versus a downwind target. The authors then set about a structure on how to find the best K_p and K_i gains. First, a type of mission must be defined with the necessary restrictions identified. Then the initial heading is divided into eight pairs of initial heading angles $(0^\circ, \pm 45^\circ, \pm 90^\circ, \pm 135^\circ, \pm 180^\circ)$ and four apparent wind directions

(45°, 90°, 135°, 180°). Thus, for a specific mission type, a lookup table of 32 gain pairs for specific maneuvers is created.

In order to create the lookup table, an exhaustive brute force search algorithm was used. This algorithm is shown in figure 4. The first two "for loops" ensure that all possible apparent wind directions and initial heading angle combinations are tested. The next two "for loops" ensure that all possible combinations of K_p and K_i gains are considered. The range of possible (K_p, K_i) pairs are bounded by maximum values for the specific hardware used. Next, the simulator is run that returns the value that is being minimized. For this investigation, the authors use a scenario where the sailboat is navigating a river collecting water quality measurements. The sailboat has a predefined initial position as well as a target location for each simulation. The simulator returns the maximum lateral distance the sailboat achieved. The goal of this is to find K_p and K_i gains that keep the sailboat within the safe zone of the river while minimizing lateral distance. If the sailboat does not reach the target then the simulator returns -1. If the lateral distance returned is a minimum for the specific initial heading and apparent wind angle the K_p and K_i gains are saved into the lookup table N. Once completed, this algorithm returns 32 optimal K_p and K_i gains for the sailboat to use through an exhaustive brute force search.

```
1  begin
2      θ ← 45;
3      for i_w from 1 to 4 do
4          α_aw ← 45 + (i_w − 1) × 45;
5          for i_s from 1 to 8 do
6              θ ← θ − 45;        d_min ← 0;
7              for k_p from 1 with step 0.5 to 10.5 do
8                  for k_i from 1 to 20 do
9                      d_l ← sim(θ, α_aw, K_p, K_i);
10                     if d_l ≠ −1 then
11                         if d_l < d_min then
12                             d_min ← d_l;
13                             N_{i_s,i_w} ← (K_p, K_i);
14                         end
15                     end
16                 end
17             end
18             if θ = −180 then
19                 θ ← 180;
20             end
21         end
22     end
23 end
   Output: N
```

Figure 4: Pseudo Code for GS-PI

Michael Levy
Prit Chovatiya

Short term Path Planner (High-Level Control)

The second crucial control law explored in this paper covers the trajectory planning aspect of the sailboat for targets that lie upwind. As stated earlier, this is a particularly difficult task for a sailboat for two reasons. Firstly, the sailboat cannot take a direct path to the target and hence has to repeatedly "tack" to move upwind. As seen in figure 5, for a target location B given a starting point A, the sailboat has to move in a zig-zag fashion which offers many possible trajectories for doing so. Here, the P1 route in red restricts its movements to a small lateral displacement from the target, which leads to an increased amount of tacks. While routes P2 and P3 tack significantly less while increasing the lateral displacement from the target point. In this paper, two parameters are considered to dominate these maneuvers: the angle of approach 'θ_t' for the tacking and tacking distance 'd_t'.

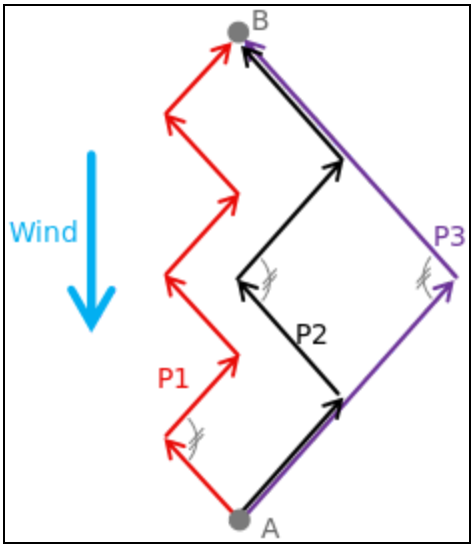


Figure 5: Tacking of a sailboat for upwind trajectories

This scenario has a myriad of possible combinations of angles and tacking distances making it a complex problem for path planning. Additionally, the wind direction shifting throughout the approach of the target will make the velocity made good (VMG) towards the target vary while the wind is changing direction. The VMG is the component of the boat's velocity vector that is going straight towards the target. Thus, the closer the angle the boat can sail towards the target, the faster it will get to it. The wind direction shifting (as it normally does) adds further complexity to the path planning, which requires a global optimization method to find the most efficient maneuver given a particular task. Developing a real-time optimization based on these parameters is crucial as it offers flexibility for the sailboat to change its route planning strategies for varied mission conditions. For instance, a water-quality assessment study in a narrow channel would require the sailboat to cover a diverse area, for which the tacking distance can be constrained to a larger set in the optimization formulation. Contrarily, a rescue mission would

Michael Levy
Prit Chovatiya

require the sailboat to change its control to ensure a minimum-time optimization. To investigate this, a path planning algorithm is implemented to decide optimal waypoints to conduct a tack. This results in an optimal orientation and displacement for the sailboat.

A mathematical model for tacking is responsible for finding points where the boat should tack between the target location and the boat's current location. This requirement is met if the points in the trajectory lie in a straight line with respect to the previous point and is hereby mathematically formulated. In the figure given below, 'P_0' represents the starting point, 'P_d' is the desired location, '$\boldsymbol{\theta}_t$' is the orientation and 'd_t' is the tacking displacement [1]. The lines 'L_1' and 'L_2' represent the maximum tacking displacement around the desired location and 'P_t' is the projection of the trajectory from the previous point for a given angle on the tacking displacement lines. It is to be noted that the figure below shows the first step in the trajectory which is repeated until the desired point is reached.

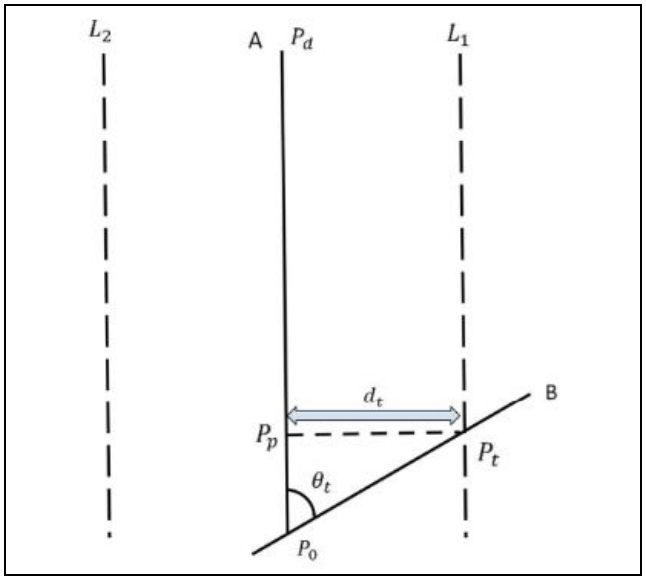


Figure 6: Short-term path planning geometry

The starting point $P_0 = (x_o, y_0)$ and the ending point $P_d = (x_d, y_d)$ are connected by line 'A' which can be given by the following equation:

$$a_A = \frac{y_d - y_0}{x_d - x_0}; \qquad b_A = y_0 - a_A x_0$$

where 'a_A' is the slope of the line and 'b_A' is the equation of the line. Furthermore, the equation for propagating the trajectory for the given orientation '$\boldsymbol{\theta}_t$' and displacement 'd_t' is shown below. Where the orientation is the difference between the line to the target and the current heading angle of the sailboat.

Michael Levy
Prit Chovatiya

$$a_B = \frac{a_A + \tan(\theta_t)}{a_A \tan(\theta_t) - 1}; \qquad b_B = y_0 - a_B x_0$$

Where 'a_B' is the slope and 'b_B' is the equation of the line. This allows us to obtain the projection 'P_t' = (x_t, y_t) of the maximum allowable displacement 'd_t' on the resulting path. This is then used to obtain the equations of the lateral displacement lines around 'b_A':

$$b_{L_1} = y_t - a_A x_t; \qquad b_{L_2} = 2b_A - b_{L_1}$$

Finally, the number of tacking points for a given step size '$d_{o,p}$' that is required to go from the starting point to the target point is given by the following equation:

$$k = \left\lfloor \frac{d_{0,d}}{d_{0,p}} \right\rfloor$$

The parameter 'k' is then used to update the step length after every tacking maneuver by using the following difference equations:

$$\Delta x_{p,0} = x_p - x_0; \qquad \Delta y_{p,0} = y_p - y_0$$

This results in a new point 'P_k' = (x_k, y_k) which can be computed by $x_k = k\,\Delta x$ and $y_k = k\,\Delta y$ for k = 1, 2, 3 …. n steps in the maneuver. As one would expect, an odd number of steps k = (1, 3, 5…) and even steps k = (2, 4, 6…) would lie on the tacking bounds L_1 and L_2 respectively. For the general case where the desired target is not exactly at an angle of 0^o to the wind ($e \neq 0$), then the distance between L_1 and L_2 is modified appropriately to account for the same. This allows the orientation to be constant which simplifies the problem further. The algorithm is summarized by the pseudo-code given below.

Michael Levy
Prit Chovatiya

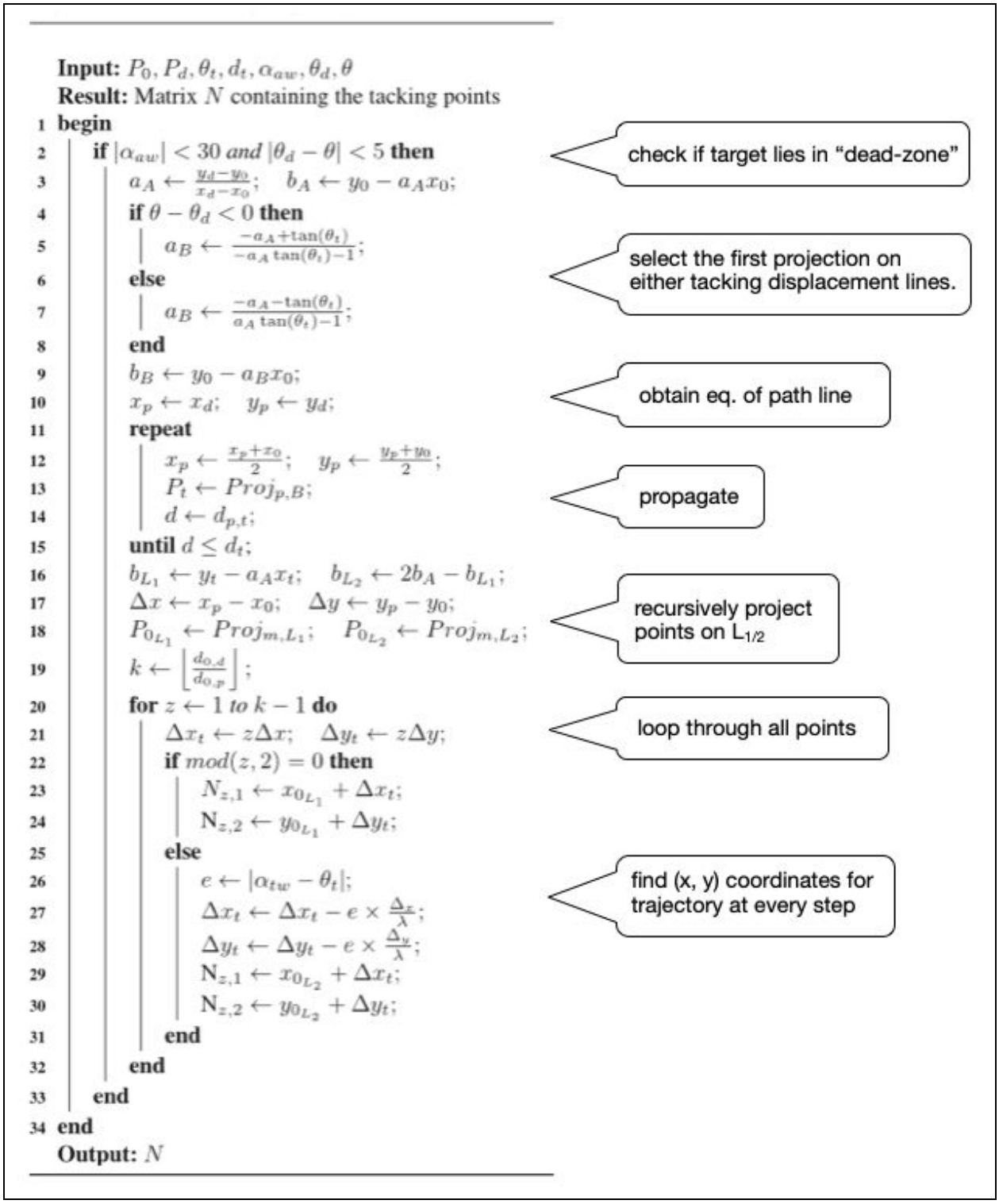


Figure 7: Pseudocode for short-term path planning

To verify the functioning of this algorithm the authors did a simple experiment where they iteratively tested various combinations of orientation '$\boldsymbol{\Theta}_t$' and displacement 'd_t'. The orientation

Michael Levy
Prit Chovatiya

was varied from 20º to 80º and displacement was changed from 10% to 90% of the distance to the target. This represents scenarios where the wind might be lower and the boat will have to tack through a larger angle, versus heavier wind where the boat can sail closer to the wind. The experimental results were plotted on a contour plot in figure 8 where the two design parameters were on the x-y axis and the mission-completion-time was plotted on the z-axis.

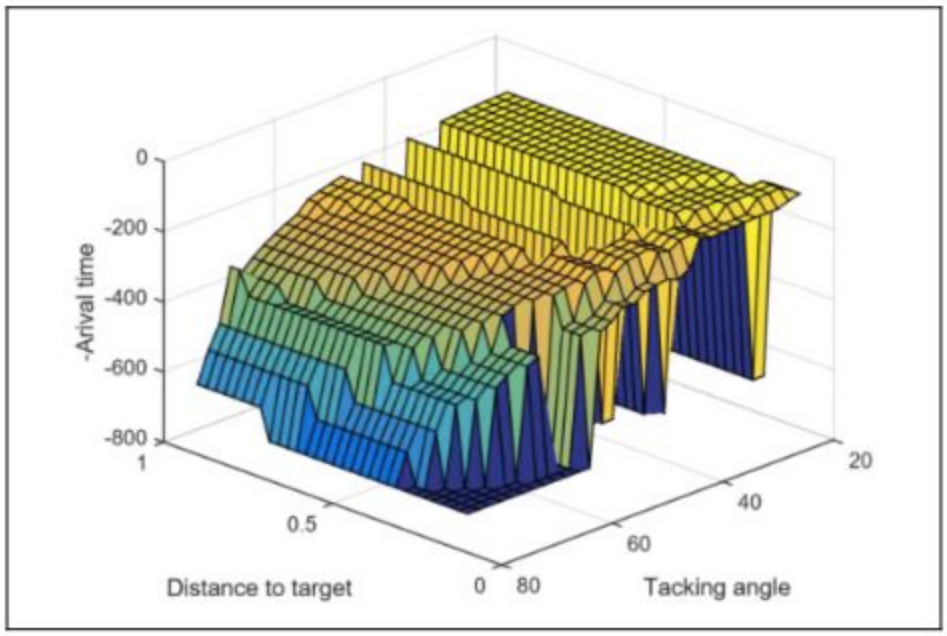


Figure 8: Initial results from brute force

As it can be seen from figure 8, there are multiple local minima for different sets of parameters for a constant wind orientation. Since a constant wind angle is hardly the case in a real scenario, experimenting with varying parameters is necessary. Variations in the sailboat heading angle and speed, wind angle and speed, tacking orientations and displacements are important to consider for efficiently planning a trajectory. Consequently, the above-mentioned design parameters result in a large design space which can be best optimized by an optimization technique, particularly a global search based genetic algorithm (GA).

Michael Levy
Prit Chovatiya

Input: parameters of the genetic algorithm
Result: θ_t and d_t that are optimal

```
1  begin
2      initialization of parameters of genetic algorithm;
3      random generation of initial population;
4      for i ← 1 to N_ger do
5          foreach individual do
6              find the time to target using individual's θ_t
                and d_t;
7          end
8          compute fitness;
9          apply roulette selection;
10         apply crossover;
11         apply mutation;
12         store best individual;
13         generate new population;
14     end
15 end
```

Output: best individual (θ_t, d_t)

Figure 9: Pseudocode for genetic algorithm

A genetic algorithm is a type of an evolutionary-inspired optimization method which uses the principle of "survival of the fittest". The algorithm randomly generates *N* number of design candidates ($\boldsymbol{\theta}_t$, d_t) and computes the objective function value (or their fitness) for each one of them. Here, the objective can be either minimizing time/energy or maximizing area, but for the sailboat trajectory optimization, we use minimizing time as the objective. Design candidates that yield the lowest time to complete a mission are pooled together for 'parenting' the next generation of design candidates. This is done by a method called 'roulette selection' to ensure that 'fit' individuals are given priority for translating 'good traits' to the next generation. This process of generating a population, evaluating the fitness and performing crossover is done repeatedly for a given number of generations until a stopping criterion is reached. The algorithm can be further explained by the pseudo-code provided in figure 9. Finally, the algorithm yields the tacking points that minimize the mission time, which is then used by the GS-PI controller.

Michael Levy
Prit Chovatiya

Results

Gain Scheduling PI Controller (Low-Level Control)

The resultant $(K_p,\ K_i)$ lookup table from the GS PI brute force algorithm is shown in table 1. In order to validate these results, the authors compared a proportional controller $(K_p = 1,\ K_i = 0)$, as well as a static PI controller $(K_p = 1,\ K_i = 14)$. The static controller K_p and K_i gains were determined by finding the simulation with the maximum lateral distance (hardest scenario for the sailboat) and then performing an exhaustive brute force search to find the K_p and K_i gains that provided a minimum distance for the scenario.

$\theta \backslash \alpha_{tw}$	45°	90°	135°	180°
0°	(9, 18)	(1, 13)	(10.5, 20)	(10.5, 20)
−45°	(10.5, 20)	(10.5, 20)	(10.5, 20)	(10.5, 20)
−90°	(1.5, 20)	(1.5, 20)	(1.5, 20)	(1.5, 20)
−135°	(1, 15)	(1, 15)	(1, 16)	(1, 14)
180°	(10.5, 2)	(10.5, 2)	(1, 1)	(10.5, 2)
135°	(1, 14)	(1, 13)	(2.5, 1)	(1, 14)
90°	(1.5, 20)	(10.5, 13)	(3.5, 17)	(1.5, 20)
45°	(10.5, 20)	(10.5, 20)	(10.5, 20)	(10.5, 20)

Table 1: GS $(K_p,\ K_i)$ Lookup Table for River Navigation

Shown in figure 10 is the resultant trajectory for all three control methods with an initial heading of 45° and wind direction of 180°. All three controllers clearly reach the target while staying inside the safe zone, however, the GS-PI controller especially reduces the lateral distance covered by the sailboat by over 40% compared to the static PI controller. In figure 11, with a wind direction horizontal to the river, the GS-PI controller does a much better job heading towards the target and reducing the lateral distance covered by almost 30% over the static PI controller. The proportional controller (P) also does not stay inside the safe zone in this scenario. The authors showed that overall, the GS-PI controller is the best controller to use in narrow environments.

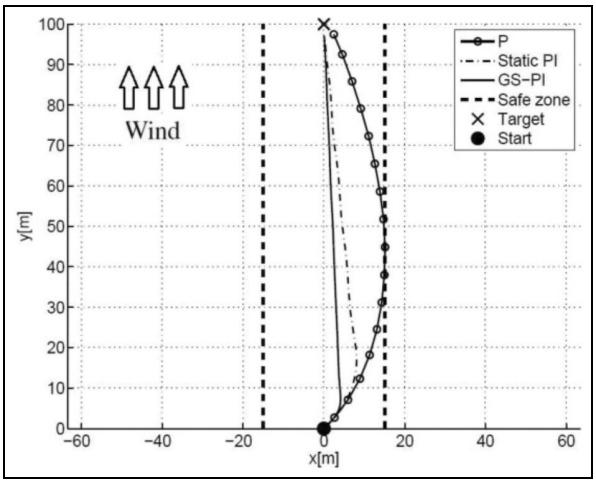


Figure 10: (θ = 45°, α = 180°)

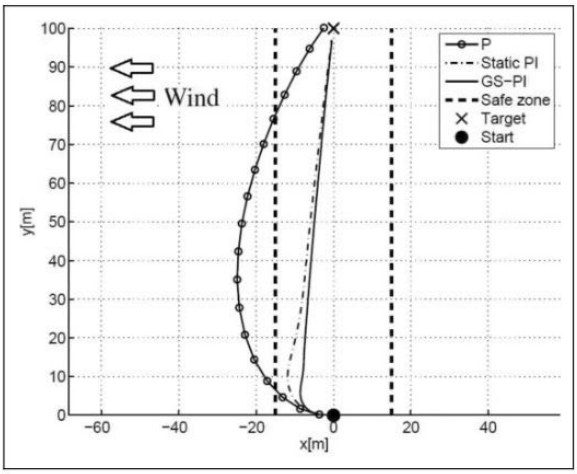


Figure 11: (θ =− 90°, α = 90°)

Overall, the GS-PI controller improved the sailboat's behavior and enabled it to reach targets efficiently in various environmental conditions. This meets the author's objective of creating a controller that can conduct operations in a constrained environment efficiently. The static PI controller would have been able to operate in the conditions presented in this paper, however, it would not mimic how a human would control the rudder. The GS-PI is a strong step in the right direction to having rudder control be the same as a human. The development here has shown to be a more efficient solution to an already robust controller. The method the authors took to create the GS-PI lookup table could be implemented again for different mission types and thus is a very useful development.

The largest assumption that was used throughout this paper was that there were calm waters and no current. Experiments with this controller operating in a real world environment is necessary to ensure that the simulator accurately found the optimal K_p and K_i gains. Additionally, it is clear from the lookup table that there are redundant initial positions that result in the same K_p and K_i gains. It is this author's belief that the true wind velocity may be another viable variable to create the GS-PI lookup table. From the author's personal experience, the control of a rudder needed in light winds is vastly different than the control needed in heavy winds. Incorporating true wind velocity into the lookup table may also compensate for real world sea conditions as the loading on the rudder due to heavy winds is similar to a rough sea state.

Short term Path Planner (High-Level Control)

Upon implementing the low-level and short-term path planning methodologies with the sailboat, three important experiments were conducted to obtain the results and visualize the performance of the proposed strategy. The first two experiments were to test the short-term path planning algorithm with a fixed orientation 'θ_t' and displacement 'd_t' while the third experiment tested the genetic algorithm for the variable pair (θ_t, d_t). For consistency, the starting point is set to (0, 0) with a heading angle of 0^o and an initial velocity vector of (2 m/s, 0 m/s) for all experiments. The first experiment tests the short-term path planning algorithm for upwind targets with a head-on wind vector of (0, -200 m/s) and a tacking displacement constraint of 30 meters. Furthermore, the experiment also varies the orientations 30^o to the left and 50^o to the right. In a similar fashion, the second experiment is run by modifying the wind heading angle to 25^o to the right [1]. The results from both the experiments are shown below.

Michael Levy
Prit Chovatiya

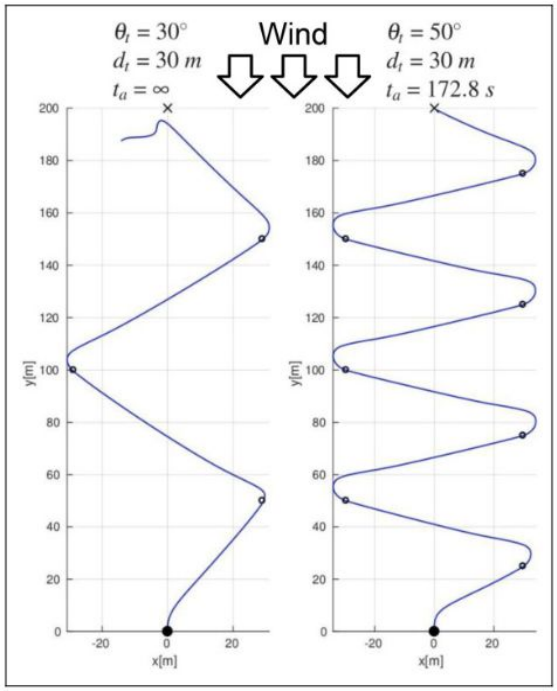


Figure 12: Wind Heading Angle = 0°

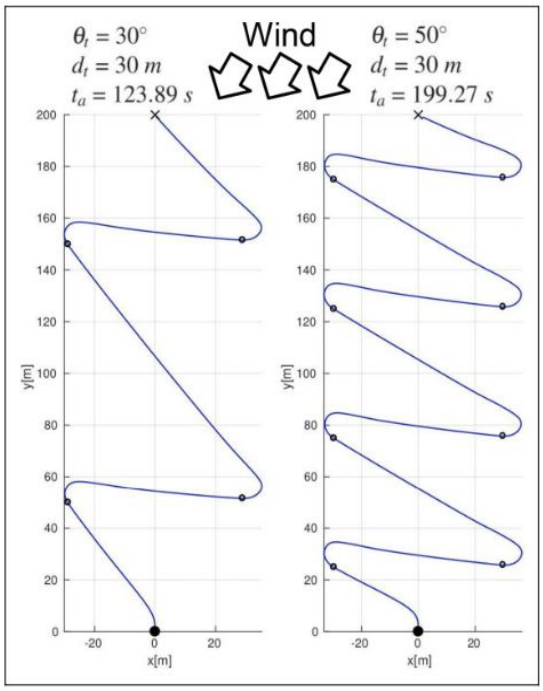


Figure 13: Wind Heading Angle = 25°

It can be seen that the algorithm performed as expected with some notable differences. When the orientation changes from 30° to 50°, the number of tacks increases due to the VMG being significantly lower. Furthermore, as the wind direction changes from 0° to 25°, the tacking points become asymmetrical but the path planning algorithm changes its control appropriately and still reaches the desired position. Additionally, the relationship between the number of tacking points, and the time it took for the sailboat to reach the target was directly proportional.

Upon testing the short-term path planning algorithm for minimum-time problems, it was also deemed valuable to test the same algorithm in conjunction with the low-level controller developed earlier. For this scenario, the sailboat was constrained to visit three waypoints within a time of 30-minutes and the results are shown below.

Michael Levy
Prit Chovatiya

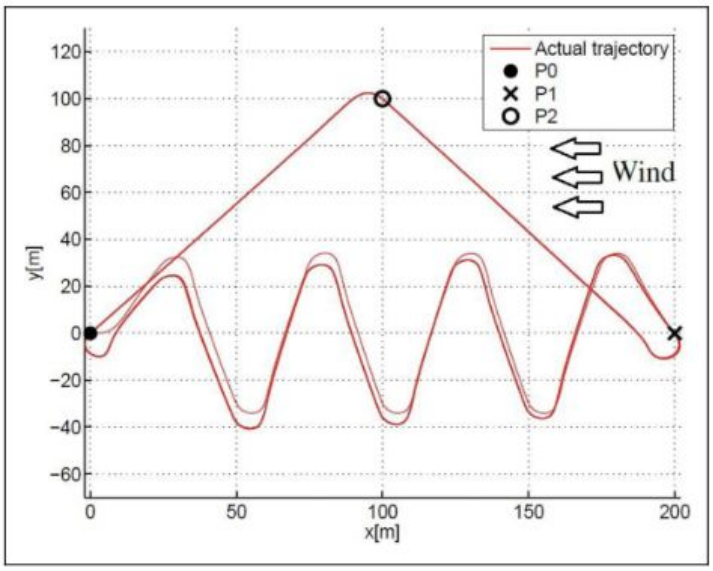


Figure 14: Multiple-waypoint scenario

As can be seen, the sailboat successfully covers all three waypoints in a 30-minute window autonomously. These experiments showed the successful functioning of the short-term path planning algorithm and revealed some key insights into obtaining efficient performance out of the sailboat.

On the other hand, the third experiment used the GA to optimize the orientation and displacement for a given wind direction and velocity with the same initial conditions as the previous experiments. To gain further insight into comparing the brute force method with the GA, both of the algorithms were tested on the previous scenarios mentioned in the table below.

Scenario	Wind direction	Wind velocity
1	60°	10 m/s
2	90°	15 m/s
3	60°	15 m/s

Table 2: Scenarios for testing GA

The results from the simulations are shown in the table below:

Scenario	Method	Time to target	Simulation time
1	Brute force	102.4 s	2364.3 s
	GA	97.72 s	176.48 s
2	Brute force	105.2 s	2172.42 s
	GA	114.44 s	216.28 s
3	Brute force	82 s	2559.96 s
	GA	80.16 s	196.32 s

Table 3: Brute Force Versus Simulation results

As seen in table 3, the time to target for the GA was less than the brute force algorithm in two of the three scenarios, pointing at a better performance by the GA. This can be attributed to the ability of the GA to explore design space which cannot be done by brute force and also the higher precision in design variables. On the contrary, the GA yielded a slower time to target for scenario-2 which can be attributed to the convergence of the GA to a local minima. Despite the slower performance in that particular scenario, the time to target was only 8.9% slower than the brute force method.

In all, GA does provide a great starting point for sailboat trajectory optimization for short-term path planning which performs better than previous methods explored in the field. Furthermore, the GA is extremely fast at finding an optimal set of tacking orientation and displacement as shown in Table 3. Another advantage of GA is its flexibility to find optimal solutions given varied constraints which can be easily modelled into the algorithm. This is important as this promises a scalable approach that can be implemented in numerous real-world applications making it indeed a very practical method. It is because of this that we think that the GA algorithm could be integrated in future sailboat architectures. On the other hand, this method fails to consider the dynamically changing wind directions while tacking. Due to the GA ability to incorporate different variables future models could incorporate varied wind directions. The recommended approach is to somehow integrate a gaussian model of the wind directions with the GA, which then dynamically changes it's approach accordingly. These gaussian models of wind directions could be gathered from different geographic locations where the sailboat will conduct operations. Another recommendation is to further implement various global optimization algorithms and evaluate their performance with respect to simulation time and their robustness to convergence to a global minimum.

Conclusion

Overall, a sailboat navigation system consisting of both a low-level control strategy (via a GS-PI controller) and a high-level control strategy (via short term path planning with GA) is successfully implemented and results verified. The GS-PI implementation has proven to increase the performance of the controller in varied wind and environmental conditions. Specifically the GS-PI controller out performed both the proportional (P) and static PI controller when navigating a narrow channel. The authors also provided an architecture to construct future GS-PI controllers for different missions. With regards to the path planning approach, a short-term path planning algorithm is implemented to find tacking waypoints that are further optimized by a GA. Experiments show that the aforementioned approach is successful in reaching the desired targets in less time and are computationally efficient. The method also shows promise in handling various time/energy/spatial constraints with ease and simplicity of implementation.

References

[1] Santos, D.H., Goncalves, L.M., "A gain-scheduling control strategy and short-term path optimization with genetic algorithm for autonomous navigation of a sailboat robot," International Journal of Advanced Robotic Systems, January-February, 2019, pp. 1-15.

[2] Xiao L., Jouffroy J., "Modeling and Nonlinear Heading Control of Sailing Yachts," IEE Journal of Oceanic Engineering, Vol. 39, No. 2, April, 2014.

[3] N. Davies, "A real-time yacht simulator," M.S. thesis, Dept. Eng. Sci., Univ. Auckland, Auckland, New Zealand, 1990.

[4] Images used in this paper are all obtained from [1]. Accessed on April 19, 2020.

Michael Levy
Prit Chovatiya

APPENDIX

Contributions

Michael Levy and Prit Chovatiya thoroughly enjoyed working on this paper together. They collaborated on the following topics: Importance of the Problem, Background, Mathematical Model, Assumptions, Techniques Used, Results, Conclusion, and finally the implementation of the GS-PI and GA (shown below). From the initial research to polishing the final draft of this paper, all the tasks were handled by both. Furthermore, for specific contributions, two major methods and results discussed in this paper: GS-PI and GA were primarily explored and discussed by Michael Levy and Prit Chovatiya respectively. They also switched roles and inspected each others' work in both the sections as well. Overall, this investigation was extremely interesting and proved to be a very valuable experience for both of us.

Implementation of the Genetic Algorithm (GA)

Upon thorough investigation of some of the optimization methods used in the paper, it was decided to implement the optimization for the short-term path planning algorithm. As a refresher, the GA was used to find the optimal tacking orientation and displacement ($\boldsymbol{\theta}_t$, d_t) to yield a minimum amount of time the sailboat take to travel to a desired location. In the original paper, this was done through a simulator which used these parameters and evaluated the time-to-target metric which is essentially the objective function for this problem.

Due to limited computational capabilities, running the simulator for the sailboat for GA optimization would take an unrealistic amount of time. Due to this, a "dummy" function was used to replace the simulator, which would yield faster results. This allowed us to implement a GA optimization algorithm which is essential to the methods used in the paper.

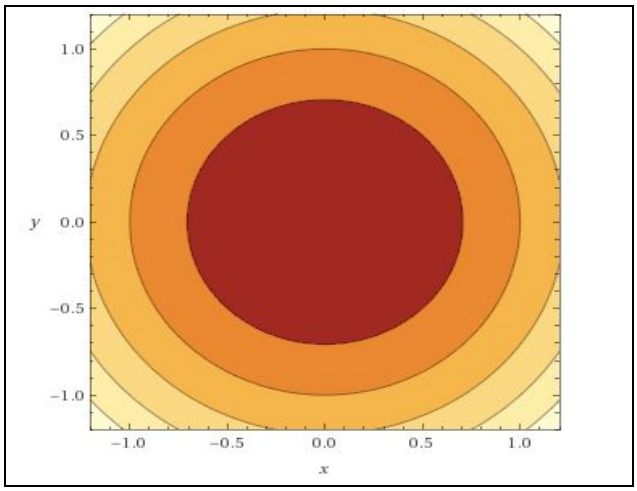
The optimization problem formulation was done in the following manner. The "dummy" objective function for the same is given below:

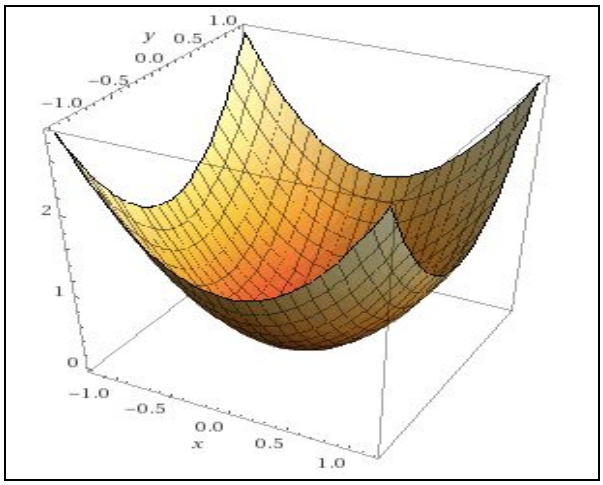
$$f(\boldsymbol{\theta}_t , d_t) = \boldsymbol{\theta}_t^2 + d_t^2$$

It was considered that this function supposedly yielded the time to target 'T', and as stated earlier in the paper, the goal is to minimize the this, hence the objective function which the GA maximizes turns out to be given by:

$$1 / f(\boldsymbol{\theta}_t , d_t) = 1 / (\boldsymbol{\theta}_t^2 + d_t^2)$$

To visualize this function and how it behaves, the contour plots below display the same.





The first step in the implementation is to generate a population size with Y generations and each generation has N number of individuals in them. The code for the following is shown below:

```
def generate_population(size, x_boundaries, y_boundaries):
    population = []
    for i in range(size):
        individual = {"x": random.uniform(x_boundaries[0],x_boundaries[1]),
                      "y": random.uniform(y_boundaries[0],y_boundaries[1])}
        population.append(individual)
    return population
```

Then the “fitness” for each of them is evaluated from the objective value the function yields. Based on those function values, the individuals are sorted in an ascending order and the “roulette” wheel technique is applied to them to choose the parenting population for the next generation. The code for the following is shown below:

Michael Levy
Prit Chovatiya

```
def choice_by_roulette(sorted_population, fitness_sum):
    offset = 0
    normalized_fitness_sum = fitness_sum
    lowest_fitness = objective_func(sorted_population[0])

    if lowest_fitness < 0:
        offset = -lowest_fitness
        normalized_fitness_sum += offset * len(sorted_population)

    draw = random.uniform(0, 1)

    accumulated = 0
    for individual in sorted_population:
        fitness = objective_func(individual) + offset
        probability = fitness / normalized_fitness_sum
        accumulated += probability

        if draw <= accumulated:
            return individual
```

Furthermore, the parenting population is then mutated by small perturbation, here we used a uniform random distribution for perturbation between [-0.05, 0.05]. Upon gaining the mutated population, two mutated individuals are “mated” using crossover technique. Here, the arithmetic mean is taken for the crossover but there are many other techniques too.

```
def crossover(individual_a, individual_b):
    # arithmetic mean is used for crossover
    xa = individual_a["x"]; ya = individual_a["y"]
    xb = individual_b["x"]; yb = individual_b["y"]
    return {"x": (xa + xb) / 2, "y": (ya + yb) / 2}

def mutate(individual):
    next_x = individual["x"] + random.uniform(-0.05, 0.05)
    next_y = individual["y"] + random.uniform(-0.05, 0.05)
    lower_boundary, upper_boundary = (-4, 4)
    # make sure don't violate variable bounds
    next_x = min(max(next_x, lower_boundary), upper_boundary)
    next_y = min(max(next_y, lower_boundary), upper_boundary)
    return {"x": next_x, "y": next_y}
```

Upon implementation of the three basic techniques required for the genetic algorithm namely: selection, mutation and crossover, the genetic algorithm recursively calls these functions for Y generations each with N individuals in them.

Michael Levy
Prit Chovatiya

```
def make_next_generation(previous_population):
    next_generation = []
    sorted_by_fitness_population = sorted(previous_population, key=objective_func)
    population_size = len(previous_population)
    fitness_sum = sum(objective_func(individual) for individual in population)

    for i in range(population_size):
        first_choice = choice_by_roulette(sorted_by_fitness_population, fitness_sum)
        second_choice = choice_by_roulette(sorted_by_fitness_population, fitness_sum)

        individual = crossover(first_choice, second_choice)
        individual = mutate(individual)
        next_generation.append(individual)

    return next_generation
```

For our case the genetic algorithm was run for 100 generations each consisting of 100 different individuals and the results are shown below:

```
# ------------------- running the GA -------------------- #

generations = 100
population = generate_population(size=100, x_boundaries=(-4, 4), y_boundaries=(-4, 4))

for i in range(1,generations+1): population = make_next_generation(population)

best_individual = sorted(population, key=objective_func)[-1]
print("Optimalilty Achieved")
print('Optimal orientation and displacement:',best_individual)
print('Function value at the optimum: ', 1/objective_func(best_individual))
```

```
Optimalilty Achieved
Optimal orientation and displacement: {'x': 0.004755540268566108, 'y': 0.008644530076823188}
Function value at the optimum:  9.734306349505452e-05
```

As it can be seen from the output that the GA found a point very close to the minimizer and yielded a minimum for the function. It is to be noted that since this is a heuristic method, it reveals slightly different results every run.

Implementation of GS-PI

It was clear that in order to simulate the GS-PI a simulator must be built to replicate the sailboat dynamics. An attempt at replicating the simulator described in the University of South Denmark paper is shown below [2]. All constants defined were used from their sailboat platform. Additionally, along with the simulator an accompanying GS-PI algorithm has been built to generate a lookup table. However, this lookup table would be done on apparent wind angle and true wind velocity. No investigation was done on this and it is this author's belief that the true wind speed will have a profound effect on the necessary K_p and K_i gains. Although the simulator was not able to be fully replicated, it is this author's belief that with more time accurate results could be created. The implementation of the simulator is shown on the published code below.