Schuyler Smith
CS 229 Final Project

12/13/13

Autonomous Hydrofoil Sailboat

Figure 1: A rendering of what our final boat will look like. Like the real AC72, it will be a catamaran with two hulls. The four hydrofoils can be seen projecting down beneath the boat: two near the center and two attached to the two rudders near the back. The final sail, shown here as a wireframe, will be rigid foam rather than fabric.

Introduction

The America's Cup is one of the oldest and most prestigious sailing competitions in the world. The most recent one, held this past summer in San Francisco, was especially unusual because it allowed boats to hydrofoil for the first time.

A hydrofoil is a wing-like structure mounted under the hull of a boat. As the boat increases its speed, the foils create lift that can raise the boat out of the water, which greatly reduces wetted area and hence drag: it is far more efficient to move water around just the foild than the entire hull. Consequently, hydrofoil sailboats can sail much faster than traditional boats, but they are much more difficult to control.

Some friends and I have formed the Stanford Autonomous Sailing Team, funded by Stanford Robotics Club, to design and build an autonomous scale model (one meter long, rather than 26 meters) of the AC72 high-performance hydrofoiling catamarans used in the most recent America's Cup.

Model foiling sailboats have been built before, but because of the sensitivity and complexity of the controls, human RC operators usually can't keep them "on foil" for more than a few seconds at a time. Using hydrofoils at this scale is particularly difficult because some of the techniques used by larger boats (such as the AC72) do not translate well. For example, a 26m boat like the AC72 is relatively immune to waves, but a 1m boat is not. As a result, the passive stabilization methods used by the AC72 will be inherently less effective at small scale. Our design, therefore, uses actively stabilized foils that remain completely submerged more reliably and are less affected by waves. This and other features make our design significantly more complex than existing designs, and as a result full human control will be impossible.

Hence, sailing our boat to its full potential is a nontrivial control problem. This paper will explore approaches I used to control the boat.

Today, the boat is still incomplete. We only began building the boat this quarter and plan to have it finished by the end of the school year so we can enter it in an autonomous sailing competition this summer. Hence, the results from this paper are from simulation only, but I believe that my simulation is reasonably realistic, and that the methods described will translate well to the real world.

Method

The boat has three main classes of sensors that can be used as input to a machine learning algorithm:

Position Provided by GPS and an IMU (accelerometer, gyroscope, compass)

Wind Speed and direction

Height Height above the water. This allows the boat to detect when it is on foil, and compensate for waves.

In turn, the boat will be able to control the position of the sail, the angle of the rudders (for steering), the elevation of four foils, and a movable ballast system (a weight that can be adjusted left and right to help balance the boat).

To distill boat control into an optimization problem, I separated higher-level planning (route finding and steering) from the task of sailing, and delegated these two tasks to the **planner** and the **trimmer**, respectively. The trimmer is the focus of this paper. The advantage of this approach is that since the path of the boat is out of its control, the trimmer has only one thing it needs to optimize - boat speed - so our reward function is very well defined. The planner also doesn't lend itself well to machine learning anyway, and is better implemented with more traditional deterministic route-finding algorithms.

Hence, we can formulate the task of the trimmer as a Markov Decision Process in the following way:

States will be represented as $s = (v_b, \vec{\phi}_b, \vec{v}_w, h) \in S$ where v_b is the instantaneous speed of the boat (direction is irrelevant to the trimmer, because we can translate everything into boat-centric coordinates), $\vec{v}_w$ is the velocity of the wind (relative to the boat), $\vec{\phi}_b$ is the rotation of the boat in three dimensions (pitch, yaw, and roll), and h is the height of the boat above the water. Hence, the state space is seven-dimensional.

Actions are represented as the control outputs $a = (\vec{\theta}_s, \vec{\theta}_f, b) \in A$, where $\vec{\theta}_s$ is the angle of the sail (which has two sections), $\vec{\theta}_f$ is the angle of the foils, and b is the position of the movable ballast. There are four independently adjustable foils, but we can exploit some more symmetry, and instead adjust only the average angle, left/right balance, and front/back balance. Hence, the action space has a total of six dimensions.

Rewards are boat speed. While conveniently easy to measure empirically, the relationship between states, actions, and rewards is obviously still complex.

The state space and action space are both effectively continuous. Because of the fairly high dimensionality (seven for the state space and six for the action space) and limitations of the processor onboard the boat, discretization of the state and action spaces ended up being impractical. With a large number of discrete states processing requirements became too large, and value iteration took a very long time to learn all the valuations. With a small number of discrete states sailing performance suffered because of the oversimplification.

Other requirements of the real boat also led to several other complications. First, testing a wide variety of states in the real world is impractical, because conditions do not vary quickly. For example, the boat must perform well in a variety of wind speeds, but on any given day the boat will only be exposed to a small range of wind speeds.

Second, experimentation in the real world carries with it a high cost for mistakes. Failure to stabilize the boat will flip it very quickly, and righting it and restarting it is something we would prefer not to have to do often.

An important first step in learning how to control the boat is building a model of the dynamics of the boat and state transitions. Unfortunately, as a result of these complications experimentation to build a deterministic or stochastic model is difficult.

To combat these, I use two main approaches. First, to improve the model's ability to generalize from very incomplete data, I built a fairly detailed model of the boat dynamics, which will be described in more detail in the Implementation section. To keep the model from getting too complex and due to development time constraints, the model actually predicts only most likely actions, it doesn't model the full probability distribution. That is, rather than modeling $P : S \times A \times S \to [0, 1]$, we model $T : S \times A \to S$.

Second, I used partial human control to guide exploration while learning the state space. Left to random exploration alone the boat tends to stay in one place, slowly drift downwind, or flip over. Instead, I used partial human assistance to aid learning. In simulation it's easier to control all aspects of the boat because I don't have to do it in real time. Since this won't be possible for the real boat, I developed a bootstrapping method where the computer manages an increasing subset of the controls over time (for example, the foil but not the rudder or sails). Again, more details will be presented in the Implementation section.

Once I have a model of state transitions I can perform value iteration as described in the lecture notes. However, I found that this was needlessly complicated. Since we want to optimize speed and speed is continuous, recursively computing true values is not useful - an action that increases the speed of the boat immediately is always better in the long run than one that does not. Hence, we can just as easily take the value of a state to be the reward for that state, the instantaneous speed of the boat. Then computing the optimal policy is straightforward - we just have to choose the action that optimizes the reward of the predicted next state.

$$\pi^*(s) \quad = \quad \arg\max_{a \in A} R(T(s, a))$$

However, this is still very complex to compute in practice because we must optimize over all actions. As discussed above, parameterizing the action space is out of the question.

It does seem reasonable, however, to want the controls to change smoothly (the sail can't actually adjust itself instantaneously anyway, for example). With that in mind, another approach I tried was to, rather than optimize over all possible actions, only consider actions that are small permutations from the current one. For example, for each setting x we can try x, $x + \epsilon$, and $x - \epsilon$. This is much more manageable, but with six dimensions in the action space and three choices each it still leaves $3^6 = 729$ possible actions. Unfortunately, while this seemed very promising, it also doesn't perform very well: in my tests I couldn't reach a balance between fine enough granularity to find local optima and coarse enough to respond quickly to changing conditions.

Hence, my final approach to actually controlling the boat is significantly different. Rather than calculating optimal actions directly from the Markov model, I decided to use the model to train an artificial neural network that controls the boat directly. That is, rather than using $T : S \times A \to S$ to choose actions, I derived a function $F : S \to A$. Normally this would be very inefficient, because training neural networks can take a long time and as described above time on the boat is at a premium. However, training the neural network from the model can be done offline, so this isn't an issue.

Implementation

All of my code is written in Python, using numpy, etc. where appropriate. The final sailboat uses an ARM-based microcontroller comparable in power to a smartphone.

Most of the early work on this project was spent developing the simulation I used to test everything. My teammates have run fluid-dynamic simulations of the sail and foils. My simulation uses the data from those runs to be as accurate as possible. From there, most of the actual physics is straightforward: the boat is modeled as a rigid body (except for the control surfaces), and calculating forces and torques is not complex.

I know that my model has some weaknesses. For example, I made no attempt to simulate the boundary layer effect: in reality wind speed increases as you rise further from the surface of the water. This changes the apparent speed and direction of the wind from the base to the tip of the sail. I also didn't put much effort into modeling waves. I have a generic noise term in the height measurements, but we don't yet know exactly how sensitive the boat will be to significant waves.

If performance on the real boat is significantly worse than in simulation these are two of many areas I can investigate, but until the boat is complete I don't believe it's constructive to worry too much.

The state transition model that the algorithm learns is based on a simplification of the simulation. In particular, it encodes most of the same physics, with everything heavily parameterized and with linear or quadratic error terms. The weights and offsets are learned in a manner similar to the method described in [1].

To train the state transition model, we need robust state exploration. As described above, this is bootstrapped with partial human control. At each stage the computer controls a subset of the action dimensions and chooses actions based on the optimal policy algorithm π^* described above. Because the size of the action space is greatly reduced and optimal performance isn't essential at this stage the disadvantages to this approach discussed earlier are less important.

The final controller is a feedforward artificial neural network with two hidden layers. The input layer is fed normalized state data, and the output layer produces actions. I use a Python library (arac) for the neural networks that is implemented in C++ and open-source, so it can be compiled for ARM and run efficiently directly on the boat's microcontroller.

To train the neural network I use a genetic algorithm to optimize the weights between nodes. Specifically, it tests the network in the simulator in 30 different conditions, and measures the average speeds the network acheived. On my quad-core desktop the performance of the network stops improving after about 1000 generations and around a day of CPU time. As mentioned above, this isn't a major drawback, because the learned weights can then be exported to the microcontroller, which can then activate the final network extremely quickly.

Results

Results are overall better than I expected. A graph of the results of optimizing the final neural network are shown in Figure 2. On the same test that the algorithm uses I scored about a 60 controlling the boat by hand in simulation, and the network acheives 1.5 times that performance in about 50 generations. Final performance is nearly twice my score. With practice I'm

sure I could improve my manual score, but nonetheless this is very impressive.

In Figure 3 you can see a heat map of the boat's performance at all points of sail. The boat performs very similarly to other high performance sailboats, so this somewhat validates my model (and our boat design). Like all sailboats, the boat performs poorly directly upwind (the top center of the map) and doesn't exceed the speed of the wind downwind (bottom of the map). However, it performs extremely well across the wind (either side of the map), which is the most important region.

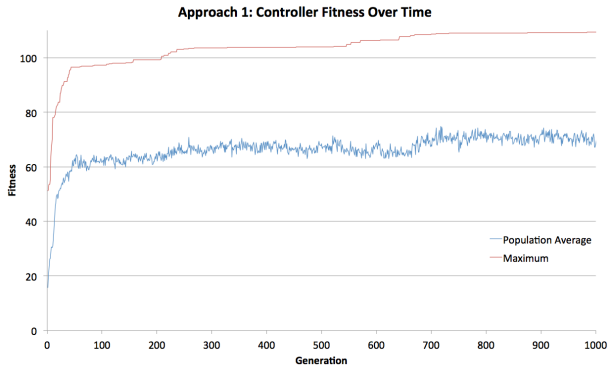

Figure 2: A graph of the neural network's performance over time, as it was optimized by the genetic algorithm.

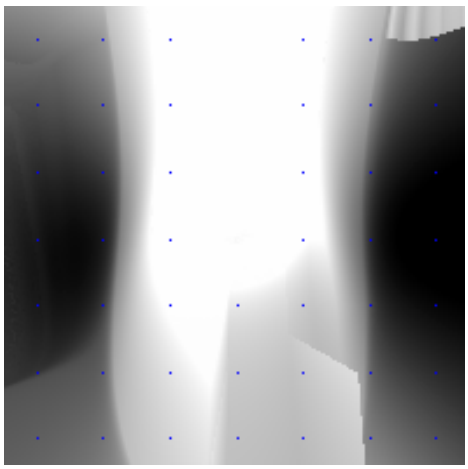

Figure 3: A heat map of the performance of the final controller at all wind speeds and directions. Zero wind is at the center, with directly upwind above that, directly downwind below, and across the wind to either side. White is speed 0, varying linearly to solid black at 3× wind speed. As expected, the boat cannot sail directly upwind nor downwind at more than the wind speed. Best performance is across the wind. The sharp transition on either side from light gray to darker gray is where the boat has enough wind transition to foils. The odd patterns in the downwind region are inconsequential, because it's more efficient to zig-zag diagonally and hit the much higher performance dark-grey regions to either side.

Future Work

The most important part of future work will be testing everything on the real boat. We only began building the boat this quarter and plan to have it finished by the end of the school year so we can enter it in an autonomous sailing competition this summer.

I expect that I'll need to make some changes to acheive real performance similar to my simulations. However, I hope that the structure of the model my system learns is robust enough to translate without too much difficulty.

If performance regresses, there are several avenues I can pursue. As mentioned above, there are some simplifications in my simulation and model that I'm aware of. I can also rework the model to directly model uncertainty. That in particular may significantly improve its performance in a real, noisier environment.

Acknowledgements

While the design and coding presented in this report are my own work, none of it would have been possible without the work of my teammates on the Stanford Autonomous Sailing Team: Wyatt Smith, Andrey Sushko, Thomas Teisberg, and Connor Anderson.

References

[1] Pieter Abbeel, Varun Ganapathi, Andrew Y. Ng. *Learning Vehicular Dynamics, With Applications to Modeling Helicopters.* NIPS 18, 2007.

[2] Pieter Abbeel, Adam Coates, Andrew Y. Ng. *Autonomous Helicopter Aerobatics through Apprenticeship Learning.* IJRP, 2010.

[3] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger and Eric Liang. *Inverted Autonomous Helicopter Flight Via Reinforcement Learning.* International Symposium on Experimental Robotics, 2004.

[4] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control.* Institut National Polytechnique de Grenoble, 2002.