Trajectory Planning for Underwater Gliders

Logan Williams

May 24, 2012

1 Abstract

The "underwater glider" is a highly useful underactuated robotic system. In this project, we analyze a two dimensional "planar underwater glider," with just one degree of actuation. By using rapidly exploring random trees, we are able to find feasible trajectories from one physical location to another. We then apply non-linear optimization techniques to the feasible trajectory to find efficient, low-power routes through the sea.

2 The Planar Underwater Glider Model

2.1 System motivation

The "underwater glider" is a very useful autonomous robot for oceanographic research, due to its ability to travel distance with minimal power consumption. The robot itself consists of a narrow, hydrodynamic body, with two long wings, as pictured in Figure 1. By inflating or deflating an oil bladder in the glider, the buoyancy of the robot can be adjusted. These small changes in buoyancy create vertical motion, which is converted to horizontal motion by the wings. This provides an extremely efficient, low power way to travel through the ocean, using long, triangle shaped trajectories. Due to the highly underactuated nature of these robots, finding trajectories from one position to another is a non-trivial task. In this project we were motivated to see if the techniques taught in 6.832 could be effectively applied to this system.

While there is significant prior work in developing models of the dynamics of these gliders [1], we found the three-dimensional models to be too complicated for the scope of this project, and instead chose to create our own planar model, with the same fundamental features and limitations as the three dimensional glider.

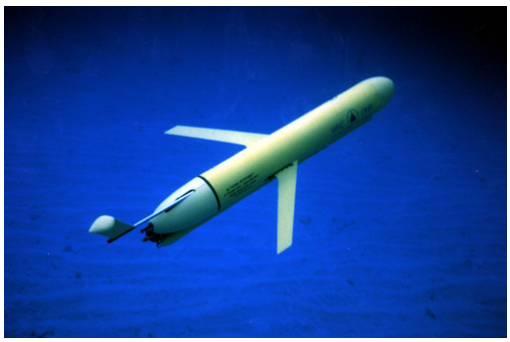

Figure 1: A SLOCUM underwater glider.

2.2 The planar model

The system analyzed in this project was a planar (two dimensional) model of the three dimensional underwater glider. In addition to the conversion to two dimensions, there were several other simplifications. The mass of the glider is modeled as two point masses of neutral buoyancy, one at the front of the glider and one at the rear, separated by a distance $2l$. To model the changes in buoyancy caused by the bladder, we allow the forward point mass to be adjusted by a control input, letting the nose of the glider become positively or negatively buoyant. This buoyancy will cause an overall force on the glider, as well as torque around the center of mass, as shown in the free body diagram below. To further simplify the model, we make the assumption that the changes in the mass of the forward mass are of small enough order that the center of mass of the glider is fixed.

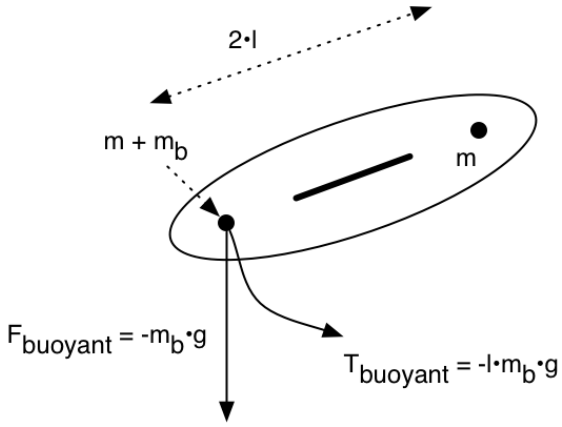

Figure 2: A free body diagram of the hull of the planar glider.

Forward forces are caused by the wing at the center of the glider, shown here as a thick line. We model this force by assuming elastic collisions with the water incident on the wing – an obviously incorrect assumption, but one that leads to a very reasonable model. As shown in the second free body diagram below, if the glider is moving along some velocity vector $\mathbf{v}$, there will be a force, normal to the angle θ of the glider (and wing), proportional to the sine of the angle of incidence of the water, $\alpha = \angle\mathbf{v} - \theta$.

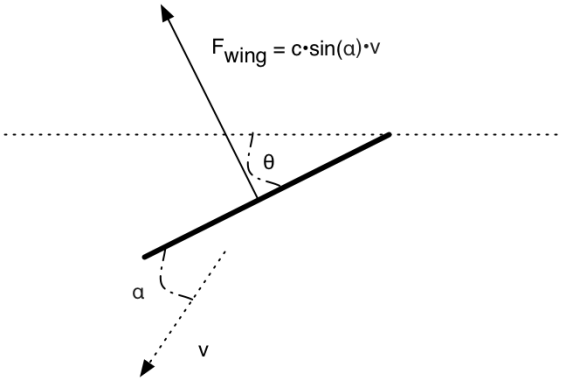

Figure 3: A free body diagram of the wing of the planar glider.

From these forces, we can derive the following system dynamics equations:

$$\ddot{x} = \frac{c \sin\alpha \sin\theta |v| - b_x \dot{x}}{2m + m_b} \quad (1)$$

$$\ddot{z} = \frac{-m_b g + c \sin\alpha \cos\theta |v| - b_z \dot{x}}{2m + m_b} \quad (2)$$

$$\ddot{\theta} = \frac{-l m_b g \cos\theta - b_r \dot{\theta}}{(2m + m_b) l^2} \quad (3)$$

$$\dot{m_b} = u \quad (4)$$

The system has a seven dimensional state vector, $\mathbf{x} = [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, m_b]^T$.

3 Methods

We found trajectory optimization alone to be incapable of finding satisfactory trajectories in most cases where the route did not already lie near the "natural trajectories" of the system. (For example, while trajectory optimization alone could find a solution along a diagonal line through x and z, it would fail to find a solution to travel horizontally along x.) We used a rapidly exploring random tree (RRT) to find a nominal trajectory to use as an initial control tape for the trajectory optimization.

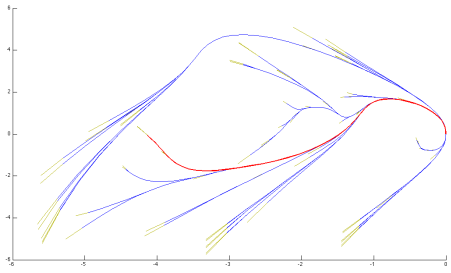

Figure 4: An example RRT figure. The "feasible nodes" have blue paths, the "reachable nodes" have yellow paths, and the red path highlights the solution.

3.1 RRT

The RRT used to find the nominal trajectory had several modifications from the basic RRT algorithm that proved necessary to make the RRT work effectively.

The first, and most important of these was the use of two trees, a "feasible" tree and a "reachable" tree, instead of just one. When a new point is added to the tree, it is added to both the feasible and the reachable tree. However, we then add two more points to the reachable tree, simulating the system forward for a brief period of time (the same period of time as is used by the `extend` operation) with minimum and maximum control input applied. When a sample is picked, its distance is compared to all of the nodes on the reachability tree. If the closest node only exists in the reachability tree, that point is used, and an `extend` operation is performed. However, if the node also exists on the feasible tree, that sample is discarded and a new target sample is picked.

With the basic RRT algorithm, it failed to find a path from $[0,0,0,0,0,0,0]^T$ to $[-4,0,*,*,*,*,*]^T$ within 300 seconds 6/10 times. When it did find a path, it took 230 ± 102 seconds to find. After modifying the RRT to use a reachable tree, the failure rate dropped to just 3/10, with an average time to solution of 111 ± 72.

Other modifications to the RRT also helped improve its ability to find feasible trajectories. Of significant importance were the way that the state space was sampled, and the distance metric used to find the closest existing node to a random sample. Rather than sampling uniformly in all seven state variables, the RRT algorithm that we used sampled randomly only from x and z, without regard for velocities or angles. Furthermore, the distance metric that was used for finding the closest point in the tree, and for evaluating the success of the `extend` only used the Euclidean distance in x and z.

While this worked acceptably, it was noticed that there were often cases where, due to the relatively uncontrollable system dynamics while in motion, that the glider would "graze" the target point, without actually reaching it. Once this

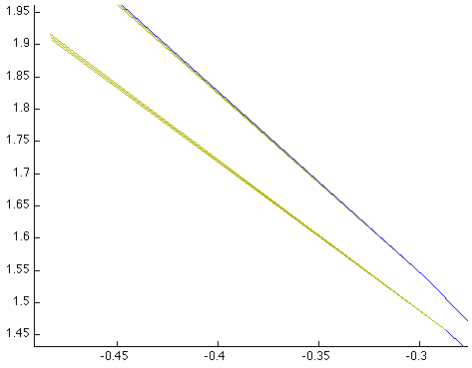

Figure 5: An example of the amount of control authority that is available while the glider is in motion. The three yellow lines shows the effect of applying maximum control input, no control input, or minimum control input.

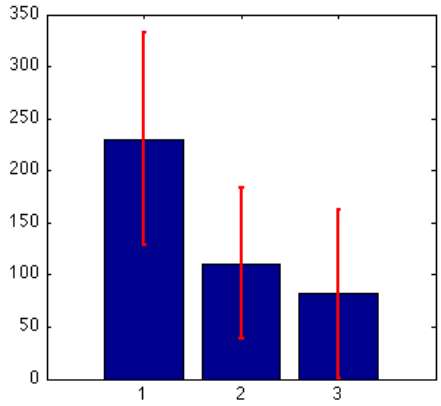

Figure 6: A graph showing the speed improvements made to the RRT algorithm for the underwater glider.

happened, when the final destination was picked as the "random" sample, the closest point to it might be a nearby, but completely inaccessible, point on the feasible tree, causing the destination to be discarded. To prevent this problem, we also added to the distance metric a comparison of θ with the angle from the glider's current location to the sample point. It was also found that weighting the θ distance slightly more than the x and z distances was helpful. The final distance metric used was as follows, where $\mathbf{x}_{xz}$ represents the $[x, z]^T$ vector of the glider, and $\mathbf{s}_{xz}$ represents the $[x, z]^T$ vector of the sample point.

$$D = (4(\angle(\mathbf{x}_{xz} - \mathbf{s}_{xz}))^2 + (x_x - s_x)^2 + (x_z - s_z)^2)^{1/2}$$

With the modified RRT described above, it failed to find a path within 300 seconds 2/10 times. When a path was found, it took 82 ± 81 seconds to find a solution.

3.2 Trajectory optimization

Initially, we intended to use the "direct method" of trajectory optimization with this system, as the output of the RRT trajectory search contains full $\mathbf{x}$ information along the entire nominal trajectory. In this method, the variables to optimize include $\mathbf{x}$ and u at every time step of the system, with the system dynamics applied as constraint equations on the optimization. However, the non-linear optimizer that we used, SNOPT, had significant numerical difficulties with this approach, for reasons that are still unknown.

Instead, we applied the "shooting method" of trajectory optimization and were able to achieve reasonable solutions. In this method, rather than applying the system dynamics as constraint equations on the non-linear optimization, the only variables to be optimized are the control tape. Gradients on the final condition constraint are computed by simulating the system forward and accumulating the gradients of the control

tape along the way.

The initial formulation of the problem was to

$$\min_{u(\cdot)} \sum_{i=0}^{N} Ru(i)^2 \tag{5}$$

subject to the following constraints:

$$x_g(N) - 0.2 < x(N) < x_g(N) + 0.2 \tag{6}$$

$$z_g(N) - 0.2 < z(N) < z_g(N) + 0.2 \tag{7}$$

with the gradients of the cost function and of the constraint computed numerically in MATLAB in the following way,

$$\frac{d\mathbf{x}(i+1)}{dh} = \frac{d\mathbf{x}(i)}{dh} + \frac{d\mathbf{f_{dyn}}(i)}{dh} \tag{8}$$

$$\frac{d\mathbf{x}(i+1)}{d\mathbf{u_g}(\cdot)} = \frac{d\mathbf{x}(i)}{d\mathbf{u_g}(\cdot)} + \frac{d\mathbf{f_{dyn}}(i)}{d\mathbf{x}} \frac{d\mathbf{x}(i)}{d\mathbf{u_g}(\cdot)} \tag{9}$$

In these equations, $\mathbf{f_{dyn}}$ represents the vector of the dynamics equations, shown below. (With the usual $\mathbf{x} = [x, z, \theta, \dot{x}, \dot{z}, \dot{\theta}, m_b]^T$). The velocity vector, $\mathbf{v}$ is equal to $[\mathbf{x}(4), \mathbf{x}(5)]^T = [\dot{x}, \dot{z}]^T$. The generalized $\mathbf{u_g} = [h, \mathbf{u}^T]^T$.

$$\mathbf{f_{dyn}} = \begin{bmatrix} h\mathbf{x}(4) \\ h\mathbf{x}(5) \\ h\mathbf{x}(6) \\ h\frac{c\sin(\angle\mathbf{v}-\mathbf{x}(3))\sin(\mathbf{x}(3))||\mathbf{v}||-b_x\mathbf{x}(4)}{2m+\mathbf{x}(7)} \\ h\frac{-\mathbf{x}(7)g+c\sin(\angle\mathbf{v}-\mathbf{x}(3))\cos(\mathbf{x}(3))||\mathbf{v}||-b_z\mathbf{x}(5)}{2m+\mathbf{x}(7)} \\ h\frac{-l\mathbf{x}(7)g\cos\mathbf{x}(3)-b_r\mathbf{x}(6)}{(2m+\mathbf{x}(7))l^2} \\ hu(i) \end{bmatrix} \tag{10}$$

Initial results of this are shown in the figure below. This method worked well when the path that was being optimized already lay along the natural trajectories of the system, as in this example. Note however that even in this case it still had difficulties finding an optimal trajectory that satisfied the final conditions by landing inside the box.

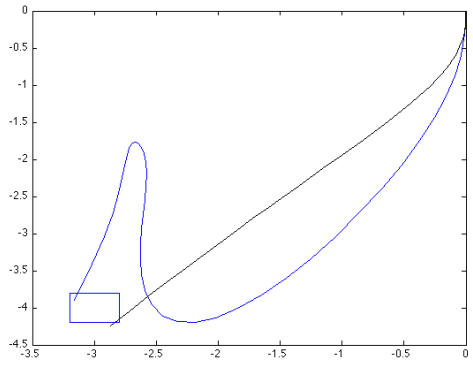

Figure 7: Initial results from the "shooting method" of trajectory optimization.

To further improve trajectory optimization, we removed the constraint equations on the final condition, and instead incorporated the final condition into the cost function that was to be minimized. This cost function is shown below.

$$F = Q((x(N) - x_g(N))^2 + (z(N) - z_g(N))^2) + \sum_{i=0}^{N} Ru(i)^2$$

It was determined through experimentation that a R/Q ratio of 4 worked well. While this may seem to be under-emphasizing the importance of the final condition, in actuality it did not, as the control values are of order $\ll 1$, so $u(i)^2$ is very small.

Results achieved with this cost function were far superior to attempting to apply the final position as a constraint. An example of this is

shown in the next figure. In blue is the feasible path found by the RRT, and in black is the optimized trajectory. To compare the "power consumption" of each of these trajectories, we use the original cost function in Equation 5. The original trajectory had a cost of 15.6, while the optimized trajectory had a cost of 2.278, a significant reduction.

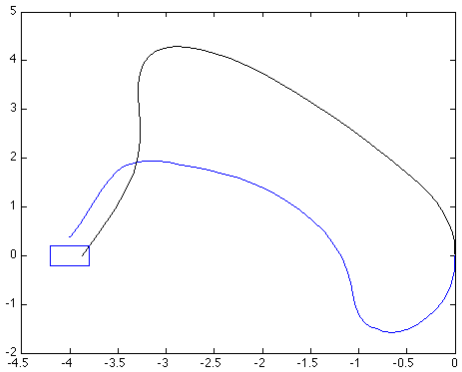

Figure 8: Successful optimization of a trajectory from $[0, 0, 0, 0, 0, 0, 0]^T$ to $[-4, 0, *, *, *, *, *]^T$, using the "shooting" method. The cost of the control tape has been reduced by a factor of 7.

4 Conclusions

The system proved generally effective at finding and optimizing trajectories for the planar glider system. The RRT was capable of finding feasible trajectories to "nearby" points in space, even when they did not lie along the natural dynamics of the system. Compiling the MATLAB program to C or something similar, as suggested by another 6.832 group, would improve the speed of the RRT search, and make finding feasible trajectories to more distant locations easier, but proved unnecessary for the scope of this project.

Once a feasible trajectory was found by the RRT, the "shooting method" of trajectory optimization proved capable of optimizing the trajectory, moving the final location closer to the target destination, while also reducing the amount of actuator power required.

Further explorations of this system could focus on analyzing the stability of the optimized trajectories. Particularly, the system loses a lot of controllability whenever θ is near $\pm\pi/2$, and it would be valuable to explore to what extent this can be avoided and controlled.

Source code, if helpful for evaluating this project, is available at `http://mit.edu/loganw/Public/6.832/`.

References

[1] Joshua G. Graver and Naomi Ehrich Leonard. Underwater glider dynamics and control. *12th International Symposium on Unmanned Untethered Submersible Technology*, 2001.