Optimal Control Strategies for an Autonomous Boat

Ajay Chandra, Richard Fu, Brandon Dimitri, and William Popovic

Abstract

Zermelo's Navigation Problem is a well-known optimal control problem which finds multiple applications in real-world maritime navigation. The typical problem considers only the case of a single control variable, steering, to optimize, however, this paper formulates the problem with a more realistic approach by accounting for the throttle input to a vessel as well as currents and drag forces present in the water. The Zermelo problem may generally be solved as a two-point-boundary-value problem, however, with the added control input and complexity to the model which we develop, this approach becomes less trivial. Therefore, this paper investigates some alternative approaches to solving the problem; first, with dynamic programming, and second, with nonlinear model predictive control. We detail the implementations of these, discuss their strengths and shortcomings, and compare the results. Finally, some additional extensions to the problem are proposed for future research.

I. Introduction

Boating using petroleum or diesel-based engines is expensive. On average, the most efficient river boats are capable of yielding a mileage of 12 miles per gallon [4]. On the surface, this sounds like a reasonable number, but for a business that runs multiple boats performing multiple river crossings per day, costs skyrocket. Furthermore, the rising prices of gasoline, which continue to increase despite decreasing crude oil prices [2], may cause these costs to go beyond control for their operators. Therefore, the need for an optimal control solution to minimize fuel for a given river crossing is justified. The paper is structured as follows – Section II covers the problem formulation and main results of both the dynamic programming and nonlinear model predictive control approaches, Section III is the conclusion, and Section IV describes the contributions of the group members.

To formulate the problem, we draw inspiration from Zermelo's Navigation Problem, which was first proposed by Ernst Zermelo in the Journal of Applied Mathematics and Mechanics in 1931 [6]:

"In an unbounded plane where the wind distribution is given by a vector field as a function of position and time, a ship moves with constant velocity relative to the surrounding air mass. How must the ship be steered in order to come from a starting point to a given goal in the shortest time?"

Zermelo [6] originally solved the problem for the general case via reformulating the problem as a Two-Point Boundary Value Problem (TPBVP) of partial differential equations. For our problem, the computation of an exact solution using this method was not feasible due to the highly nonlinear dynamics of the system and the presence of two input variables. Therefore, the comparison and evaluation of alternative solutions methods was critical to the work presented in this paper.

II. Main Body

A. Dynamics

To formulate our project dynamics, we began with the simple Zermelo's problem and modified it to have a more complex environment, a more realistic boat model, and two control inputs.

Zermelo's original problem simplifies the boat to a point mass traveling at a constant speed and neglects the current. We began expanding upon it by modeling the boat itself as a rectangular prism with an evenly distributed mass and a submerged hull depth traveling through the water. To add a realistic environmental challenge to overcome, we added a current to our river in the form of a force vector field. This vector field is parabolic with its peak in the

The authors are with Aeronautics and Astronautics Engineering, Purdue University, West Lafayette, IN, 47907 USA (e-mail: chandr73@purdue.edu, fu235@purdue.edu, bdimitri@purdue.edu, wpopovic@purdue.edu)

center of the river and approaches zero at the shores to approximate a realistic current [3]. This flow is modeled by the vector field in equation 1 which assumes a flow in the y direction with a river of length 10 m.

The vector field of the river is defined as 1 and shown in Figure 1 with $u_{max} = -5$, $x_0 = 5$, and $b = 5$.

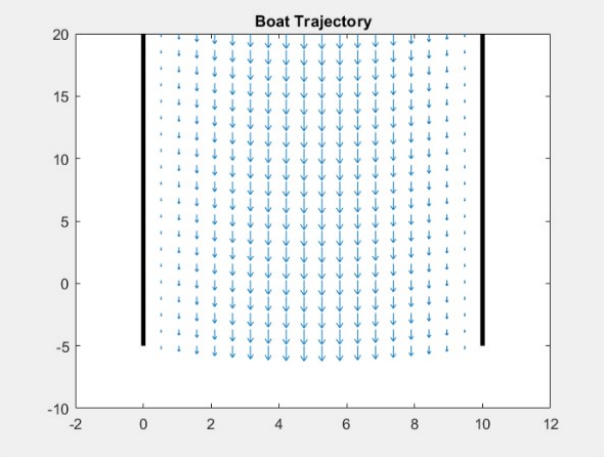

Fig. 1: Plot of the parabolic river current flowing downstream.

The parabolic flow vector field is given by

$$F(x, y) = (u_{max} \cdot (1 + (\frac{x - x_0}{R})^2))\hat{j} \tag{1}$$

where u_{max} is the maximum flow velocity and R is half the width of the river.

To provide coordinates for the boat, we defined an (x, y) coordinate system from the perspective of looking down at the boat from above as shown in Figure 2. The boundary conditions for the boat were defined by requiring it to begin its voyage from a stationary start at a known initial location and reach a known final destination. To simplify the problem, the initial location of the boat was defined as the origin of the coordinate system and the shores of the river are parallel to the y-axis.

The boundary conditions for the boat were defined by requiring it to begin its voyage from a stationary start on the left shore (as shown in Figure 1) and reach a known final destination on the other shore.

We further complicated our control by making throttle and rudder direction components of our control vector. Throttle control, displayed as the thrust force T, allowed our speed and acceleration to be variable. Rudder control is the angle at which T is applied, measured from the vector of the boat's velocity. While the heading and its rate of change are still components of our state vector as in Zermelo's problem, they are now functions of more complex dynamics. The throttle and rudder angle are applied to the back of the boat instead of at the center of mass to more realistically model the dynamics of a boat.

We added to the dynamics by including the drag forces from the water. Modeling the drag force of the water better models the forces and disturbances present by impacting our state equation. To model the more complicated shape of the boat and placement of the control, the calculations required a value for moment of inertia. For this, we used the shape of the boat in the (x, y) plane, a rectangle, as shown below in Figure 2 and implemented its properties in equation 2.

Moment of Inertia

$$I = \frac{m(w^2 + l^2)}{12} \tag{2}$$

The drag forces were modeled for each direction using equation 3. The area values are the projected area in the given direction and were calculated using equation 4. The density of fresh water ρ was entered as 1000 kg/m^3 and coefficient of drag C_d, approximated as being the same in both the x and y directions, was arbitrarily chosen as 0.1.

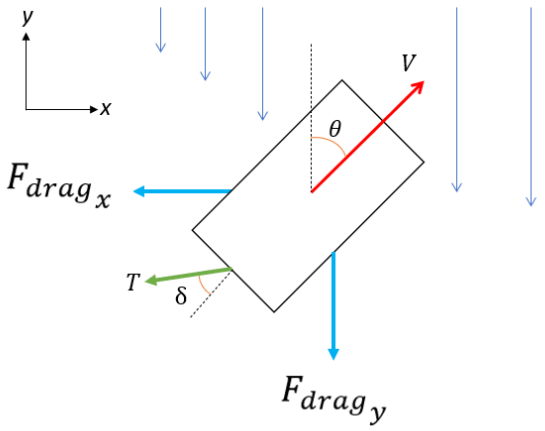

Fig. 2: Boat Model

$$F_{drag,x} = \frac{1}{2}\rho A_x C_d \dot{x}^2 \quad \text{and} \quad F_{drag,x} = \frac{1}{2}\rho A_y C_d \dot{y}^2 \tag{3}$$

$$A_y(\theta) = wh\cos^2(\theta) + 2lhwh\sin^2(\theta) \quad \text{and} \quad A_x(\theta) = A_y(\theta - \frac{\pi}{2}) \tag{4}$$

Using the properties of the boat and equations 1, 2, 3, and 4, we derived the equations of motion 5.

$$\ddot{x} = \frac{1}{m}(T\sin(\theta+\delta)+F_{drag,x}),\ \ \ddot{y} = \frac{1}{m}(T\cos(\theta+\delta)+F_{drag,y}),\ \ \ddot{\theta} = \frac{1}{I}(-Tl\sin(\delta)+d\cdot(F_{drag,x}\cos(\theta)-F_{drag,y}\sin(\theta))) \tag{5}$$

With this information, we assembled a state space representation, inspired by 1 of the dynamics of the boat shown in 6.

$$\dot{\vec{x}} = f(\vec{x}(t),\vec{u}(t),t) = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \ddot{x}\\ \ddot{y}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}\sqrt{\dot{x}^2+\dot{y}^2}\cos\theta\\ \sqrt{\dot{x}^2+\dot{y}^2}\sin\theta\\ \arctan(\frac{\dot{y}}{\dot{x}})\\ \frac{1}{m}(T\sin(\theta+\delta)+F_{drag,x})\\ \frac{1}{m}(T\cos(\theta+\delta)+F_{drag,y})\\ \frac{1}{I}(-Tl\sin\delta + d\times(F_{drag,x}\cos(\theta)-F_{drag,y}\sin(\theta)))\end{bmatrix} \tag{6}$$

With our dynamics fully modeled, we began implementing our control techniques.

B. Dynamic Programming

The first control method implemented was Dynamic Programming. We designed our own controller curtailed to our project requirements. First, we discretized our three control options for T and δ as shown in equation 7. These corresponded to a bang-bang controller implementation in which the controller can choose from a maximum, minimum, or zero values for T and δ.

$$\delta \in [-50°, 0, 50°] \quad \text{and} \quad T \in [-10, 0, 10] \tag{7}$$

We then set our initial state x_0 as x_i, our current position. Then, for all nine possible control input combinations, we iterate the dynamics of the system for a given number of steps. The control option with the lowest resulting cost

function is taken for another given number of steps. For this controller, our cost function was only the distance to the target state x_f. So, the controller would always choose the control strategy that ended with (x, y) coordinates closest to the desired x_f and y_f. After the control has been applied and the model propagated, the new state becomes our new x_i and the process repeats. For our implementation, the calculations of the control horizon and the steps between control changes were both iterated for five time steps since fewer steps were less responsive to disturbances in the system of a change in the initial condition or end points, and more steps caused much greater computational times due to the exponential increase in the computations required with an additional time step. For instance, the controller requires 59,000 computations to take five time steps, but if we add a sixth step, the computations required increase almost tenfold to over 500,000.

With this methodology implemented, we obtained the initial results shown in Figures 3 and 4.

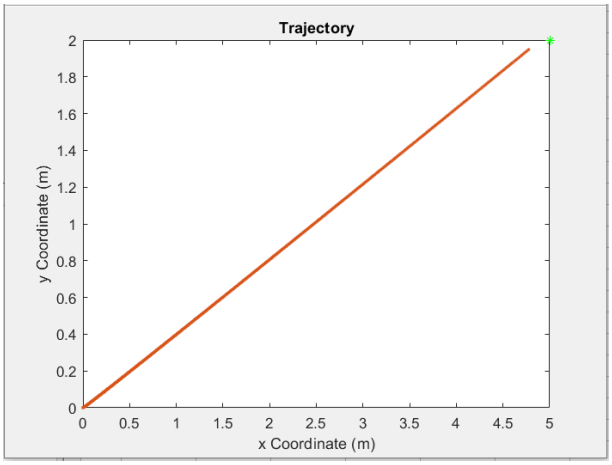

Fig. 3: Initial Dynamic Programming Trajectory

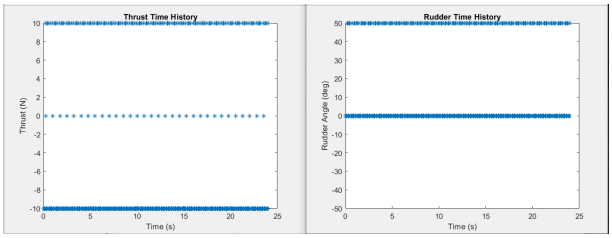

Fig. 4: Initial Dynamic Programming Control Input

In this initial implementation, we kept the river current at zero as we got the controller working. As expected, the optimal path was a straight-line approach. Since the rudder options are relatively coarse compared to the controls on a real boat, it can be seen to switch from 0 to 50° along the time history as the boat zigzags across a direct trajectory.

From these initial results, we saw areas where the controller could be further improved. First, the controller was computationally expensive. It took a significant amount of time for even the simplest version of this problem to be solved. This is why we only have three options for both of our control inputs. Another potential strategy to reduce computation time, increasing the time step, led to the controller being unable to converge to our final position within our specified tolerance. Instead, the controller would jump back and forth across our goal position with no end. This issue was solved by lowering the size of the time steps relative to the error tolerance, but this led to longer computation times.

The time step issue was addressed by making it variable. Knowing that the algorithm does not need to be accurate in the middle of our problem, we choose to have larger time steps and save computation power. Only as we approach our goal position do we need to begin decreasing the duration of each time step to gain accuracy. To this end, we set certain distances from the goal at which to lower the duration of each time step. These numbers were chosen

by examining where the controller failed. It is worth noting that this strategy proved to have diminishing returns. To get within an error of 0.25 length units of our goal, we needed a time step of 0.003 seconds. To get within 0.1 length units, we needed a time step of 0.00005 seconds. Reducing our error by a little more than half required lowering our time step by multiple orders of magnitude, greatly increasing the computational complexity. Figure 4 shows this modified controller and how much closer we get to our goal position. It is also worth noting how the points in the middle are sparse compared to near the end where the points become very tight, as seen in Figure 5.

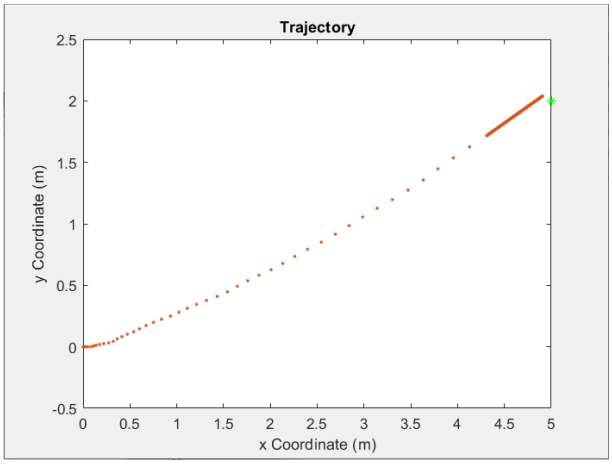

Fig. 5: Dynamic Programming Trajectory with Variable Time Step

We also began examining how this controller works in the presence of the perturbation of the river current. Here, we found the biggest downfall of this controller strategy. We found that the controller can compensate for small perturbations such as in the case shown in Figures 6 and 7, but under larger river currents, the boat is swept downstream.

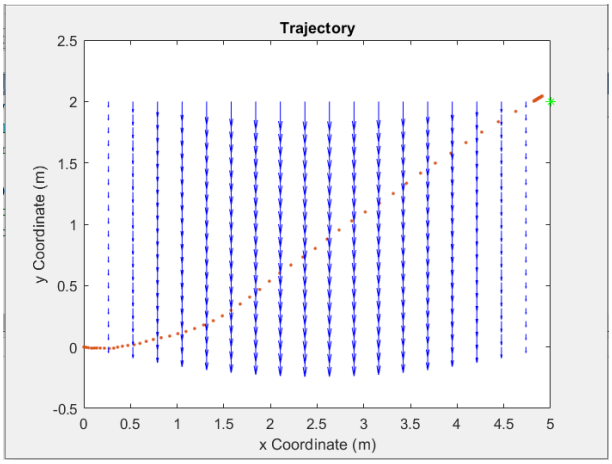

Fig. 6: Successful Dynamic Control Trajectory Under Small Current

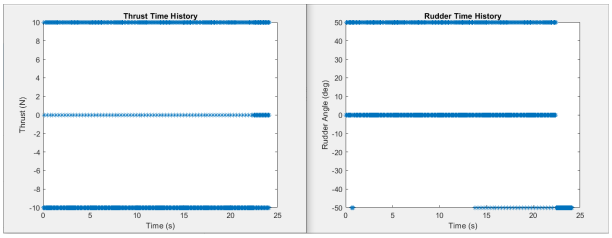

Fig. 7: Successful Dynamic Programming Control Inputs Under Small Current

It is worth noting that in the presence of a current, both rudder directions are used. This makes sense as the boat needs to account for the turning force for the river current at different parts of the stream. This successful run was under a max current of only 0.001 m/s at the center of the river.

The controller could be further improved in a few ways. Instead of only reducing the size of the time step as we approach our end goal, we could also reduce the number of time steps that the boat goes before the control is re-examined. This would help reduce our computation time and lower our error. We could also further reduce our error time by further optimizing our code by limiting our use of loops. Finally, we could make the controller more robust against the river current by adding an estimator for the dynamics into the controller.

Though these options offered some improvement during development, we decided our efforts would be better spent examining alternative control methods that would be more robust against disturbance.

C. Nonlinear Model Predictive Control

The limitations present with dynamic programming led us to consider alternative options to find an optimal control sequence to the problem. The main alternative used in this case was the MATLAB nonlinear model predictive controller (nlmpc). Since the dynamics of the problem were nonlinear due to the trigonometric terms in the state equations, and since linearization of the state equation was difficult because of the lack of equilibrium points, this was seen as the best option available. The MATLAB nonlinear model predictive controller is capable of solving nonlinear state equations and cost equations. It searches over specified control and prediction horizons to try and find the optimal control trajectory based on specified reference states and costs.

To configure the nlmpc controller to this specific problem, we created a nlmpc object in MATLAB with 6 states, two outputs (x and y position), and two input variables (controls T and δ). We first tested the controller along a river with no current, which gave results as expected of a trajectory directly to the goal point, which was set at the point (10,10). The origin of the boat was set at (0,0) with zero initial conditions. We then tested the controller by applying a current to the river, which was modelled as a parabolic flow across the length of the river with a max velocity of 2 m/s in the positive y direction. The cost function used in the algorithm is given in Equation 8. Here, $\hat{x}$ represents the difference between the current position and goal position, v is the vessel velocity, and u is the control input, which is the vector $[T, \delta]^T$. Q, G, and R are each weighting terms to specify the relative importance of each of the corresponding variables. By varying the values of these weights, the optimal trajectories and control inputs changed. For instance, in the case of a minimum fuel trajectory, the input R could be increased to place more of a penalty on fuel usage from using the throttle. Alternatively, a minimum distance trajectory placed greater weight on Q. G was used as a weight on velocity, since a soft landing with low velocity was desirable.

Cost

$$J = \int_{t_0}^{t_f} \hat{x}' Q \hat{x} + v' G v + u' R u \, dt \tag{8}$$

Stated fully with the cost function from Equation 8; the problem becomes

$$J = \min_{x(t),u(t)} \int_{t_0}^{t_f} \hat{x}' Q \hat{x} + v' G v + u' R u \, dt \tag{9}$$

$$
\begin{aligned}
&s.t. \\
&\dot{x} = f(x, u) \\
&x_0 = [x_0, y_0, \theta_0, \dot{x_0}, \dot{y_0}, \dot{z_0}] \\
&x_1(t_f) = x_f \\
&x_2(t_f) = y_f \\
&[x_4(t_f), x_5(t_f), x_6(t_f)] = \vec{0} \\
&x_l \leq x_1 \leq x_r
\end{aligned}
$$

Formulating the problem as this indicates that we are looking for the optimal control input $u(t)$ and state history $x(t)$ (vector notation dropped for convenience) to minimize the established cost function while satisfying the system dynamics, a set of given initial conditions, end conditions of the goal position and zero velocity, and constraints on the position of the boat so that it remains within the domain of the river boundaries, x_l on the left and x_r on the right.

One issue present with the implementation of Equation 8 as the cost function was shown in Figure 8a, in which the control input penalty became disproportionately large as the vessel approached the goal point since the distance error became almost negligible. Therefore, the trajectory focused more on a low control input trajectory, rather than completing the path to the shore as would be desired. To resolve this issue, new weights were activated when the vessel got within a set distance of the goal position which placed a larger weight on the distance and velocity errors to facilitate an accurate and soft landing. Figure 8b shows the result obtained with the variable weights.

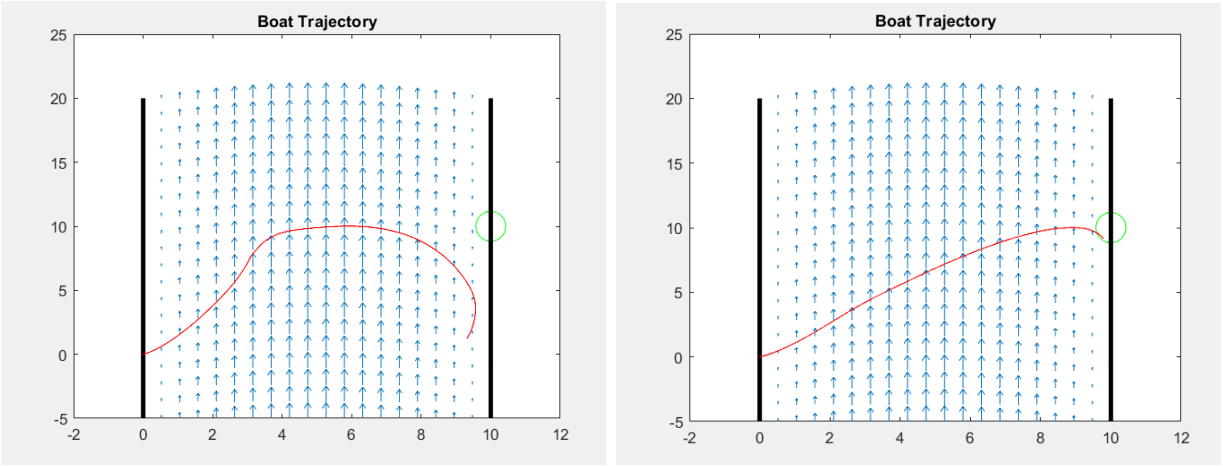

Fig. 8: a) Fixed weights throughout the trajectory b) Variable weights activated at the end of the trajectory

With the cost function now fully developed, we next tested different weights to observe the impact on trajectories and the characteristics of the solutions such as the total trajectory time, error from the goal point, and the total control cost, which was taken by calculating a Riemann sum of throttle input over each time step. The only weight varied throughout these trials was the weight on the throttle input. The rest of the weights kept the same values, which are listed in Table I. Note that the Q and R weights are represented as 2x2 matrices because they act on the vectors $[x, y]^T$ for the position and $[T, \delta]^T$ for the control inputs. * Indicates that this value was the value varied over trials as the throttle, and therefore the fuel, weight. The results of some of these trials are shown in Figure 9 and Table II.

As expected, and as supported by Table II, increasing the weight of the throttle cost in the R matrix has the effect of reducing the overall cost of the throttle input at the expense of a greater error to the goal point and a longer time to complete the trajectory.

TABLE I: Cost Function Weightings used in NLMPC Controller

Weight	Value	Useage Conditions
Q	$\begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$	Distance to goal > 2
	$\begin{bmatrix} 200 & 0 \\ 0 & 200 \end{bmatrix}$	Distance to goal ≤ 2
G	0	Distance to goal > 2
	100	Distance to goal ≤ 2
R	$\begin{bmatrix} * & 0 \\ 0 & 0.1 \end{bmatrix}$	Always

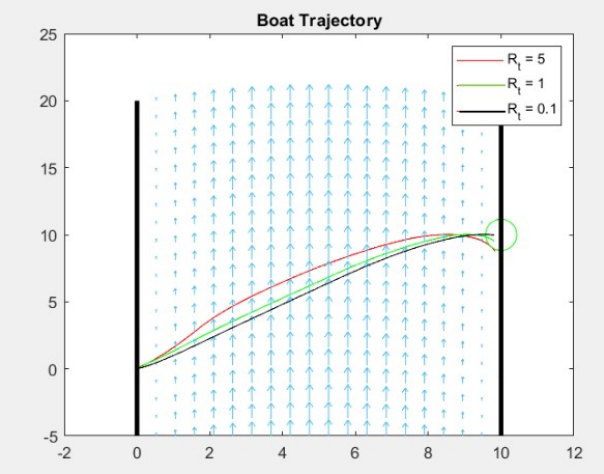

Fig. 9: Comparison of River Crossing Trajectories by Varying Cost Function Weights

The above tests were done by placing a current of 2 m/s in the positive y direction. The algorithm was also responsive to changes in the current and calculated the optimal trajectory accordingly based on the current characteristics. Figure 10 shows the computed optimal trajectories for three additional current scenarios; a current going against the direction of the vessel, a counterclockwise vortex, and a clockwise vortex.

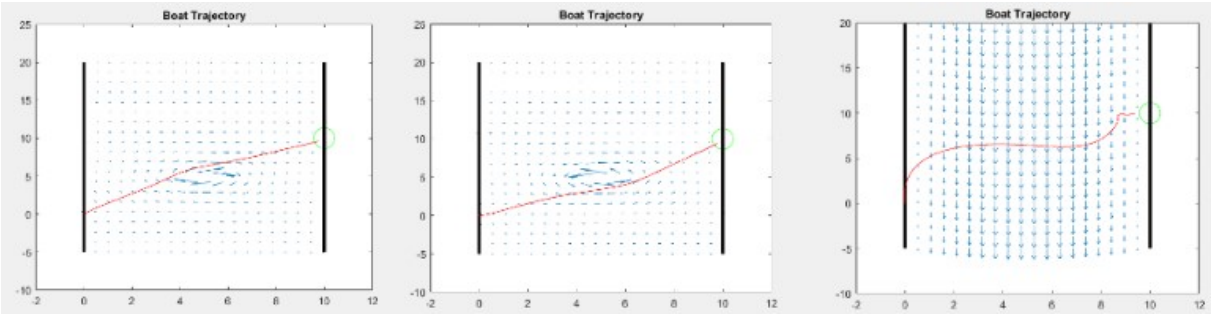

Fig. 10: Optimal Trajectories from NLMPC Controller Across Varying Currents for:
a) Clockwise Vortex b) Counterclockwise Vortec c) Parbolic Flow Against Boat Travel

Clearly, these trajectories behave as expected for the optimal conditions as they take advantage of the velocity gains provided by the vortex in (a) and (b), and (c) leverages the weakest parts of the current at the edges of the river for the majority of its travel in the +y direction where the current is weaker.

To further explore the effectiveness of the nlmpc controller in a vortex flow, the simulations run in Figure 9 were

TABLE II: Trajectory Characteristics of Various Weightings

Throttle Weight (R_t)	Throttle Cost	End-point Error (m)	Trajectory Time (s)
5	554	1.24	13.6
1	647	0.49	10.4
0.1	691	0.18	8.8

repeated under a clockwise counterclockwise vortex, which presented an interesting challenge for the vessel as it meant fighting the current along the trajectory. As a result of the current acting against the vessel, too high of a weight on throttle input resulted in the end point not being reached, as shown in Table III. The control inputs calculated for the trajectories in Figure 11 are shown in 12. The impact of the throttle weighting term can clearly be seen here with the greater throttle input in the early stages of the trip for the lower weights and the corresponding higher velocities achieved. It can also be seen from the trajectories that the vessel makes more of an effort to avoid the regions of highest current resistance near the center of the vortex with the higher cost terms, but takes a more direct path through these stronger flows at the cost of additional throttle input when the throttle cost weight is lower.

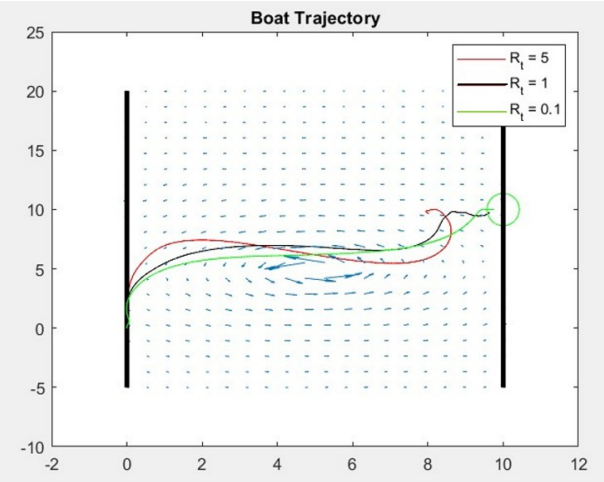

Fig. 11: Trajectories Taken in Counterclockwise Vortex Flow

TABLE III: Trajectory Characteristics of Various Weightings

Throttle Weight (R_t)	Throttle Cost	End-point Error (m)	Trajectory Time (s)
5	696	1.99	32*
1	1017	0.39	32
0.1	1245	0.23	16

* With the weight of 5, the vessel failed to reach the goal point within the defined maximum time of 32 seconds.

D. Comparison Between Optimal Control Strategies

The two control strategies presented varying challenges and benefits.

Regarding computational time expenses, we noted that dynamic programming required a significantly longer run time (6 minutes using a necessary variable time step to avoid solution divergence) than nonlinear model predictive control ($<$ 1 minute using fixed time steps). Furthermore, we note that dynamic programming is memory inefficient to implement in-situ. This is because many of the computed states become stored in memory without serving any

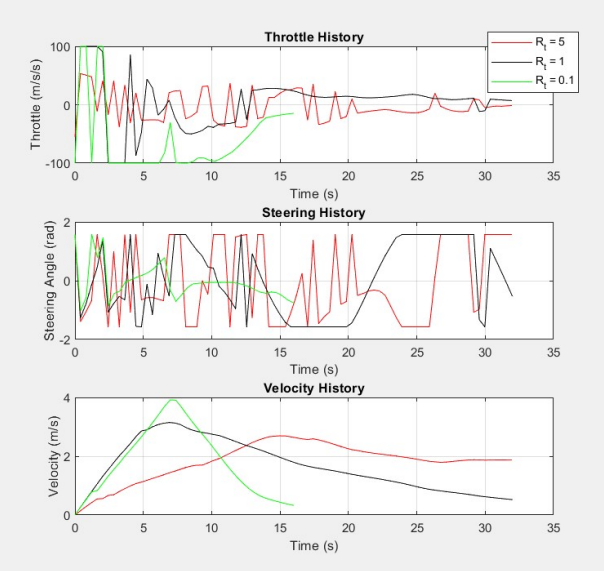

Fig. 12: Control Inputs Required for Vortex Trajectories

purpose. This may limit the kind of computing resources that may be present onboard a boat, which is already severely restricted by mass limitations.

In addition, the usage of dynamic programming necessitated the implementation of a variable time step, decreasing as the state approaches the desired final state due to the buildup of solver error.

Nonlinear model predictive control uses a searching algorithm similar to that used in dynamic programming. However, it leverages much more efficient methods than those utilized in the dynamic programming algorithm presented here. Computational expenses can still be large if large prediction and control horizons are used and it may be necessary to run several simulations with varying constraints and cost functions in order to obtain realistic solutions.

E. Future Work

The work presented in this paper has several future extensions which could be explored. Since the MATLAB nonlinear model predictive control model has so many parameters which can be set, it would be easy to vary these in order to obtain a more accurate model, however, doing so also increases the computational complexity and the chance that the algorithm will fail to find a solution as more constraints are imposed. The work on this problem also considered only the optimal control problem, however, in a real system, it is unlikely that the full state will be measured. Therefore, this problem could be extended with the addition of a Kalman filter to provide an optimal estimate of the states under additional disturbances and noise. Additional types of flow could also be tested; for instance, random flow throughout the domain. The algorithm presented tends to work best at shorter distances and with shorter prediction and control horizons to reduce the overall computational complexity, however, the scale of the problem could also be changed to observe how the algorithm performs over longer distances and times.

III. Conclusion

This research sought to expand on the traditional Zermelo's navigational problem with a more accurate model of a boat capable of providing its own thrust. The nonlinearity of the problem and the multiple control inputs indicated the need for solvers beyond those traditionally used to solve more basic optimization problems. The two techniques primarily tested for this problem were dynamic programming and MATLAB's nonlinear model predictive control toolbox.

Dynamic programming was well suited for this problem as it could evaluate many possible outcomes before making the optimal decision, however, this was also a major drawback since it meant greater computational time which only

scaled exponentially as the prediction and control horizons were increased. Additionally, the dynamic programming algorithm was only feasible over a limited range of control inputs making it best for situations in which a bang-bang controller is optimal. The input range could have been expanded, but again, this would have only led to exponential increases in computational time.

Nonlinear model predictive control proved to be a much better approach to solve this problem. By setting applicable constraints on the inputs and states in addition to developing a cost function to incorporate the elements of the problem to be optimized, much greater control was available over the calculated optimal trajectory and controls.

The two techniques used in this paper represent only a small number of the total number of optimization algorithms which exist today. Therefore, it is likely that even better optimization algorithms exist compared to those used. However, we have shown that both these options can find solutions, although their robustness may be limited as disturbances become greater or the model becomes more complex.

By solving the Zermelo problem on a more realistic boat model, we can better determine the optimal path for a boat crossing a river under some known current. The optimal path in this case may be different depending on what variable is considered most important, such as to minimize fuel consumption, time, distance, etc. This information would be beneficial to boat or ferry companies that may want to maximize their profits by increasing the total number of trips made (as is the case of a minimum time problem) or, as was more relevant with this cost function used in this paper, reducing fuel costs.

IV. Contributions of Individual Members

Our group consists of Ajay Chandra, Brandon Dimitri, William Popovic, Richard Yuan Fu. Our contributions are as follows. All team members contributed to the drafting of the report and the presentation slides.

Ajay Chandra contributed to the formulation of the dynamics and debugged issues with the earlier iterations of the solutions to the Non-Linear Model Predictive Control and Dynamic Programming Problems. He also contributed to the further optimizations of these early solutions, and investigating other available control options, including the analytic derivation of a controller using TPBVP.

Brandon Dimitri was the main contributor to the Non-Linear Model Predictive Control and Dynamic Programming problem formulation. He is also responsible for earlier iterations of the solutions to these problems. Furthermore, he formulated several means by which the existing control problems can be added to in terms of complexity. He developed the basis of the MATLAB code for dynamic programming, and MATLAB code to implement Non-Linear Model Predictive Control. He generated the tests mentioned previously in the paper by modifying the cost function weights and experimenting with different flow fields in the domain. He also formulated the cost function when programming the nlmpc code to search for the optimal conditions. Early on in the project he helped in the development and coding of the problem dynamics and found the primary paper (1) which was used to motivate the problem statement and derive the dynamics used. Finally, he contributed to the organization and formatting of the final paper as displayed here.

William Popovich contributed to the formulation of the Dynamic Programming problem and contributed to the debugging of the same. He also contributed to further optimizing the earlier iterations of the aforementioned solutions. He also investigated other available control options, including the analytic derivation of costates. He wrote the dynamics and Dynamic Programing sections of this paper.

Richard Fu contributed to the formulation of the dynamics of the boat, the development of the vector field to model the river current, and investigated the formulation of the problem as a TPBVP with an analytic derivation for optimal control.

REFERENCES

[1] Chen, E., Perper, I. (2020, May 30). Trajectory Optimization for an Autonomous Boat.

[2] Dam, A. V. (2022, April 11). Why gasoline prices remain high even as crude oil prices fall. The Washington Post. Retrieved April 24, 2022, from https://www.washingtonpost.com/business/2022/04/11/gasoline-prices-crude-prices/

[3] Fowler, M. Calculating viscous flow: Velocity profiles in rivers and pipes. River Viscosity. Retrieved April 24, 2022, from https://galileo.phys.virginia.edu/classes/152.mf1i.spring02/RiverViscosity.htm

[4] Mahidhar. (2021, June 7). Average boat mileages: With 50 examples of different boat models. Boating Valley. Retrieved April 24, 2022, from https://www.boatingvalley.com/average-boat-mileages-with-50-examples-of-different-boat-models

[5] Y. Sakawa. "Trajectory planning of a free-flying robot by using the optimal control." Optimal Control Applications and Methods, Vol. 20, 1999, pp. 235-248.

[6] Zermelo, E. (1931). Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung. ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik, 11(2), 114-124.