Trajectory Optimization for an Autonomous Boat

Eric Chen
CSAIL, MIT
ericrc@mit.edu

Isaac Perper
CSAIL, MIT
iperper@mit.edu

***Abstract*—For our final project, we implemented trajectory optimization for an autonomous motor boat using Drake. We derived the dynamics for a motorboat navigating in a body of water with varying current flows. We implemented trajectory optimization on our system using two different solvers - SNOPT and IPOPT. We show a variety of successful optimizations in a range of current flow scenarios. We demonstrate that the trajectories hold in the presence of both static and moving obstacles, and we discuss some of the challenges encountered with the optimization.**

***Index Terms*—Underactuated Robotics, Trajectory Optimization, Optimization Robustness**

I. Introduction

Some of the most interesting problems we have covered in 6.832 involve real world systems. While we have seen many interesting scenarios involving complex robots, we have not seen many problems in which the surrounding environment itself contributes additional constraints to the complexity of the control problem. We investigate an autonomous motorboat in a current field.

Boats have interesting interactions with water via drag, which is often another variable state in the system. Current fields that induce drag can become quite complex and powerful even at the macro scale of oceans. Additionally, current fields provide an opportunity for the boat to utilize the environments dynamics to its advantage while traversing a trajectory. These factors make the naval domain an interesting space on which study the behavior of trajectory optimization methods.

II. System Dynamics

The dynamics for our boat account for the x and y direction forces and θ direction torques from the vehicle inputs (F_τ and θ_τ) and drag from the water. Ship dynamics have been well researched in the field of naval engineering [1]–[3], although we do use several assumptions and approximations to simplify the dynamics for our problem. Figure 1 can be referenced for coordinate conventions and boat dimensions. We chose to make the boat symmetrical along the vertical and horizontal axis to simplify the dynamics, while maintaining interesting drag interactions. Our inputs affect second-order derivatives, yielding the state representation in Eq. 1.

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta_h \\ \dot{x} \\ \dot{y} \\ \dot{\theta}_h \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} F_\tau \\ \theta_\tau \end{bmatrix} \tag{1}$$

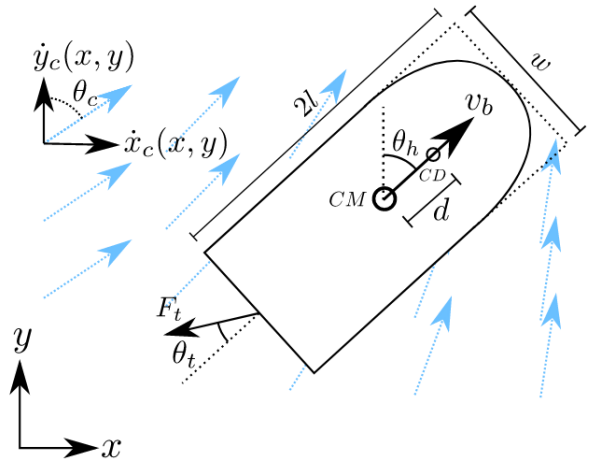

Fig. 1. The boat has a heading θ_h relative to the y axis and produces a thrust F_τ that is applied at an angle θ_t relative to the heading of the boat. The current is modeled as a vector field that returns two velocity components $\dot{x}_c$ and $\dot{y}_c$ at each (x, y) point.

The vertical and horizontal forces on the robot are a combination of the thrust inputs, and the drag forces. The drag forces are defined by the signed difference between the boat's velocity and the velocity of the current flow at the boat's center of mass:

$$\sum F_x = F_\tau \sin(\theta_h + \theta_\tau) + F_{drag_x} = m\ddot{x} \tag{2}$$

$$\sum F_y = F_\tau \cos(\theta_h + \theta_\tau) + F_{drag_y} = m\ddot{y} \tag{3}$$

$$\begin{aligned} \sum \tau \; &= -F_\tau l \sin\theta_\tau \\ &\quad + d(F_{drag_x} \cos\theta_h - F_{drag_y} \sin\theta_h) \\ &\quad + \tau_\omega = I\ddot{\theta}_h \end{aligned} \tag{4}$$

These are the equations that allow us to extract the dynamics of our boat: $\dot{\boldsymbol{x}} = f(\boldsymbol{x_t}, \boldsymbol{u_t})$.

Assumptions:

- The current of the water ($\dot{x}_c$ and $\dot{y}_c$) does not change significantly within the area of the boat at a given timestep. For the torque on the boat, we modeled the shape as a rectangle.
- The drag coefficients C and drag cross-sectional area A are a function of boat heading, such that the dynamics incorporate the forward direction of the boat being more streamlined, and the side direction of the boat having more drag. These are defined by a weighted sum of the

forward and horizontal coefficients to interpolate between them smoothly for all headings.
- The center of drag is above the center of mass, allowing drag to act as a restoring force that aligns the boat in the direction of the current.

Under these assumptions, we use $I = \frac{m(w^2+l^2)}{12}$ as the moment of the inertia. The surface areas exposed to drag in the horizontal and vertical directions vary based on the boats heading. They can be expressed in the following manner where $2l$ and w are the length and width of the boat, and h is the submerged distance of the hull in the water:

$$A_y(\theta_h) = wh\cos^2(\theta_h) + 2lh\sin^2(\theta_h)$$
$$A_x(\theta_h) = A_y(\theta_h - \pi/2)$$

These areas allow us to define the drag coefficients in the vertical and horizontal directions as a function of the boats heading in terms of the drag coefficient of the boat travelling forwards and backwards C_{front}, and travelling sideways C_{side}:

$$C_y(\theta_h) = C_{front}\cos^2(\theta_h) + C_{side}\sin^2(\theta_h)$$
$$C_x(\theta_h) = C_y(\theta_h - \pi/2)$$

With the drag coefficients defined, we obtain the drag force as follows:

$$F_{drag_x} = \frac{1}{2}\rho_w C_x(\theta_h)A_x(\theta_h)|\dot{x} - \dot{x}_c|(\dot{x} - \dot{x}_c)$$
$$F_{drag_y} = \frac{1}{2}\rho_w C_y(\theta_h)A_y(\theta_h)|\dot{y} - \dot{y}_c|(\dot{y} - \dot{y}_c)$$

In order to compute Eq. 2, Eq. 3 and Eq. 4, the last component is τ_ω, which is the drag on the boat due to its rotation in the water $\dot{\theta}_h$. This drag force acts as a restoring torque on the rotating boat, causing it to align with the current field if no motor torque is applied as seen in Fig. 2. We can obtain a decent approximation of the restoring force by integrating the drag along the side of boat:

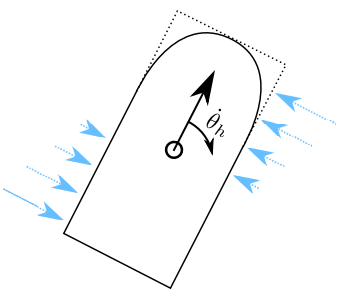

Fig. 2. The drag force from boat rotation stabilizes $\dot{\theta}_h$ generated from input torques.

$$d\tau = \frac{1}{2}\rho_w C_{side} v^2 dA \approx -\frac{1}{2}\rho_w C_{side}|\dot{\theta}_h|\dot{\theta}_h h r^2 dr$$
$$\tau_\omega = \int_{-l}^{l} -\frac{1}{2}\rho_w C_{side}|\dot{\theta}_h|\dot{\theta}_h h r^2 dr$$
$$\tau_\omega = -\frac{1}{4}\rho_w C_{side} h l^4 |\dot{\theta}_h|\dot{\theta}_h \quad (5)$$

A. Result Parameters

For all the experiments we ran, we used the following numerical values presented in Table I for all the constant parameters we just defined.

TABLE I
THE PARAMETERS USED FOR GENERATING OUR RESULTS.

Parameter	Value
m	15000 kg
l	8 m
w	5 m
h	2 m
d	7 m
C_{front}	0.05
C_{side}	0.4

The distance from the center of mass (CM) to the center of drag (CD) is denoted as d and has a large impact on the effect of torque from drag, which can be seen in Eq. 4. To make the dynamics realistic in simulation, d was set such that a current would appropriately turn the boat downstream if started from a standstill. Additionally, boat mass was increased until the acceleration due to drag was realistically noticeable in simulation.

III. Simulation

The simulator used is implemented using the Drake library [4]. Specifically, the SymbolicVectorSystem class is used to define the dynamics. Then, a DiagramBuilder is used to connect the system state with the input torques. And finally, the system is simulated using the Simulator class. An example of the boats natural trajectory with no motor torque applied can be seen in Fig. 3. The rectangle represents the boats hull, and the arrow represents the boat's heading. Note how the current drag rotates the boats heading, and how this change decreases as the boat's velocity matches the currents.

Varying the constant parameters resulted in noticeable changes in the boats performance in simulation. While the dynamics derived here are by no means fully realistic, we think they still generate believable interactions in simulation and present a good basis on which to study trajectory optimization.

IV. Optimization Formulation

Trajectory optimization takes in a desired starting state x_0 and a target state x_N, and uses a nonlinear solver to return a locally optimal trajectory $\boldsymbol{X} = \{\bar{\boldsymbol{x}}_\mathbf{0}, \bar{\boldsymbol{x}}_\mathbf{1}, ..., \bar{\boldsymbol{x}}_\mathbf{N}\}$ and $\boldsymbol{U} = \{\boldsymbol{u_0}, \boldsymbol{u_1}, ..., \boldsymbol{u_{N-1}}\}$. In order to accomplish this, the solver is

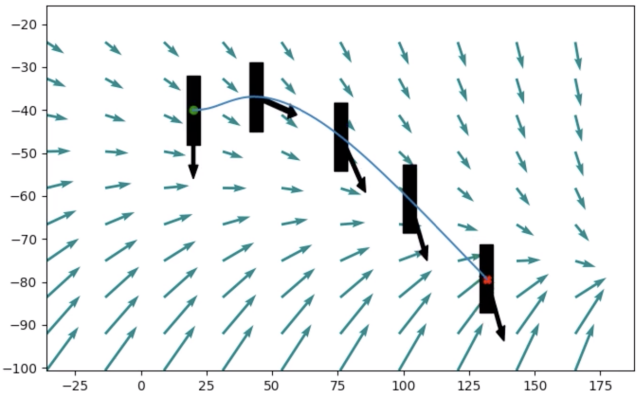

Fig. 3. The boat's natural trajectory in a whirlpool like current field, with no motor torque applied. The green dot denotes the starting location, and the red "x" denotes the location after 50 seconds. The boats heading is represented by an arrow, while the rectangular "hull" tracks the boats position.

provided with an optimization problem defined in terms of constraints on the state and torque.

In this case, the first set of constraints we apply force the solver to generate a trajectory whose starting state $\bar{\boldsymbol{x}}_{\boldsymbol{0}}$ and ending state $\bar{\boldsymbol{x}}_{\boldsymbol{N}}$ matches the desired start and goal states $\boldsymbol{x_0}$ and $\boldsymbol{x_N}$:

$$\boldsymbol{x_0} - \bar{\boldsymbol{x}}_{\boldsymbol{0}} = 0$$
$$\boldsymbol{x_N} - \bar{\boldsymbol{x}}_{\boldsymbol{N}} = 0$$

We then enforce the dynamics at each time step. We discretize our continuous dynamics $\dot{\boldsymbol{x}} = f(\boldsymbol{x_t}, \boldsymbol{u_t})$ using forward Euler integration and the duration of each timestep Δt:

$$\bar{\boldsymbol{x}}_{\boldsymbol{t+1}} - \bar{\boldsymbol{x}}_{\boldsymbol{t}} - f(\bar{\boldsymbol{x}}_{\boldsymbol{t}}, \boldsymbol{u_t})\Delta t = 0$$

We explored backward Euler integration as well, but that integration schema performed worse in our tests, despite being more stable. To generate realistic trajectories, we impose a constraint on the magnitude of the torque force and how far the torque angle can swing. Note that we do make the simplifying assumption that the rudder angle can change instantaneously (we do not constrain $\dot{\theta}_\tau$). All constant quantities are in Newtons and radians:

$$-10000 < F_\tau < 10000$$
$$-\frac{2\pi}{5} < \theta_\tau < \frac{2\pi}{5}$$

To ensure that the trajectory does not collide, we enforce the constraint that the boat should be a certain distance from each object. For every object at every time step t, we define this constraint in terms of the objects position (x_o, y_o) and radius r (including some padding to account for the boats geometry):

$$\sqrt{(x_t - x_o)^2 + (y_t - y_o)^2} > r$$

We define a set of costs that can be optionally turned on or off - a cost C_{dist} penalizing longer trajectories, and a cost C_{fuel} penalizing fuel consumption. While we found that the basic constraints worked well for most trajectories, adding these costs can sometimes help generate more direct paths. They are defined as follows:

$$C_{dist} = (\dot{x}^2 + \dot{y}^2)\Delta t$$
$$C_{fuel} = \|F_\tau\|\Delta t$$

For the numerical solvers to work, the dynamics must be differentiable. The dynamics for the boat include signed squared values, generally computed by $|x|x$. However, $|x|$ has a infinite second derivative. For numerical stability, we approximate all instances of $|x|$ with:

$$|x| = \sqrt{x^2 + \epsilon}$$

This provides second-order differentiable dynamics, and with large enough ϵ also prevents the second derivative from being too large. For our experiments we use $\epsilon = 0.1$.

A. Initial Trajectory Guess

We found that one of the largest adjustable factors that affected the convergence of the optimizer was the initial trajectory guess. One of the biggest issues we faced when approaching this project was getting the optimizer to successfully converge on more complex currents given a simple initial guess - a linear interpolation in state from the starting configuration to the goal, with constant velocities inferred from the time duration and distance covered. In order to solve this issue, we implemented a two phase approach.

In the first phase, we optimize with no current. Our initial guess is formulated as a linear interpolation from the start to the goal for the x and y components. For the θ component guess we calculate the heading required to point directly at the target from the starting configuration, and we use this constant as the guess for all time steps. For the velocities, we simply infer a constant velocity:

$$\dot{x}_{guess} = \frac{x_{goal} - x_{start}}{steps\Delta t}$$
$$\dot{y}_{guess} = \frac{y_{goal} - y_{start}}{steps\Delta t}$$
$$\dot{\theta}_{guess} = 0$$

To guess the torque forces, we use $F_{guess} = 0$. To guess the torque angles, we use a heuristic based on the following idea: many paths most likely involves turning towards the goal, heading straight, and then turning up or down to match the final desired orientation. While this is not true of many paths, it does provide a good general initial guess that can lead the solver to discovering more complex trajectories. To implement this, we simply compute the angle required to turn from the starting heading to face the goal and set the negative of this angle as the torque heading guess for the first 10 time steps.

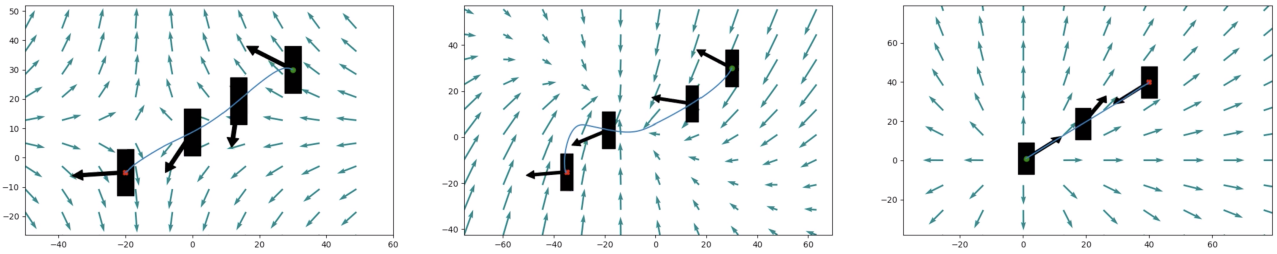

Fig. 4. Three successful trajectory optimizations using IPOPT. The start is in green, and the goal position is in red. These trajectories were generated without fuel or distance costs, and tolerances of zero on the dynamics constraint .

The last 10 time steps are set using the same logic applied to the target heading, while all other torque heading guesses are set to 0 to reflect the boat keeping a straight course towards the goal in the middle of the trajectory. The time length of 10 chosen here is based off intuition regarding how long it might take the boat to turn, but was not chosen in any rigorous manner.

In the second phase, we now optimize with the current using the trajectory generated by the first phase as the initial guess. This two phase approach was not always necessary, but did prove useful when optimizing in scenarios with complex currents or moving obstacles.

B. SNOPT Solver

For our first attempt, we used the SNOPT solver to generate an trajectory $\boldsymbol{X}$ and $\boldsymbol{U}$ from our optimization problem and initial guess. From our experience working with SNOPT in the context of this problem, the solver was frequently unable to solve more complex trajectories on a first pass, especially with the inclusion of obstacles. Using the two phase optimization approach described above, we were able to increase the number of scenarios in which the solver successfully converged. Without a cost function however, SNOPT still produced some unintuitive trajectories that - although physically valid - were far from optimal. With an added cost on either fuel C_{fuel} or distance C_{dist}, SNOPT found most optimizations infeasible.

C. IPOPT Solver

For this project, we found that IPOPT proved significantly more reliable than SNOPT. Using the same initial guesses, IPOPT was able to converge subject to a wider array of start and goal configurations. Fig. 5 shows the trajectories generated for the same start and goal configurations using both the SNOPT and IPOPT solvers.

While we still had to use introduce small tolerances in the dynamics constraints on the accelerations (no more than 0.1), to achieve convergence in a few difficult scenarios the performance of IPOPT was overall more robust than SNOPT. The performance between both solvers lined up for simple trajectories and currents, but IPOPT was able to converge on complex currents that SNOPT was not able to find solutions for. For a few examples of some trajectories generated with IPOPT in a variety of current fields, see Fig. 4.

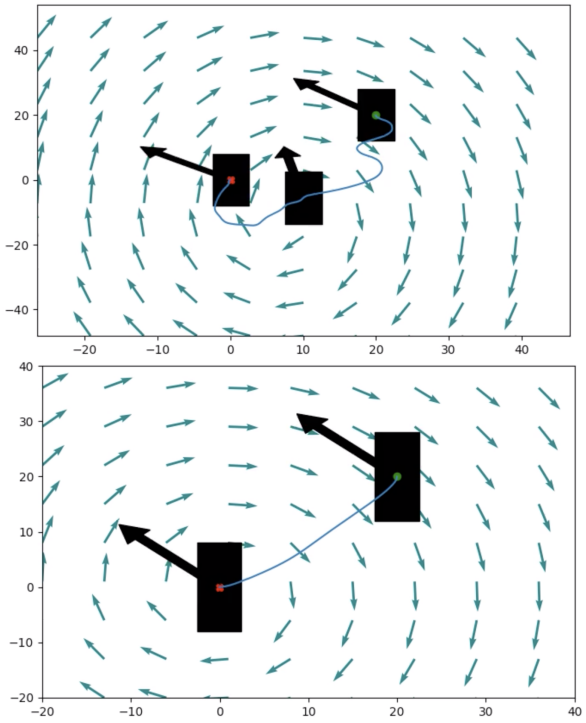

Fig. 5. The trajectories from SNOPT follow dynamics, but often produced unlikely solutions that are far from initial guess. The IPOPT solutions are much smoother and realistic, especially when handling the cost function feasibly.

We were able to converge on most scenarios in one pass (as opposed to the two phase approach often required for SNOPT), and the system was able to converge with moving obstacles. It's interesting to note that by utilizing the two phase optimization approach described, IPOPT was further able to converge in certain tricky scenarios. To fit the fixed optimization duration, IPOPT produced examples of delaying the boat to avoid moving obstacles. We also explored a boat slip docking scenario in a current, shown in Fig. 7.

V. Future Work

A. Optimization Robustness Experiments

While the results of the trajectory optimization are impressive in simulation, the current approach would most likely not scale well to a real autonomous boat. The real world

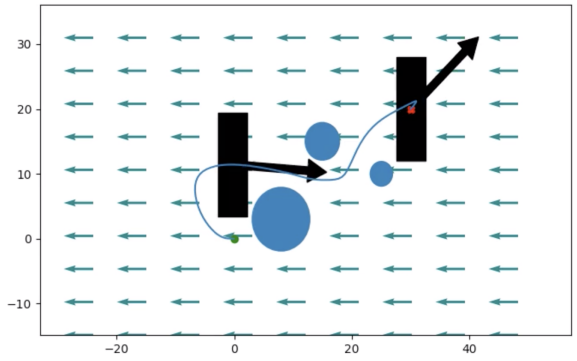

Fig. 6. A trajectory generated by IPOPT that works against the current to navigate through a set of static obstacles. The solver is also capable of generating trajectories when the obstacles are moving.

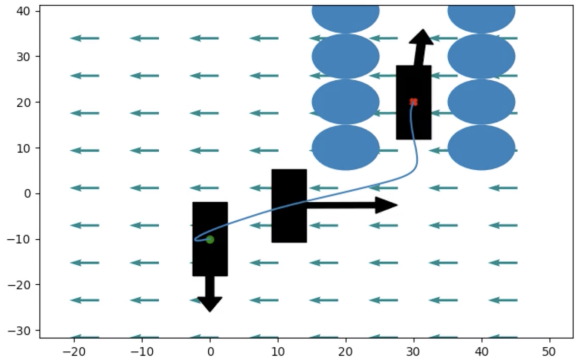

Fig. 7. IPOPT was able to dock the boat into a slip despite a lateral current. The boat handle this collision free path by gain momentum while approaching the slip, and maintaining a slightly upstream heading as much as obstacle constraints allowed.

dynamics of a boat are more complex than the simplified model presented here. One of the main issues that prevents scaling is the partial observability of the environment [1]. The approach described here requires full deterministic awareness of the current field and all object trajectories. To test our trajectories in such conditions, we could simulate our optimal control actions through a noisy current field environment, and observe how the trajectory diverges from the optimal.

We could further explore the stability of these trajectories in the presence of noisy environments. In particular, we can compare the performance of robust trajectory generation using methods such as Robust MPC or Tube MPC [5]–[7]. The common approach is modeling the environment with some bounded noise, and optimizing for a controller that can keep the vehicle within a invariant tube determined by the magnitude of the noise [6]. Thus, if the trajectory tube is obstacle free for all time, then a controller that keeps the vehicle in the tube also guarantees the vehicle will not collide with obstacles as long as error in the state estimation stays below the noise bound.

We expect that the additional difficulties of robustness in the optimization formulation would lead to more instances of problem infeasibility. This could be primarily caused by the large impact of current field on the boat's dynamics, and even small amounts of noise could lead to excessively large tubes in the more complex current environments, and thus could not be obstacle free.

B. Problem Feasibility

One problem encountered in the project was determining which parts of the problem formulation were causing infeasibility. Moreover, both the SNOPT and IPOPT solvers would often take much longer to determine a problem was infeasible compared to the time required to find a feasible solution when one existed. This can be expected, as there is a large space of potential solutions to check before one can determine a solution is infeasible. However, a faster infeasibility check, such as methods found in [8], [9], could be useful for a complex optimization framework. Loosening constraints iteratively for infeasible problems could provide a more useful initial guess or even a useful albeit non-optimal solution in the context of the boat trajectory problem, without the overhead of running a full optimization just to find that a certain formulation is infeasible.

VI. Conclusion

While trajectory optimization can generate impressive results in complicated dynamics, we've seen first hand how sensitive the solvers can be. From the initial guess, to the various hyperparameter choices - small details can determine whether or not the solver converges on a case by case basis. Finding the right duration to optimize over, or the right solver can clearly have a noticeable impact. Some of these issues can be mitigated in some sense - by implementing a two phase approach for example - but such approaches can add additional complexity to the problem, and feel like they should ideally be unnecessary. When formulating these optimizations, one should add constraints where necessary, but avoid extreme tolerances when not needed, and provide a simplistic educated initial guess that accurately adheres to the constraints and goals as you'd expect.

Code

All code along with sample videos and relevant documentation can be found here: https://github.com/echen9898/docking.

Acknowledgment

We'd like to thank the entire 6.832 staff for making this class so enjoyable. We'd like to especially thank Tobia Marcucci for his valuable insights on this project, and for going above and beyond throughout the semester.

Survey Keyword: underactuated

References

[1] L. Jaulin, *Combining interval analysis with flatness theory for state estimation of sailboat robots*. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.408.1785&rep=rep1&type=pdf

[2] A. B. Martinsen, A. M. Lekkas, and S. Gros, "Autonomous docking using direct optimal control," *IFAC-PapersOnLine*, vol. 52, no. 21, pp. 97–102, 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2405896319321755

[3] B. S. Thomas and P. D. Sclavounos, "Optimal Control Theory Applied to Ship Maneuvering in Restricted Waters," *Journal of Engineering Mathematics*. [Online]. Available: http://web.mit.edu/flowlab/NewmanBook/Sclavounos.pdf

[4] R. Tedrake and t. D. D. Team, *Drake: Model-based design and verification for robotics*, 2019. [Online]. Available: https://drake.mit.edu

[5] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*, A. Garulli and A. Tesi, Eds. London: Springer London, 1999, vol. 245, pp. 207–226, series Title: Lecture Notes in Control and Information Sciences. [Online]. Available: http://link.springer.com/10.1007/BFb0109870

[6] B. T. Lopez, J.-J. E. Slotine, and J. P. How, "Dynamic Tube MPC for Nonlinear Systems," *arXiv:1907.06553 [cs, eess]*, Jul. 2019, arXiv: 1907.06553. [Online]. Available: http://arxiv.org/abs/1907.06553

[7] D. Mayne, M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, Feb. 2005. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0005109804002870

[8] R. H. Byrd, F. E. Curtis, and J. Nocedal, "Infeasibility Detection and SQP Methods for Nonlinear Optimization," *SIAM Journal on Optimization*, vol. 20, no. 5, pp. 2281–2299, Jan. 2010. [Online]. Available: http://epubs.siam.org/doi/10.1137/080738222

[9] E. Khorshid, A. Alfadli, and A. Falah, "Application of constraint infeasibility detection methods in engineering design problems." *Journal of Engineering, Design & Technology*, vol. 16, no. 2, p. 285, 2018. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=edb&AN=129774736&site=eds-live&scope=site