Collision-free Trajectory Planning for Autonomous Surface Vehicle

Licheng Wen [1] , Jiaqing Yan [1] , Xuemeng Yang [1] , Yong Liu [1,2] ,and Yong Gu[1]

***Abstract*— In this paper, we propose an efficient and accurate method for autonomous surface vehicles to generate a smooth and collision-free trajectory considering its dynamics constraints. We decouple the trajectory planning problem as a front-end feasible path searching and a back-end kinodynamic trajectory optimization. Firstly, we model the type of two-thrusts under-actuated surface vessel. Then we adopt a sampling-based path searching to find an asymptotic optimal path through the obstacle-surrounding environment and extract several waypoints from it. We apply a numerical optimization method in the back-end to generate the trajectory. From the perspective of security in the field voyage, we propose the sailing corridor method to guarantee the trajectory away from obstacles. Moreover, considering limited fuel ASV carrying, we design a numerical objective function which can optimize a fuel-saving trajectory. Finally, we validate and compare the proposed method in simulation environments and the results fit our expected trajectory.**

I. Introduction

Autonomous surface vehicles(ASVs) motivated by its cheapness and eco-friendly operation, has sparked off much concern in the realm of robotics. Known for its reliable safety and efficiency, the ASV can meet the demands for fast, unmanned, technically reasonable multi-missions[28], including hydrological surveys, environment monitoring, and maritime rescuing.

The trajectory planning problem is not novel to researchers. As the traditional graph-based and sampling-based path searching methods [6] [15] [11] output position-only path, ASVs find it hard to follow due to lacking of directions and velocities on path. The generation of kinodynamic feasible trajectory requires taking ASV's nonlinear and under-actuated system into account. [7] first provides a differential flatness description of ASV's dynamics. [16] applies B-spline parameterization to deal with the dynamical constraints. [19] optimizes the trajectory by minimizing a cost function derived from the snaps and accelerations.

Although plenty of works on ASV trajectory planning have been proposed, there are still two critical issues that few researchers have mentioned. Firstly, there are usually obstacles in the target area. However, previous researches fail to focus on that. Besides, those who take obstacles into accounts rarely perform their trajectories in simulation environments or field experiments either. Secondly, energy cost plays a key role in our issue since ASVs carrying limited fuel out for missions. Researchers have considered some factors as optimized objectives in trajectory planning, such as accelerations, jerks, and path smoothness, but never concentrate on fuel-saving.

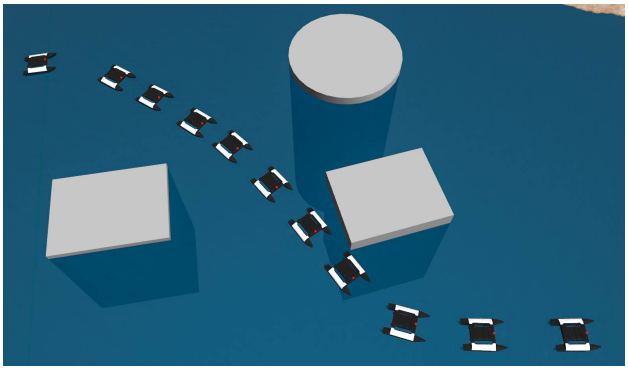

Fig. 1. Composite images of Kingfisher model sailing in the Gazebo simulation environment

In this paper, we propose a complete trajectory planning method to address these two issues systematically. Based on the modeling of ASV with two thrusts, we propose a framework that decouples the problem into front-end path searching and back-end trajectory optimization. We propose a method called "sailing corridor" to guarantee the trajectory both collision-free and kinodynamic feasible.

As for the energy-saving factor, fuel consumption is affected directly by the pumps' thrusts, which is also our control input. By setting minimum control input as the optimization objective function, we can obtain a fuel-saving trajectory.

The contributions of the present work are:

- Propose a method to generate smooth and collision-free trajectories using sailing corridor constraints;
- Design a numerical objective function which can optimize a fuel-saving trajectory;
- Perform experiments in simulation to verify our method's efficiency and accuracy;

The paper is organized as follows. Section II reviews the previous work related to our problem. Section III presents the ASV dynamic model and the front-end/back-end methodology. Section IV presents trajectory optimization problem with different objective functions, including fuel-saving. We also introduce constraints to make the trajectory smooth and collision-free. In Section V, the proposed solution is evaluated and compared with published methods in a simulation environment. Finally, Section VI summarizes and concludes the presented work.

[1]Licheng Wen , Jiaqing Yan , Xuemeng Yang ,Yong Liu and Yong Gu are from College of Control Science and Technology, Zhejiang University,Hangzhou, China

[2]Yong Liu is with the State Key Laboratory of Industrial Control Technology and Institute of Cyber-Systems and Control, Zhejiang University, Zhejiang, 310027, China (Yong Liu is the corresponding author, yongliu@iipc.zju.edu.cn)

II. Related Work

Path searching is to find a path in the configuration space of the vehicle from the initial location to the target region. This problem can be solved by several available methods, which produces feasible or optimal waypoints, and those methods are divided into two main aspects: graph-search based methods and incremental sampling-based ones. The first one describes the map as an occupancy grid or lattice where vertices represent a collection of reachable configurations, and edges represent links between vertices. The Dijkstra algorithm presented in [14] is a typical graph searching method, which discretizes the known map to cell-grid space and adds weights for each cell to find the shortest path. [6] first proposes a widely-used A* algorithm for unmanned vehicles, to produce an optimal path due to the implementation of heuristics, but usually costs much computational time and memory. [20] proposes the hybrid A* algorithm, which implemented the A* with practical constraints and performed well in DARPA Urban Challenge. [21] proposes an energy-efficient method combining Dijkstra's search, energy consumption, and Voronoi-Visibility. It can perform well in terms of the path's smoothness and continuity but does not address dynamic constraints. [23] discretizes the planning area with grids with states, called state lattice, where the path searching methods applied. It produces a smooth path for vehicles and performs well in relatively simple environments. The latter one randomly samples the configuration space or state space to find a feasible path, dealing with high dimensional spaces well with time constraints. [15] proposes a commonly used sampling-based algorithm, rapidly-exploring random tree(RRT), which builds a random tree of trajectories to find a path and is applied to dynamic systems. However, it generates jerky or not curvature continuous trajectories under some circumstances. [11] proposes RRT*, which iteratively builds a tree and lowers the given cost function through the state space. It is proved to be both asymptotic optimal and computationally efficient but still has the same deficiency as RRT. [10] extends RRT* to deal with differential-constraints models, applied well in anytime motion planning for Dubins' vehicles.

Trajectory generation used after path and waypoints found, ensures the trajectory smooth and collision-free, taking the vehicle's dynamic constraints into account. The traditional approaches to this problem can also be divided into two aspects: the interpolating curve approaches and the numerical optimization methods [5]. The first one replaces non-smooth pieces with shorter linear or parabolic segments [9]. It can generate highly-smooth trajectories but does not perform well in the higher-order case. [24] uses Bézier Curves to smooth the trajectory depended on control points with low computational time but lacks malleability when the curve degree increases. [26] introduces B-spline to generate a smooth and continuous curvature trajectory, which is a piece-wise polynomial parametric curve. [22] smooths piece-wise linear collision-free trajectories after sample-based planners computing and uses local spline refinement to avoid collisions. The latter one defines object functions about different constrained variables and minimizes or maximize them to produce expected trajectories. [29] achieves C_2 continuous by designing a function that takes the velocity, acceleration, and jerk into account, but not considering the dynamics constraints. [19] proposes the Minimum Snap method, which generates a safe and smooth trajectory by minimizing a cost function derived from the snap and accelerations, satisfying dynamics constraints. It is mainly applied but not limited to aerial vehicles and is also available for other differential systems. As the learning methods develop, the method of using machine learning to learn models via data from a manual operation is presented in [17], and [1] improves the performance by reinforcement learning. However, learning techniques do not deal with high-dynamics environments well with obstacles around. Moreover, safety is also critical for trajectory generation, which means collision-free and dynamically feasible. [13] proposes an MPC-based method under input and state constraints. However, this method is only efficient when the linearized model is fully controllable or if a control Lyapunov function can be synthesized [25]. In conclusion, it is of vital importance and challenging to generate smooth and collision-free trajectories for a non-linear and under-actuated dynamics system.

III. Methodology

A. ASV Dynamics

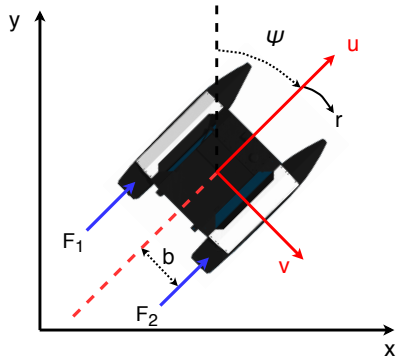

Fig. 2. Under-actuated ASV module, x-y denotes earth-fixed reference frame and u-v denotes body reference frame

In this paper, we use Kingfisher robot from Clearpath Robotics (see also [8]) as ASV model. It's an agile, battery operated boat with two propeller engines designed for research and rapid prototyping.

Two reference frames are introduced when characterizing ASV kinodynamics. The earth-fixed reference frame use North-East coordinates fixed to the ground, while the body reference frame follows the centroid of ASV. From [4] [18], the kinodynamics equations of ASV shows in Fig.2 are

$$\begin{cases} \dot{\eta} & = R(\psi)\nu \\ M\dot{\nu} & = \tau - C(\nu)\nu - D(\nu)\nu \end{cases} \tag{1}$$

Let $\eta = [x \quad y \quad \psi]^T \in \mathcal{R}^3$ and $\nu = [u \quad v \quad r]^T \in \mathcal{R}^3$ denote pose vector and velocity vector, respectively. x and y

are earth-fixed reference frame coordinates, and ψ represents the anticlockwise angle from earth-fixed frame to body reference frame, also called heading angle. u and v denote velocities in surge and sway, respectively. r is the yaw rate. $R(\psi)$ is the rotation transform matrix from body-fix reference frame to earth-fix reference frame. M is the intertial matrix, $C(\nu)$ denotes a Coriolis and Centripetal matrix and $D(\nu)$ denotes the damping matrix.

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

$$C(\nu) = \begin{bmatrix} 0 & 0 & mv \\ 0 & 0 & -mu \\ mv & -mu & 0 \end{bmatrix}$$

$$D(\nu) = \begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & 0 \\ 0 & 0 & N_r \end{bmatrix}$$

The jerk of ASV can be calculated using model dynamic equation (1) as

$$\begin{cases} \ddot{u} = \dfrac{\dot{\tau_u}}{m} - v\dot{r} - \dot{v}r - \dfrac{X_u}{m}\dot{u} \\ \ddot{v} = \dot{u}r + u\dot{r} - \dfrac{Y_v}{m}\dot{v} \\ \ddot{r} = \dfrac{\dot{\tau_r}}{I_z} - \dfrac{N_r}{I_z}\dot{r} \end{cases} \tag{2}$$

For the under-actuated ASV module discussed in this paper, two motor thrusts are incapable of providing direct control in sway velocity. Thus the control input $\tau = [\tau_u \quad \tau_v \quad \tau_r]^T \in \mathcal{R}^3$ can be represented by linear combination of two thrust forces, as b denotes half of ASV width:

$$\begin{cases} \tau_u = F_1 + F_2 \\ \tau_v = 0 \\ \tau_r = b\,(F_1 - F_2) \end{cases} \tag{3}$$

Unlike other researches usually use position, direction and velocities as state variable $\mathbf{x}$, we also add acceleration into state variable for trajectory optimization in the following section. So the state variable and control input $\boldsymbol{\tau}$ are:

$$\begin{aligned} \boldsymbol{x} &= [x \ y \ \psi \ u \ v \ r \ \dot{u} \ \dot{v} \ \dot{r}]^T \\ \boldsymbol{\tau} &= [\tau_u \ \tau_r]^T \end{aligned} \tag{4}$$

B. Two-Stage Trajectory Planning

In this paper, we propose a framework, which is divided into two sections: the front-end and back-end. The first step produces several collision-free waypoints in obstacle-occupied environments using path-searching methods. The second step generates a smooth and time-adjusting trajectory accounting for dynamics and state constraints. Considering limited energy while performing tasks, we design the Minimum Control Input Function to save energy. Non-smooth trajectory brings unnatural and even unsafe motions, so we design the Minimum Acceleration Function to sail as smoothly as possible.

We suppose that those obstacles in the target area are known and fixed before the trajectory planned. As Fig.3 shows, given the initial position, the target destination, and a map with known obstacles, the front-end produces a collision-free trajectory through the waters with obstacles. This trajectory constrains N waypoints, and each of them has exact coordinates and is away from obstacles. Considering computational-efficiency and optimality, we use RRT* planner [11][10] as front-end path searching module. As Algorithm 1 shows, RRT* iteratively builds the tree by randomly sampling the state s_{rand} from the configuration space and choose a path x_{new}, which extends the closest node $s_{nearest}$ in the tree toward the sample. If there is no obstacle lying in this path, RRT* compares the neighbors of s_{new} and evaluates the cost of each node as the parent node instead of directly inserting the new node s_{new} into the tree. The node with the lowest cost becomes the parent node and is added to the tree. The total cost contains the cost of reaching the potential parent node and the cost of the path to s_{new}. This process iteratively modifies the tree, reduces the total cost and makes the results asymptotic optimal.

Algorithm 1 Rapidly-exploring Random Tree*

$T \leftarrow$ InitializeTree()
$T \leftarrow$ InsertNode($\emptyset, s_{init}, T$)
for $i = 1$ to N **do**
  $s_{rand} \leftarrow$ Sample(i)
  $s_{nearest} \leftarrow$ Nearest(T, s_{rand})
  $(x_{new}, u_{new}, T_{new}) \leftarrow$ Steer($s_{nearest}, s_{rand}$)
  if CollosionFree($s_{nearest}, s_{rand}$) **then**
    $S_{near} \leftarrow$ Near($T, s_{new}, |V|$)
    $s_{min} \leftarrow s_{nearest}$
    $c_{min} \leftarrow$ Cost($s_{nearest}$) + c(s_{new})
    for $s_{near} \in S_{near}$ **do**
      $(x', u', T') \leftarrow$ Steer(s_{near}, s_{new})
      if CollosionFree(x') and $x'(T') = s_{new}$ **then**
        $c' = Cost(s_{near}) + c(x')$
        if $c' < Cost(s_{near})$ and $c' < c_{min}$ **then**
          $s_{min} \leftarrow s_{near}$
          $c_{min} \leftarrow c'$
        end if
      end if
    end for
    $T \leftarrow$ InsertNode(s_{min}, s_{new}, T)
    $T \leftarrow$ Rewrite($T, s_{near}, s_{min}, s_{new}$)
  end if
end for
return T

After the front-end complete generating waypoints, we need to optimize the trajectories which meet the vehicle's dynamic constraints and stay collision-free. Waypoints generated by sample-based planners sometimes result in jerky or unnatural motions, which may cause over-actuation or side sift. It is difficult for an actuator to execute such a path. In

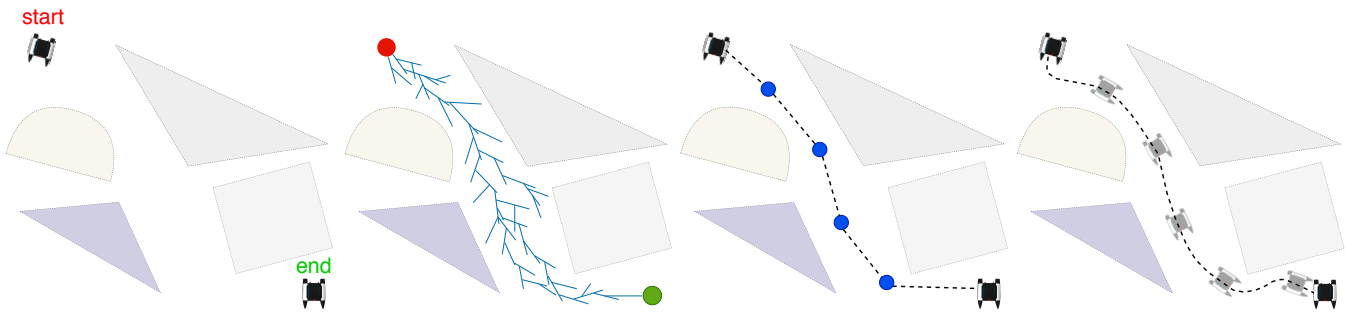

Fig. 3. (a). The given map with obstacles and demonstrate location of initial and target point; (b). The front-end RRT* searches the feasible path from the initial point to the target destination ; (c). Several waypoints are extracted from previous path through obstacles; (d). The proposed method generates a collision-free and smooth trajectory

order to generate high-quality trajectories, we optimize each sub-trajectory considering the ASV's dynamics and state's constraints. Suppose N waypoints generated after front-end, and then there are $N+1$ sub-trajectories considering both initial and target points. It is vital to allocate time for each sub-trajectory to achieve time-optimal. In terms of saving energy, we propose an object function accumulating the input control and time cost. In terms of improving smoothness, we propose another objective function accumulating acceleration and time cost. Meanwhile, the state's constraints are taken into account, including dynamics constraints and sailing corridor constraints. Sailing corridor constraint is a novel method we propose in this paper, applied to ensure the trajectory collision-free.

IV. Trajectory Optimization

A. Optimization Problem

In order to sail from the initial point (usually the current location) to the target destination and avoid all the obstacles, the trajectory must pass all the waypoints provided by front-end path searching. It should be clear that the state of the start/end point is consistent with our requirements, including position, heading angle, velocity, and acceleration on each axis. When passing the waypoints, we do not make too many requirements on heading angle and velocities.

Suppose we receive N waypoints from the front-end, the whole trajectory is divided into $N+1$ sub-trajectories. Our optimization object makes the trajectory begin with the initial state $\boldsymbol{z}_0$, passing through all waypoints and finally reaches the target state $\boldsymbol{z}_f$. Furthermore, the whole trajectory should satisfy all kinodynamic constraints.

As Fig.4 shows, for the ith ($i \leq N+1$) sub-trajectory, t_i denotes time cost for ASV to complete this sub-trajectory. $\boldsymbol{x}_i(t)$ and $\boldsymbol{\tau}_i(t)$ denotes the state function and control input function respectively at current sub-trajectory. Obviously, the ith sub-trajectory starts from $(i-1)$th waypoint and ends with ith waypoint, as the first sub-trajectory starts from the initial point and the last ends with the target point.

Let J_i be the objective function of ith sub-trajectory. We propose two different objective functions:

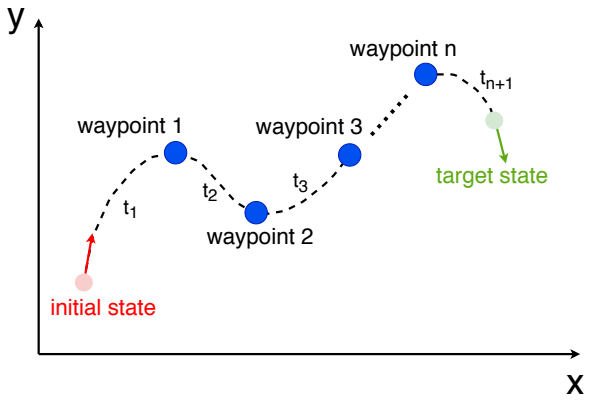

Fig. 4. Optimize each sub-trajectory and allocate time optimally, blue points denote n waypoints, pink point and green point denote initial and target state, respectively

1) Minimum Control Input Function: The most challenging problems for autonomous surface vehicles always contain using limited fuel patrolling as long as possible. This reminds us that it is necessary to plan a fuel-saving trajectory for practical tasks. Two thrusts generate forces F_1, F_2 which exert influence on fuel consumption directly and $\boldsymbol{\tau} = [\tau_u, \tau_r]^T$ are linear combinations of two forces. Minimizing control input $\boldsymbol{\tau}$ guarantees ASV sailing in an eco-friendly way along the output trajectory.

Besides, the time factor should be taken into account, as we expect ASV to consume less time while ensuring the task is completed well. Thus optimization problem is as follow:

$$\begin{aligned} min\ J_i &= \int_0^{t_i} \|\boldsymbol{\tau}_i(t)\|^2 + \lambda_1 t^2 dt \\ s.t.\quad & (7),\ (8),\ (9),\ (10) \end{aligned} \tag{5}$$

where λ_1 plays a normalization factor.

2) Minimum Acceleration Function: Though minimum control input has advantages in most scenarios, the ASV tends to sailing straightly as well as turning at a small radius. When smoothness becomes a major concern for trajectory planning, minimizing acceleration objective function is a more reasonable choice. Acceleration $\dot{\boldsymbol{\nu}}$ is included in state

variables $\boldsymbol{x} = [\boldsymbol{\eta}; \boldsymbol{\nu}; \dot{\boldsymbol{\nu}}]$. Similar to (5), time factor is also added into objective function.

$$J_i = \int_0^{t_i} \|\dot{\boldsymbol{\nu}}_{\boldsymbol{i}}(t)\|^2 + \lambda_2 t^2 dt$$
$$s.t. \quad (7),\ (8),\ (9),\ (10) \tag{6}$$

where λ_2 is a normalization factor.

B. Trajectory Constraints

Apart from dynamic equations and objective functions, some constraints are also required in our problem.

For the ith sub-trajectory,

1) Time constraint:

$$t_i \leq T \tag{7}$$

where T is the maximum time for each sub-trajectory.

2) Velocity constraint and acceleration constraint:

$$\begin{aligned} \boldsymbol{\nu}_{min} \leq \boldsymbol{\nu}_i(t) \leq \boldsymbol{\nu}_{max} \\ \boldsymbol{a}_{min} \leq \dot{\boldsymbol{\nu}}_i(t) \leq \boldsymbol{a}_{max} \end{aligned} \tag{8}$$

3) Control input constraint:

$$\boldsymbol{\tau}_{min} \leq \boldsymbol{\tau}_i(t) \leq \boldsymbol{\tau}_{max} \tag{9}$$

4) Sailing corridor constraint: We also need to consider the constraints resulting from obstacles. Though the front-end algorithm generates waypoints from a collision-free path, the output path connec ts two adjacent waypoints directly, without considering kinodynamics. As the situation in Fig.5 shows, there is no obstacle lies between the ith waypoint and the $(i+1)$th waypoint. However, the generated sub-trajectory is subjected to kinodynamics constraints and the initial state at the ith waypoint. We suppose ASV has a velocity at a negative y-axis direction when leaving the ith waypoint, which needs to adjust a large angle to arrive the $(i+1)$th waypoint. The red dotted line shows a possible but not feasible trajectory because ASV crashes into an obstacle during the voyage.

We propose a method called sailing corridor, which demarcates an area including two endpoints of the current sub-trajectory while excluding all the obstacles surrounding. Using $\boldsymbol{p}_i$ denotes the coordinates of the earth-fixed frame on the ith sub-trajectory, sailing corridor constraints can be formatted as:

$$corridor_{min} \leq \boldsymbol{p}_i(t) \leq corridor_{max} \tag{10}$$

5) Waypoints constraint: Finally, we need to optimize all the sub-trajectories in order and combine them into the final trajectory. As described above, besides the initial state and target state, we only constrain the position(x-y coordinate) on each waypoint. It is worth mentioning that for continuity and smoothness of the whole trajectory, both state variables and control input can not rise sudden changes. The characteristic of system states and control functions should remain continuous on each sub-trajectory, the connection of two adjacent trajectories must have consistent states (including velocities and acceleration) and control inputs.

The optimize procedure presents in Algorithm 2: using initial state and the coordinate of first waypoint, we obtain the analysis formula of time cost, control input function and state variable function regarding the first sub-trajectory; Then we use the last state of the previous sub-trajectory and the next waypoint to optimize the current sub-trajectory. Repeat the above process to generate each sub-trajectory, and we can finally obtain the whole path by connecting them.

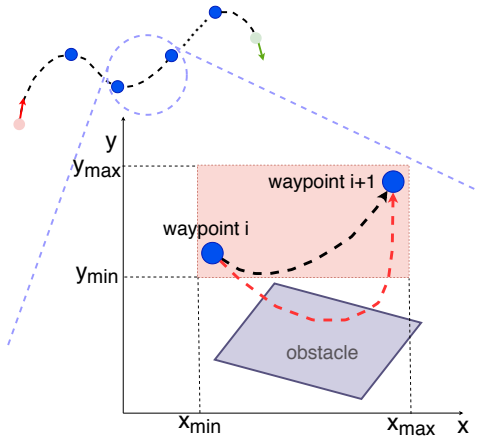

Fig. 5. The trajectory (black one) with sailing corridor constraints stays collision-free, while the other (red one) collides with the obstacle.

V. Experiment and Results

A. Simulation Environment

In the experiment, we adopt Kingfisher robot [8] as the ASV model and use the Gazebo with unmanned surface vehicle plugin provided by [3] as simulation environment in this section. According to technical specifications of Kingfisher boat, the parameters for ASV model are as $M = diag(29 \quad 29 \quad 2.8), \quad D = diag(20 \quad 20 \quad 20)$

Algorithm 2 Optimize Trajectory

Input: T,$\boldsymbol{z}_0$,$\boldsymbol{z}_f$,waypoints
Output: t,$\boldsymbol{x}$,$\boldsymbol{\tau}$

for $i = 1$ to $(N+1)$ **do**
  if $i = 1$ **then**
    $\boldsymbol{x}_i(0) = \boldsymbol{z}_0$
  else
    $\boldsymbol{x}_i(0) = \boldsymbol{x}_{i-1}(t_{i-1})$
  end if
  if $i = N+1$ **then**
    $\boldsymbol{x}_i(t_i) = \boldsymbol{z}_f$
  else
    $x_i(t_i) = waypoint_i.x$
    $y_i(t_i) = waypoint_i.y$
  end if
  Scan surrounding obstacles obtain sailing corridor
  $t_{i,max} \leftarrow T$
  Set constraints $\boldsymbol{\nu}_{min}$ $\boldsymbol{\nu}_{max}$ $\dot{\boldsymbol{\nu}}_{min}$ $\dot{\boldsymbol{\nu}}_{max}$ $\boldsymbol{\tau}_{min}$ $\boldsymbol{\tau}_{max}$
  Set objective function J_i
  Optimize $(t_i, \boldsymbol{x}_i, \boldsymbol{\tau}_i)$
end for

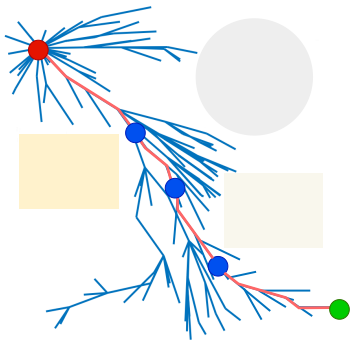

Fig. 6. RRT* produces a collision-free path from start point to target, and several feasible waypoints are extracted from that path

As for trajectory optimization method, trapezoidal direct collocation described in [2][12] is used. We apply a low-order spline to approximate the optimal trajectory. To be specific, the dynamics, objective function and control trajectories are approximated using a linear spline, and the state trajectory is a quadratic spline obtained by integration of the dynamics spline. The integration of a linear spline is computed using the trapezoid rule.

We place 3 different obstacles (two as cuboid and one as a cylinder) compactly in a 20×20 square meters task area as shown in Fig.1. Our goal is to make the ASV sailing from the top left corner to the bottom right corner with no collision to any of the obstacles. Both two cuboids have 5 meters in length and 4 meters in width centering at $(4.7, 8)$ and $(15, 6)$, respectively. We set the start point at $(2, 15)$ where ASV heading east with zero velocities and accelerations. We demand ASV still heading east when it arrives at the endpoint $(18, 1)$.

B. Simulation Result

Firstly we provide all the obstacles information and locations of start/goal point to front-end RRT^* planner. After having iterated hundreds of times, the front-end planner has successfully found a feasible path. Three waypoints are extracted from such path which are located at $(8, 10)$,$(10, 5)$,$(12, 3)$, respectively, as shown in Fig.6.

1) Constraints Setting: As described in Section IV, these waypoints along with two endpoints divides the whole path into four sub-trajectories. Now we present the constraints we adapt. Firstly sailing corridor constraints are considered with the information of obstacles. This constraint varies from different sub-trajectories due to different relative positions. For the first sub-trajectory we have $x_{1,max} = 11, y_{1,min} = 10$. The second and third sub-trajectory locates between two cuboids, thus the sailing corridors are both $x_{2,min} = 7.2, x_{2,max} = 12.5$ and $x_{3,min} = 7.2, x_{3,max} = 12.5$. The last sub-trajectory just needs to stay away from the right cuboid, as $y_{4,max} = 4$. Sailing corridors are also plotted in Fig.7. We also regularize $\psi_{min} = -\pi, \psi_{max} = \pi$ for heading angle. Kingfisher model has max surge speed at 1.7 meters per second while moves slowly backwards, thus we set $u_{max} = 1.7m/s, u_{min} = -0.1m/s$. Due to the under-actuated feature, ASV cannot control the sway velocity directly, thus we regularize $u_{min} = -1.0m/s, v_{max} = 1.0m/s$. Besides, $r_{min} = -\frac{\pi}{6}, r_{max} = \frac{\pi}{6}$. We set $T = 35s$ for the max allowed time for ASV to travel in each sub-trajectory.

We use both two optimize objective functions that we introduce above to generate trajectories.

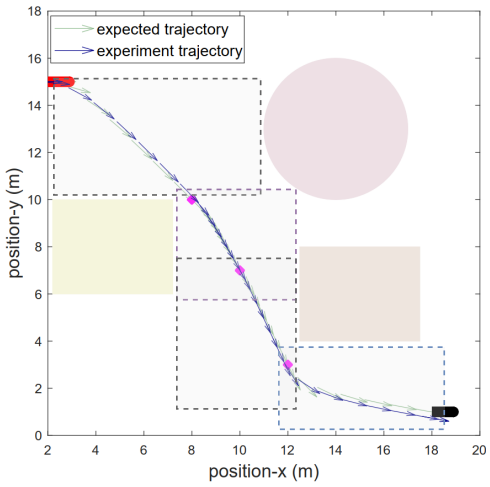

Fig. 7. Expected and experiment trajectories with the sailing corridor constraints for minimum control input optimization

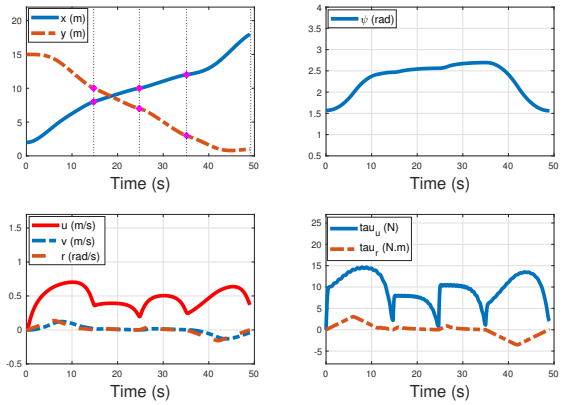

Fig. 8. Control and state variables for minimum control input optimization; The heading angle ψ are basically consistent with path directions. The velocity on surge u have an average of around $0.5m/s$ along the trajectory and sway velocity v remains low. The control input τ_u and τ_r generate cautiously for energy-saving objective.

2) Minimum Control Input Objective: The generated trajectory is demonstrated in Fig.7. The red boat shows the start position and heading angle while the black one presents the end position. Three pink points demonstrate the location of three waypoints generated by the front-end planner. The green arrow sequence denotes the expected trajectory using trapezoidal direct collocation and the blue one denotes the trajectory our model presents in the Gazebo environment. The direction of each arrow shows the heading angle ψ of the ASV at that time.

The detailed control variables and state variables are presented in Fig.8. The generated trajectory spends

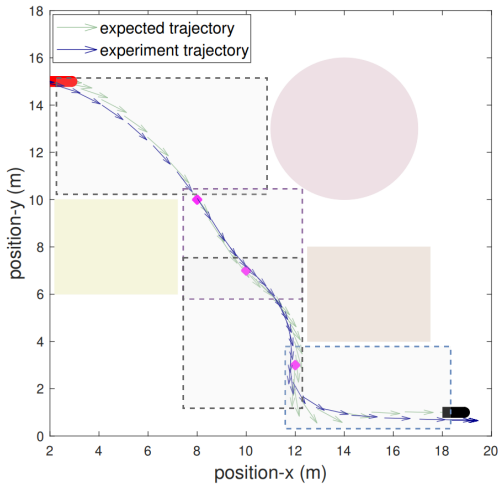

Fig. 9. Expected and experiment trajectories with the sailing corridor constraints for minimum acceleration optimization

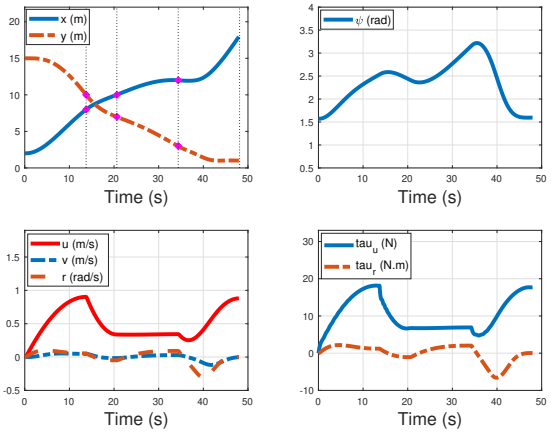

Fig. 10. Control and state variables for minimum acceleration optimization; The heading angle ψ changes sharply nearby the third waypoint. Velocities u, v and r are all inclined to remain constant after reaching relatively high speed. Thus the control input τ_u and τ_r are high to keep maintaining such velocities.

$t_i = [14.76 \quad 10.01 \quad 10.36 \quad 14.07]$ seconds on each sub-trajectory and the time duration of $t = 49.21$ seconds. We choose the optimization objective as the minimum control input $\boldsymbol{\tau}$, ASV tends to have a relatively large τ_u at the beginning of each sub-trajectory that makes itself have a high surge velocity. Then it needs fewer thrusts to maintain the velocities when getting closer to the end of the sub-trajectory to save fuel.

3) Minimum Acceleration Objective: The generated trajectory using minimum acceleration objective is demonstrated in Fig.9. The same symbol represents the same meaning as above. The detailed control variables and state variables are presented in Fig.10. The generated trajectory spends $t_i = [13.75 \quad 6.88 \quad 13.70 \quad 13.79]$ seconds on each sub-trajectory and the time duration of $t = 48.12$ seconds. Since the optimization objective changes to the acceleration $\dot{\boldsymbol{\nu}}$, the curves of velocities perform smoother than minimum control input in Fig.8. Furthermore, experiment trajectory tends to keep current velocity and disinclines to change directions.

C. Analysis and Comparisons

Though ASV dynamic model is slightly different between the simulation environment and theoretical model, it still shows that the experiment trajectory can fit the expected one pretty well. As we can see in both Fig.8 and Fig.10, the ASV not only arrives at the target with a demanding heading angle but also passes through all the waypoints considering its own width. The waypoints provided by the front-end guide our ship away from obstacles and sailing corridors ensure the safety of the final trajectory.

We compare with the method presented in [27], and both of the two approaches generate trajectories for under-actuated ASV in the environment with obstacles. The result shown in Wang's article performs relatively low-speed planning with an average speed of $0.269m/s$. Though it manages to avoid two obstacles in the environment, the trajectory shows the ASV sails just at the edge of the giant obstacle which makes it a dangerous and unacceptable trajectory in the real world. As a comparison, the closet distance between ASV and the nearest obstacle in our trajectory generated by minimum input control and minimum acceleration is $1.22m$ and $0.53m$ more than the above one, respectively. It proves the effectiveness and security of our proposed method.

TABLE I
COMPARISON OF TRAJECTORY

Trajectory	Avg Speed(m/s)	Min Obstacle Dist.(m)	Avg Ctrl. Input (N/s)
Methods in [27]	0.269	0	3
Minimum Control Input	0.4512	**1.22**	8.3
Minimum Acceleration	**0.503**	0.53	**10.21**

VI. CONCLUSION

In this paper, we propose a method to generate a collision-free and kinodynamic feasible trajectory for autonomous surface vehicles. We decouple the trajectory planning problem into a front-end feasible path searching and a back-end kinodynamic trajectory optimization. We adopt a sampling-based RRT* path searching to find an obstacle-safe path and extract several waypoints from it. After modeling the type of two-thrusts under-actuated surface vessel, we optimize the position-only path into a trajectory that satisfies the kinodynamic and model constraints. From the perspective of security in the field voyage, we propose a sailing corridor method to guarantee the trajectory away from obstacles. Moreover, considering limited fuel ASV carrying, we design a numerical objective function which can optimize a fuel-saving trajectory. With trapezoidal direct collocation, all control and state variables are approximated into spline and can be used directly to control the movement of ASV. Finally, we validate our proposed method in a complex simulation environment.

ACKNOWLEDGEMENT

This work is supported by the National Key R&D program of China under Grant 2018YFB1305900 and the National Natural Science Foundation of China under Grant 61836015.

REFERENCES

[1] Abbeel, P.: Apprenticeship learning and reinforcement learning with application to robotic control. Stanford University (2008)
[2] Betts, J.T.: Practical methods for optimal control and estimation using nonlinear programming, vol. 19. Siam (2010)
[3] Bingham, B.: Unmanned surface vehicle plugins for gazebo simulation. https://github.com/bsb808/usv_gazebo_plugins. Accessed January 5, 2020
[4] Fossen, T.I.: Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons (2011)
[5] González, D., Pérez, J., Milanés, V., Nashashibi, F.: A review of motion planning techniques for automated vehicles. IEEE Transactions on Intelligent Transportation Systems **17**(4), 1135–1145 (2015)
[6] Hao, Y., Agrawal, S.K.: Planning and control of ugv formations in a dynamic environment: A practical framework with experiments. Robotics and Autonomous systems **51**(2-3), 101–110 (2005)
[7] Hervagault, Y., Prodan, I., Lefevre, L.: Trajectory generation with communication-induced constraints for surface vehicles. In: 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), pp. 482–487. IEEE (2017)
[8] Inc., C.R.: Heron unmanned surface vessel for aquatic research. https://clearpathrobotics.com/heron-unmanned-surface-vessel/. Accessed January 4, 2020
[9] Kallmann, M., Aubel, A., Abaci, T., Thalmann, D.: Planning collision-free reaching motions for interactive object manipulation and grasping. In: Computer graphics forum, vol. 22, pp. 313–322. Wiley Online Library (2003)
[10] Karaman, S., Frazzoli, E.: Optimal kinodynamic motion planning using incremental sampling-based methods. In: 49th IEEE conference on decision and control (CDC), pp. 7681–7687. IEEE (2010)
[11] Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. The international journal of robotics research **30**(7), 846–894 (2011)
[12] Kelly, M.: An introduction to trajectory optimization: How to do your own direct collocation. SIAM Review **59**(4), 849–904 (2017)
[13] Kim, H.J., Shim, D.H., Sastry, S.: Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In: Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301), vol. 5, pp. 3576–3581. IEEE (2002)
[22] Pan, J., Zhang, L., Manocha, D.: Collision-free and smooth trajectory computation in cluttered environments. The International Journal of Robotics Research **31**(10), 1155–1175 (2012)
[14] LaValle, S.M., Hutchinson, S.A.: Optimal motion planning for multiple robots having independent goals. IEEE Transactions on Robotics and Automation **14**(6), 912–925 (1998)
[15] LaValle, S.M., Kuffner Jr, J.J.: Randomized kinodynamic planning. The international journal of robotics research **20**(5), 378–400 (2001)
[16] Louembet, C., Cazaurang, F., Zolghadri, A.: Motion planning for flat systems using positive b-splines: An lmi approach. Automatica **46**(8), 1305–1309 (2010)
[17] Lupashin, S., Schöllig, A., Sherback, M., D'Andrea, R.: A simple learning strategy for high-speed quadrocopter multi-flips. In: 2010 IEEE international conference on robotics and automation, pp. 1642–1648. IEEE (2010)
[18] McCue, L.: Handbook of marine craft hydrodynamics and motion control [bookshelf]. IEEE Control Systems Magazine **36**(1), 78–79 (2016)
[19] Mellinger, D., Kumar, V.: Minimum snap trajectory generation and control for quadrotors. In: 2011 IEEE International Conference on Robotics and Automation, pp. 2520–2525. IEEE (2011)
[20] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., et al.: Junior: The stanford entry in the urban challenge. Journal of field Robotics **25**(9), 569–597 (2008)
[21] Niu, H., Lu, Y., Savvaris, A., Tsourdos, A.: An energy-efficient path planning algorithm for unmanned surface vehicles. Ocean Engineering **161**, 308–321 (2018)
[23] Pivtoraiko, M., Kelly, A.: Efficient constrained path planning via search in state lattices. In: International Symposium on Artificial Intelligence, Robotics, and Automation in Space, pp. 1–7 (2005)
[24] Rastelli, J.P., Lattarulo, R., Nashashibi, F.: Dynamic trajectory generation using continuous-curvature algorithms for door to door assistance vehicles. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings, pp. 510–515. IEEE (2014)
[25] Shen, C., Shi, Y., Buckham, B.: Trajectory tracking control of an autonomous underwater vehicle using lyapunov-based model predictive control. IEEE Transactions on Industrial Electronics **65**(7), 5796–5805 (2017)
[26] Shiller, Z., Gwo, Y.R., et al.: Dynamic motion planning of autonomous vehicles. IEEE Transactions on Robotics and Automation **7**(2), 241–249 (1991)
[27] Wang, J., Wang, J.: Neurodynamics-based distributed receding horizon trajectory generation for autonomous surface vehicles. In: International Conference on Neural Information Processing, pp. 155–167. Springer (2018)
[28] Yan, R., Pang, S., Sun, H., Pang, Y.: Development and missions of unmanned surface vehicle. Journal of Marine Science and Application **9**(4), 451–457 (2010)
[29] Ziegler, J., Bender, P., Dang, T., Stiller, C.: Trajectory planning for bertha—a local, continuous method. In: 2014 IEEE intelligent vehicles symposium proceedings, pp. 450–457. IEEE (2014)