DESIGN, MODELING, AND CONTROL OF SOFT DYNAMIC SYSTEMS

Yaorui Zhang
yaoruizh1997@gmail.com

DESIGN, MODELING, AND CONTROL OF SOFT DYNAMIC SYSTEMS

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master

in

Computer Science

by

Yaorui Zhang

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 2021

Examining Committee:

Bo Zhu

Bo Zhu, Chair

Lorie Loeb

Lorie Loeb

Xing Dong

Xing-Dong Yang

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies

Abstract

Soft physical systems, be they elastic bodies, fluids, and compliant-bodied creatures, are ubiquitous in nature. Modeling and simulation of these systems with computer algorithms enable the creation of visually appealing animations, automated fabrication paradigms, and novel user interfaces and control mechanics to assist designers and engineers to develop new soft machines. This thesis develops computational methods to address the challenges emerged during the automation of the design, modeling, and control workflow supporting various soft dynamic systems. On the design/control side, we present a sketch-based design interface to enable non-expert users to design soft multicopters. Our system is endorsed by a data-driven algorithm to generate system identification and control policies given a novel shape prototype and rotor configurations. We show that our interactive system can automate the workflow of different soft multicopters' design, simulation, and control with human designers involved in the loop. On the modeling side, we study the physical behaviors of fluidic systems from a local, collective perspective. We develop a prior-embedded graph network to uncover the local constraint relations underpinning a collective dynamic system such as particle fluid. We also proposed a simulation algorithm to model vortex dynamics with locally interacting Lagrangian elements. We demonstrate the efficacy of the two systems by learning, simulating and visualizing complicated dynamics of incompressible fluid.

Acknowledgement

I want to show my sincere gratitude to everyone who helped me in my graduate study. First, I would like to thank everyone in Bo Zhu's lab. The projects mentioned in the thesis are all built on the cooperation of members in the lab. Especially, I would like to thank Yitong Deng and Shiying Xiong, who gave me a lot of support in the projects, as well as useful suggestions in writing the thesis; Shuqi Yang for answering my questions so patiently and explaining things so clearly; Xingzhe He, Dongkai Chen and Chongyang Gao for helping me build the neural networks. I shall never thank my advisor, Professor Bo Zhu too much. I'm so lucky to have such a great advisor who is both strong in the research field and supportive to his students. He encouraged me a lot when I failed to get a satisfying result and could always direct me back to the right path.

Next, I would like to thank my program director, Lorie Loeb, who always encourages me to chase my dreams and do what I want to; Xing-Dong Yang for being responsive and direct me to the right path when I first get to the research field.

Last but not the least, I want to thank my parents, Lide Zhang and Side Zhang, who are always there for me; my friend, Will Xue for cheering me up in my most depressing moments.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The thesis consists of three projects that are all exploring the design, modeling and control of soft dynamic systems. Full content of the multicopter project and vortical flow project can be found in papers [Deng et al., 2020; Xiong et al., 2021] that I coauthored.

1.1 Soft Multicopter

Today, rigid multicopters have been widely used in photography, product delivering, monitoring, etc., but have you ever imagined operating a deformable drone which could adjust its shape according to the environment or the terrain, entering narrow spaces and conducting tasks that are hard for those typical drones made of metal and plastic?

Unfortunately, soft drones are not common since they are hard to control. To the best of our knowledge, there's no methods to control the deformation of drones so far. For real drones, the observable data from sensors can be very limited, and the deformation make the posture of a soft drone hard to define.

To solve the problem, we decompose the state of a soft multicopter into translation, rotation, and pure deformation components, which are measured purely and conveniently by Inertial Measurement Units (IMUs), and train a neural model to predict their nonlinear couplings. The learned dynamic model is integrated into a non-conventional Linear Quadratic Regulator (LQR) based control loop, enhanced by a novel online relinearization scheme that enables the soft drone to perform various tasks.

We show examples of controlling both 2D and 3D drones. To design 2D drones with organic shapes, we provide a web-based interface to sketch drones contours.

We demonstrate the efficacy of our approach by generating controllers for a broad spectrum of customized soft multicopter designs and testing them in a high-fidelity physics simulation environment.

1.2 Multi-body constraint systems

Position based dynamics has been a hot topic due to its advantage over stability, robustness and speed, compared with traditional simulation methods Müller et al. [2007]. A stable status could be reached after several iterations via constraint functions. However, not all systems could be easily analysed and find a explicit constraint function. Meanwhile, it's an interesting topic to predict the future status of a system based on observations from previous frames, even with no or few proir knowledges.

Yang et al.'s work Yang et al. [2020] integrates position based dynamics and neural networks, demonstrates that for both rigid and soft bodies, the constraints could be learned purely from the observation data, and the future status can be predicted in a diligent manner.

In this work, we propose to use the position based framework to learn soft dynamic systems with a looser connection, e.g, fluids. We do this by constructing connection by distance, learning the pair-wise relationship among neighbors, accumulate neighbor influences, find the constraint, and then do projection. In this way, we can learn from one set of particle data and apply it to a dynamics with the same property but with a different setting, e.g. a different number of particles, or applied to extra forces. We designed neural networks with different structures, and show their ability of learning the constraint of a fluid system.

1.3 Predicting multi-body vortical flow motion

We propose a vortex segment cloud method that combines the flexibility of the vortex particle method and the stability and accuracy of the vortex filament method. To represent the vorticity field, we discretize it as a series of vortex segments of certain lengths. To deal with local topology changes, we devise three local segment reseeding operations including splitting, merging and deletion to enable sophisticated topology changes of vortical fluid. With this method, we realize the reconnection of nonclosed vortex tubes using the pure Lagrangian method for the first time to the best of our knowledge. After generating the simulation data, we built the scenes with artistic consideration and rendered realistic result in Houdini.

Chapter 2

Related Work

2.1 Soft Multicopter Control

Multicopter Control In recent years, multicopters have emerged to dominance in the realm of commercial UAVs, thanks to their simple mechanical structures, optimized efficiency for hovering, and easy-to-control dynamics [Tedrake, 2020; Agrawal and Shrivastav, 2015].

Various methods have been successfully developed to control multicopters, including PD/PID [Tayebi and McGilvray, 2004], LQR [Du et al., 2016; Bouabdallah et al., 2004], differential flatness [Mellinger and Kumar, 2011], integral sliding mode [Waslander et al., 2005], and MPC [Wang et al., 2015] methods. Nonconventional geometries [Du et al., 2016], hybrid wing-copter modes [Xu et al., 2019], articulated structures [Zhao et al., 2018b,a], and foldable structures [Floreano et al., 2017] have been tackled in the controller design problem. Recent works have also been done to extend drone's ability to actively deform itself to pass through tight spaces [Zhao et al., 2018b; Kulkarni et al., 2019] or to perform secondary functionalities like grasping [Anzai et al., 2018], via elaborately designed assembly of linked multicopters.

Du et al. [Du et al., 2016] create an automatic system that generates feedback controllers for user-designed non-standard rigid drone geometries with any number of rotors. Our work shares the same spirit with them where we also aim to facilitate novel drone designs by automatically generating the controller for them. PD control is lightweight and effective, but it generally requires the handcrafting of cascading structures and the empirical tuning of various parameters. LQR controllers, in comparison, comes in one piece, requires less parameter tuning, and it solves for the optimal control signal. However, it requires the full dynamic model and assumes all states are measurable, and it is designed for linear systems only.

Learning-based Soft-Body Control The control of soft robots has been extensively studied [George Thuruthel et al., 2018]. However, up to date it remains a very challenging topic due to the under actuated nature of the high-dimensional state space for a soft body [Rus and Tolley, 2015]. A broad array of control mechanics, including the simulation-driven control [Bieze et al., 2018], morphological computing schemes [Urbain et al., 2017], and learning-based physics simulators [Battaglia et al., 2016; Spielberg et al., 2019] have been proposed to reduce the complexity or accelerate the computation of a soft-body control problem. For example, Spielberg et al. [Spielberg et al., 2019] propose a end-to-end training method that uses an autoencoder to map high-dimensional state vector to low-dimensional latent vectors as input to the neural network controller. Among these approaches, neural physics simulators [Battaglia et al., 2016; Li et al., 2019] play an important role in connecting the real-world physics and the numerical controller by providing an differentiable surrogate model.

2.2 Learning multi-body constraint systems

Position-Based Dynamics Position based dynamics has been used widely in game and visual effects industries due to its stable, robust and fast nature. A number of research have been conducted to simulate different dynamics [Müller et al., 2007], including fluid [Macklin and Müller, 2013]. Yang et al.'s work [Yang et al., 2020] integrates position based dynamics and neural networks, demonstrating that a simple constraint could be learned to represent complex physical systems.

Learning Fluid Dynamics Using neural networks to learn fluid parameters or to accelerate fluid simulation has been extensively studied. SPNets [Schenck and Fox, 2018] proposed an approach to simulate fully differentiable fluid dynamics and give examples to learn fluid parameters from data, perform liquid control tasks, and learn policies to manipulate liquids. Yang et al. [Yang et al., 2016] proposed a data-driven projection method to accelerate grid-based fluid simulation by using a neural network to avoid iterative computation, while Dong et al.'s fluidnet [Dong et al., 2019] accelerates Eulerian fluid simulation by generating multiple neural networks before the simulation. All of these methods requires prier knowledge of fluid properties. In Sanchez-Gonzalez et al.'s work [Sanchez-Gonzalez et al., 2020], Graph Network-based Simulators has been used to represents the state of a physical system with particles (including fluids), expressed as nodes in a graph, and computes dynamics via learned message-passing.

2.3 Predicting multi-body vortical flow motion

Vortex particle method Although early works adopted point vortices to numerically simulate the dynamical evolution of 2-D inviscid flow in an unbounded domain [Rosenhead, 1931; Takami, 1964], the modern vortex method is marked by the vortex blob method proposed by [Chorin, 1973], which removes the singularity in the kernel function by replacing the point vortex with certain vortex cores. The vortex blobs might be of various shapes, such as an isotropic sphere or a small vortex sheet [Pfaff et al., 2012a]. Various options exist for the vorticity distribution in vortex blob methods, such as the Gaussian distribution [Park and Kim, 2005], the Rankine vortex model [Loiseleux et al., 1998], and the Krasny model [Krasny, 1988], etc. There are possible limitations of using the vortex blob method to solve large-scale complex vortical flow. For long-term computational accuracy, a vortex blob method requires that each vortex element overlap its neighboring blobs, which consumes a massive number of vortex elements for computational stability [Hald and Del Prete, 1978; Hald, 1979]. Besides, the shape of every single blob is different from the common filamentous or tubular structures in the flow field, making it challenging for vortex methods to form coherent structures under a turbulent setting [She et al., 1990]. Finally, a vortex blob method updates the vorticity stretching term with the original, transposed [Choquin and Huberson, 1990], or symmetrical [Cottet and Koumoutsakos, 2000] form by taking the derivative of the kernel function. The correctness of a numerical stretching relies on the distribution of vortex elements together with the choice of a proper kernel function to ensure numerical precision and adaptability [Angelidis, 2017]. In the absence of ambiguity, we refer to the vortex particles/points below as vortex blobs.

Vortex filament method Vortex filaments are important for 3D turbulence dynamics [Xiong and Yang, 2017, 2019b], providing one of the most efficient numerical methods to reproduce the complexity of smoke with sparse discrete primitives [Weißmann et al., 2014; Eberhardt et al., 2017]. The numerical simulation of vortex filaments can be traced back to Hasimoto's study on the local induction approximation

(LIA) of some isolated vortex filaments, which validates that a single vortex filament in LIA fits well with the experimental results of the propagation of isolated waves on a twisted structure [Hasimoto, 1972; Hopfinger et al., 1982; Aref and Flinchem, 1985]. For the first time, [Angelidis and Neyret, 2005] simulated the flow field with a large number of closed vortex filaments. From a Hamiltonian perspective, [Weißmann and Pinkall, 2009] proposed a physically conservative model that compensates for the discretization errors inherent to the polygonal vortex filament model. [Barnat and Pollard, 2012] developed a new set of reconnection criteria to simulate smoke with a filament graph. Then, [Padilla et al., 2019] simulated elaborate physical phenomena with the thickness of vortex filaments taken into account, such as the dynamic evolution of an ink drop. A potential limitation of vortex filament methods is the need for tedious mesh repair operations to handle their topological changes, such as splitting and merging [Chorin, 1990, 1993; Marzouk and Ghoniem, 2007; Bernard, 2009]. These operations also make it challenging to establish large-scale parallel processing algorithms. The vortex sheet method, also based on mesh connectivities, is specialized to capture codimension-1 vortex structures evolving in three-dimension space [Pfaff et al., 2012b; Brochu et al., 2012].

Chapter 3

Design and Control of a Soft Multicopter

3.1 Soft Multicopter Dynamics

A soft multicopter can be interpreted as a soft, continuum body Ω attached with n rotors. Let $\mathbf{X}$ be the material coordinates of points in a body domain Ω and $\mathbf{x}$ be their world-space coordinates. The two coordinate systems are bridged by a deformation mapping $\mathbf{x} = \Phi(\mathbf{X})$. The soft material model is described by its density ρ, damping γ, and an elasticity model denoted by a functional $\epsilon(\Phi)$. In the simulation code, ϵ can be implemented as a Neo-Hookean model, co-rotated model, and so on [Kim et al., 2012; Skallerud and Haugen, 1999]. We design a control mechanism that is independent from the exact deformable model implementation, meaning that it can work with different numerical or real-world deformable systems by observing different data sets. A rotor on a soft drone is defined by a tuple $\{u_i, \lambda_i, \mathbf{T}_i, \mathbf{r}_i\}$, with u_i as the magnitude of the propeller thrust, λ_i as the spinning direction, $\mathbf{T}_i$ as the thrust direction in the world space, and $\mathbf{r}_i$ as the rotor position in the material space. We assume each rotor is stick to a local point near the surface of the body in material space. The rotor direction is given as the average of the surface normals in the local region around $\mathbf{r}_i$, i.e., $\mathbf{T}_i = \sum_{j\in Nb(i)} \mathbf{n}_j / |\sum_{j\in Nb(i)} \mathbf{n}_j|$ in a discrete setting, with $\mathbf{n}_j$ as the normal direction of a neighboring surface triangle.

From Newton's second law, the soft multicopter dynamics can be written as:

$$\ddot{\mathbf{x}} + \gamma\dot{\mathbf{x}} + \nabla_{\mathbf{x}}\epsilon = \mathbf{b}(\mathbf{X}) + \mathbf{g}. \tag{3.1}$$

The left-hand side of Equation 3.1 describes the soft body's internal forces, including the inertial force, damping force, and elastic force. The right-hand side describes the body's external forces, including the thrust input $\mathbf{b}(\mathbf{r}_i) = \lambda_i u_i \mathbf{T}_i$, and the gravity $\mathbf{g}$. Compared with the rigid-body multicopter dynamics equation (e.g., see [Du et al., 2016]), the soft-body version does not have the Euler's equation to describe the body's rotational movement. The torque effect of a rotor is considered in the elastic solve by enforcing boundary conditions from $\mathbf{b}$. The spinning torque effect is eliminated on the design stage by implementing each rotor as a pair of propellers spinning in the opposite directions.

3.2 Drone Designs

3.2.1 2D drone design

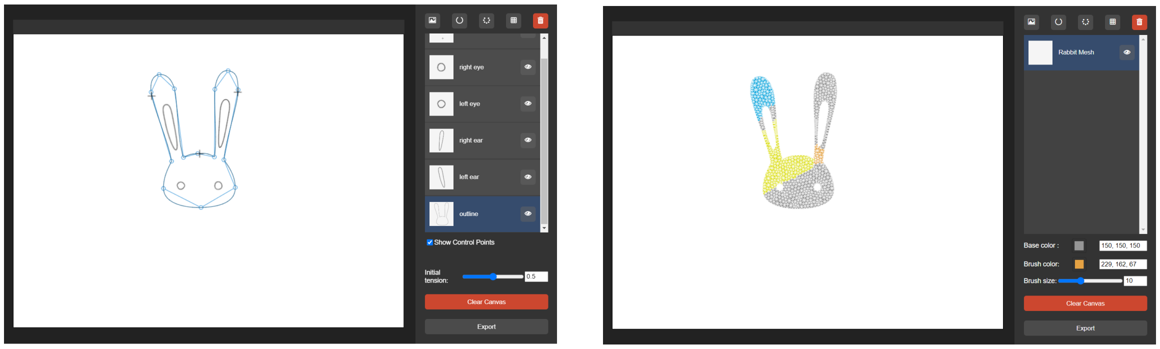
To customize 2D drones, we develop a web-based interface to sketch contours and assign rotor positions of drones.

The interface provides 4 types of layers: curves, point sets, images and meshes. A curve layer is where the users actually sketch the contour of the drone. It's similar to the pen tool in other software. Users can draw Hermite splines by assigning control points, and move the tangent handler to adjust the tension.

A point-set layer is where users could assign rotor positions. By simply clicking on the screen, users can assign as many rotors as they want. The rotors should be attached to the boundary of the contour. Otherwise the dot would be ignored and no rotor would be actually added in our simulator.
An image layer allows users to add pictures as references. When users click "change color", a color layer with a screen blend mode would be added on top of the picture, making it easier for users to see distinguish the contour of the picture.
By clicking "export", the curved drone contours will be exported as a sequence of vertices, as well as the rotor positions. Since the curve is defined by connected segments, by specifying the detail level, users can define the smoothness of the curves. We use TetGen[Si, 2015] to generate a volumetric mesh (for 2D case it refers to a triangle mesh) from the segment data.
The generated mesh could be used directly in our simulation program. It can also be imported again into our sketching interface. It would create a new layer where users can assign colors to the triangles of the mesh. The mesh could be exported again with a value converted from the color information of each triangle. Then in our simulator, the weight could be interpreted as the material type.

To try this tool, please visit `https://2d-contour-generator.readthedocs.io/en/latest/intro.html`3.1 Users can export the curved drone contours as a sequence of vertices with specifying the detail level, as well as the rotor positions.

Future work One main gap for the tool is from the contour to the mesh. We developed a simple CPP program integrating TetGen to generate a triangle meshe file from the input file with segment information. The workflow would be much smoother if we could generate the triangle mesh directly on the webpage. Another possible improvement is that although we allow users to draw super organic shapes, symmetric shapes are desired in many cases in drone designing. Mirroring functions



[✓ Show Control Points]

Initial tension: 0.5

Base color : 150, 150, 150

Brush color: 229, 162, 67

Brush size: 10

Figure 3.1: Left: sketching the contour; Right: painting the mesh.

could be very helpful. Besides, not all rotor configurations make sense. To better control a soft shape, it's possible to give suggestion for rotor settings.

3.2.2 3D drone design

3D drones are modeled in Maya as closed surface meshes. Then again we use TetGen[Si, 2015] to create tetrahedron meshes from the triangle meshes and add rotors to the assigned positions.

3.2.3 Geometry and Material

As mentioned previously, we separate the representation of a soft drone's state into three geometric components: the deformation vector **s**, the rotaiton vector **e** and the position vector **p**. The **s** and **e** vectors which jointly defines the soft drone's deformation is measured by Inertial Measurement Units (IMU) only. Since rotation in 2D can be represented by one scalar only, for 2D drones the IMU will only output the angle between the measured vector and the horizontal. The measured vectors are shown in Figure 3.3. The specifications of our models tested are presented in Table. 3.1. For the 3D examples, the sensing scheme is shown in Figure 3.2. Each IMU is able to output the rotation information of itself as a rigid object, which is attached to a local region on the surface of the drone. In other words it defines its own reference

Table 3.1: Design Specifications

3D models						
specs	Donut	Starfish	Flower	Leaf	Octopus	Orange peel
mass(kg)	1	1	1	1	1	1
modulus(N/m^2)	1e4	3e3	6e3	3e3	1e4	5e2
length-x(m)	3.5	3.6	3.6	2.4	3.6	3
length-y(m)	0.36	0.375	0.225	0.075	1.5	1.3
length-z(m)	3.5	3.6	3.6	4.5	3.6	2.9
num sensors	4	4	8	4	8	5
num rotors	4	5	9	4	9	5
max thrust(N)	10	10	10	10	10	10

2D models						
specs	Engine	Bunny	Diamond	Elephant	Rainbow	Long Rod
mass(kg)	1	1	1	1	1	1
modulus(N/m^2)	6e3	6e3	6e3	6e3	6e3	6e3
length-x(m)	1.90	1.12	1.47	2.46	2.08	0.1
length-y(m)	2.16	1.75	1.42	1.69	1.30	8.0
num sensors(m)	6	3	8	3	4	8
num rotors(m)	6	3	4	2	2	5
max thrust(N)	10	10	10	10	10	10

frame with its X,Y,Z axes. On the figure, the X, Y, Z axes are coded by Red, Green, Blue respectively, with the Y axes pointing out of the plane. Each measurement will be done by an individual IMU, and for the peripheral measurements we will only make use of the measured Y axis neglecting the X and Z axes.

The stiffness of the models are defined by the elastic modulus, which measures the drone's resistance to being deformed elastically. We adjust the modulus of each example according to their shape to make sure the deformation is not too subtle to notice nor too dramatic to control.

Though the table only shows examples with one elastic modulus value that applies to the whole model, the value is actually assigned to vertices separately. By assigning different values for each part of a drone, we could create multi-material drones.

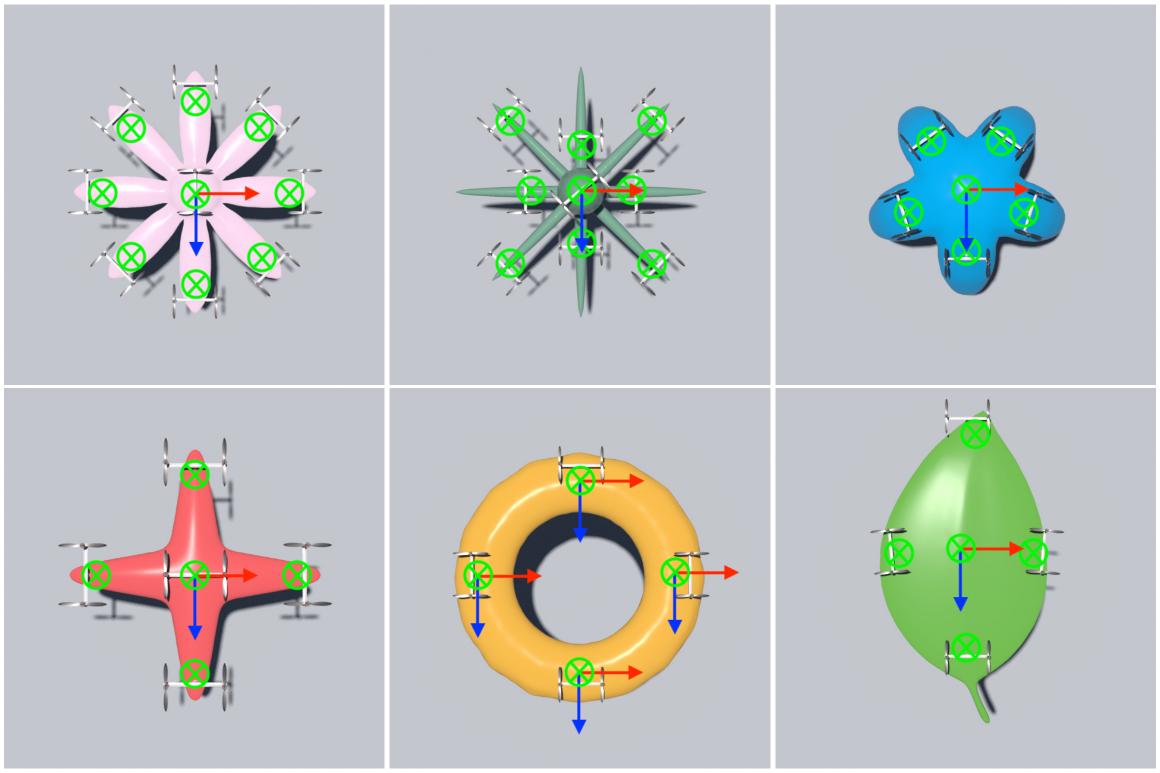


Figure 3.2: Drone's sensor placement (3D). Top row: Flower, Octopus, Orange Peel; Bottom row: Starfish, Donut, Leaf.

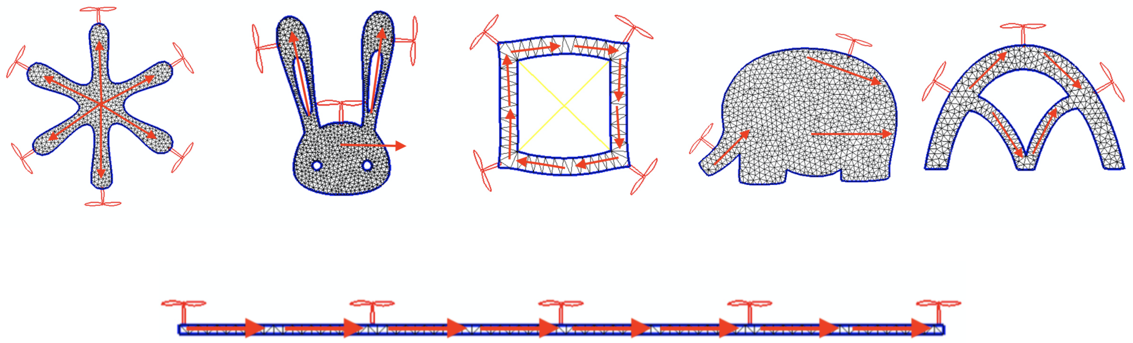


Figure 3.3: Drone's sensor placement for 2D designs (illustrated by red arrows).Top row: Engine, Bunny, Diamond, Elephant, Rainbow; Bottom row: Long Rod.

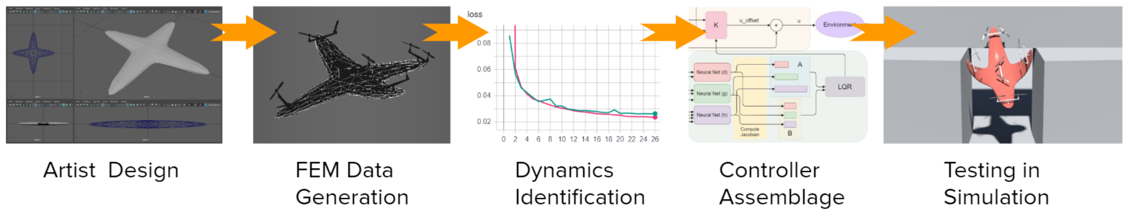


Figure 3.4: System overview: the workflow of our system consists of five stages to automate the control policy generation procedure for a soft drone design.

3.3 Dynamics Identification

As shown in Figure 3.4, our system takes soft drone geometries with customized rotor and sensor configurations as input, and returns a functional that computes full-state feedback control matrices depending on the drone's current state.

3.3.1 Geometric Representation

The design philosophy of our geometric representation is motivated by the rigid-deformable coordinate decomposition technique proposed in [Terzopoulos and Witkin, 1988] and applied in many following reduced deformable simulators [Pentland and Williams, 1989; Sorkine and Alexa, 2007; Lu et al., 2016]. The key insight is to view a soft body's deformation as a decomposition of three components: rotation, translation, and pure deformation. Mathematically, for a point $\mathbf{X}_i$ in material space, the relationship among the three components can be written as:

$$\Phi(\mathbf{X}_i) = \mathbf{R}(\mathbf{e})\mathbf{S}(\mathbf{X}_i) + \mathbf{T}(\mathbf{p}), \tag{3.2}$$

with $\mathbf{p}, \mathbf{e} \in \mathbf{R}^3$ describing the position and orientation of a local rigid frame bind to the soft body, $\mathbf{R}, \mathbf{T} \in \mathbf{R}^{3\times3}$ as the corresponding rotation and translation matrices, and $\mathbf{S}(\mathbf{X}_i)$ describes the pure deformation mapping of $\mathbf{X}_i$ within the local frame.

For a rigid drone, the state of the drone at any given time can be uniquely determined by $\mathbf{p}$, $\mathbf{e}$, and their derivatives, i.e., $\mathbf{x}^T := [\mathbf{p}, \dot{\mathbf{p}}, \mathbf{e}, \dot{\mathbf{e}}]$, by assuming $\mathbf{S}(\mathbf{X}_i) = \mathbf{X}_i$. However, for deformable drones, due to the existance of non-constant $\mathbf{S}$, the combination of $\mathbf{p}$ and $\mathbf{e}$ no longer determines the drone's configuration uniquely, in particular, since particles in deformable bodies can move independently, a single rotation matrix cannot describe the distribution of particles, thereby leaving ambiguities for information such as rotor positions, orientations, moment of inertia which all influence the drone's dynamics significantly.

As a result, we seek to extend the state space to eliminate these ambiguities. Inspired by Equation 3.2, we extend the previous state with an additional vector $\mathbf{s} \in \mathbf{R}^m$ (m can be arbitrary) that represents the deformation in body frame which $\mathbf{e}$ defines. Given that such $\mathbf{s}$ is present, we have $\mathbf{x}^T := [\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \mathbf{p}, \dot{\mathbf{p}}]$. In this work, $\dot{\mathbf{s}}$ is constituted of scalar angle values extracted from IMU measurements (see Figure 3.5 for an example).

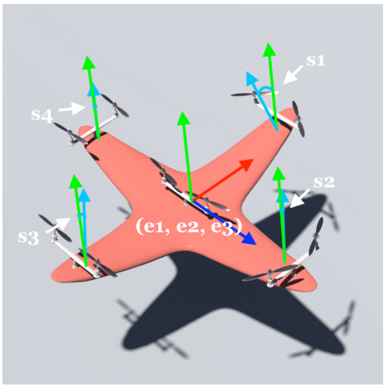


Figure 3.5: Geometric representation of a soft multicopter

The next step is to formulate the temporal evolution equations for the extended $\mathbf{x}$ with decomposed components. Here we use three new functions $\{\mathbf{d}, \mathbf{g}, \mathbf{h}\}$ to describe the temporal relations among $\{\mathbf{s}, \mathbf{e}, \mathbf{p}\}$ and the rotor thrusts $\{\mathbf{u}\}$. We make a reasonable assumption that the dynamics of pure deformation will not be influenced by rotation or translation, *i.e.*, $\dot{\mathbf{s}}_{next} = \mathbf{d}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{u})$ for some function $\mathbf{d}$; the dynamics of rotation will not be influenced by position, *i.e.* $\dot{\mathbf{e}}_{next} = \mathbf{g}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \mathbf{u})$ for some function $\mathbf{g}$; the dynamics of the position will be influenced by deformation, rotation as well as velocity *i.e.* $\dot{\mathbf{p}}_{next} = \mathbf{h}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \dot{\mathbf{p}}, \mathbf{u})$ for some function $\mathbf{h}$. Therefore, in state-space form, the dynamics are expressed as follows:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \dot{\mathbf{s}} \\ \ddot{\mathbf{s}} \\ \dot{\mathbf{e}} \\ \ddot{\mathbf{e}} \\ \dot{\mathbf{p}} \\ \ddot{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{s}} \\ \frac{\mathbf{1}}{\alpha}(\mathbf{d}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{u}) - \dot{\mathbf{s}}) \\ \dot{\mathbf{e}} \\ \frac{\mathbf{1}}{\alpha}(\mathbf{g}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \mathbf{u}) - \dot{\mathbf{e}}) \\ \dot{\mathbf{p}} \\ \frac{\mathbf{1}}{\alpha}(\mathbf{h}(\mathbf{s}, \dot{\mathbf{s}}, \mathbf{e}, \dot{\mathbf{e}}, \dot{\mathbf{p}}, \mathbf{u}) - \dot{\mathbf{p}}) \end{bmatrix}, \tag{3.3}$$

where α represents the timelapse between sensor updates.

3.3.2 Learning-based identification

We train three simple neural networks to learn $\{\mathbf{d}, \mathbf{g}, \mathbf{h}\}$, respectively. We use the residual block [He et al., 2016] with convolution layers replaced by linear layers, as previously explored by [Weinan, 2017; Lu et al., 2018]. We do not use any normalization throughout the networks. The neural networks consist of four residual blocks followed by one linear layer. All three functions share the same network architecture but different parameter weights.

The training data is generated with our implementation of a Finite Element simulator. Given a drone geometry, we initialize the drone as undeformed, lying at the origin, and apply a random thrust to each rotor and observe the drone's position, rotation and deformation at 100 Hz. Each set of random thrust is applied for 0.6 seconds. Other data generation schemes we tried also consist of using a rigid LQR controller to generate the thrusts, or apply a different random thrust each frame, but the former yields poor test loss due to the confined distribution of LQR control outputs, while the latter were too noisy to train. The insight that we need to give the system enough time to respond to a signal and display meaningful behavior is

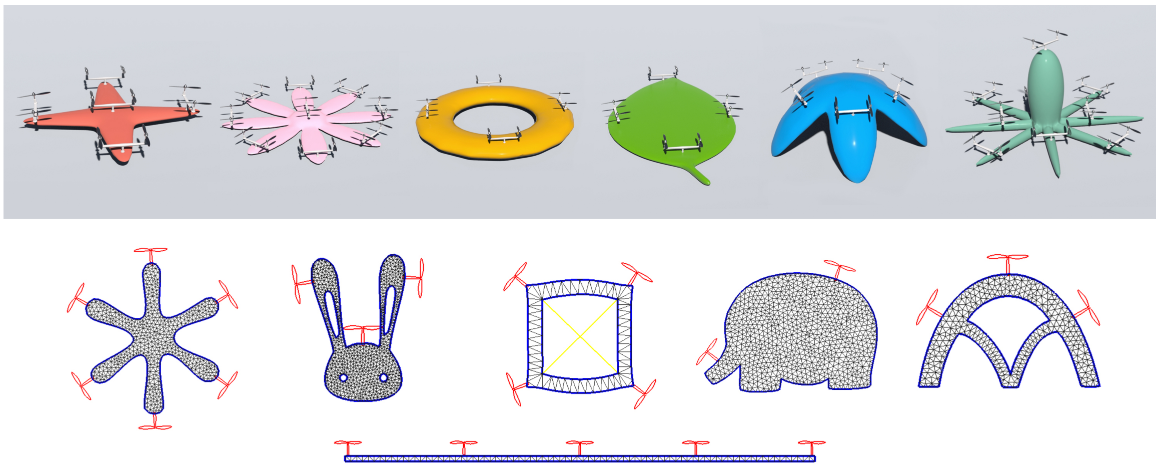


Figure 3.6: 3D (top) and 2D (bottom) soft drone designs with unconventional shapes and rotor layouts

inspired by [Holl et al., 2020]. We refer the readers to the supplementary for the simulation and training details.

3.4 Evaluation and Results

To verify that our system can handle different soft drone designs, we developed a number of different models in both 2D and 3D that include both symmetrical and asymmetrical structures, even and odd number of rotors, single or multiple materials, with virtual springs to add material complexities (see Figure 3.6).

Locomotion Control We demonstrate that our method is able to control the locomotion of soft drones with low Young's modulus with dominating superiority to the state-of-the-art LQR controllers. For benchmark testing we implement the traditional LQR controller as if each drone is rigid in its undeformed shape. The geometry-updating LQR is enhanced with the capacity to observe the deformation of the soft drone at each control step, and update the relevant dynamics information regarding rotor position, orientation and rotational inertia accordingly. The correctness of our

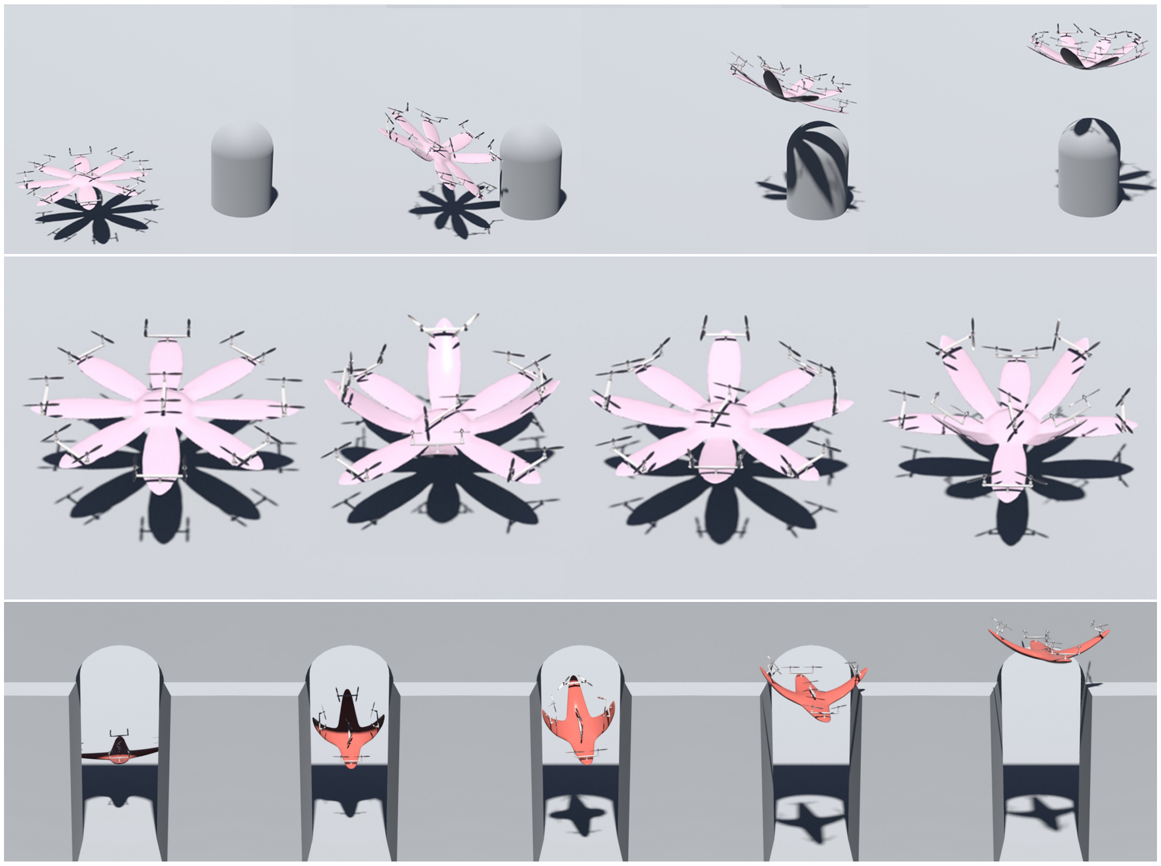


Figure 3.7: First Row: Locomotion animation; Second Row: Deformation animation; Third Row: Obstacle avoidance animation

benchmark models are verified by increasing the modulus to be 15 times as much so that the drones being tested are approximately rigid.

Target Reaching			
metrics	ours	LQR	geometry-updating LQR
survival time (s)	**20.0**	3.54	5.28
final error (m)	**0.126**	49.517	22.084
thrust usage (N)	**26740**	90596	61947

Deformation Control A talent unique to our controller is its ability to decide how the drone is shaped while controlling its locomotion at the same time. In the experiment shown in the second row of Figure 3.7, we require the flower drone to maintain at the origin while deforming into two shapes. In the first shape, which is ordered at timestep 0, the lateral pedals lie flat, while the axial pedals rise to reach an ordered angle of 1.0 radian. In the second shape, which is ordered at timestep 2000, the lateral pedals will rise while the axials will lie flat.

From a practical standpoint, the controller's ability to reconcile all three control tasks in altitude, position and deformation at once, enables it to pass through restrained terrains, maintaining a velocity and attitude while deforming in ways to reduce its width or surface area. In the example shown in Figure 3.7, the drone is challenged to pass through a hole, formed by three concrete barricades, that is narrower than its body. In order to pass through, the human pilot can order it to close up the two wings in coordination. But at the same time it needs to pitch forward while staying as close as it can to the middle without crashing into the walls on the sides. Besides, it needs to provide enough thrust to maintain a forward velocity. As one sees in Figure 3.7, our controller is able to handle this task. First, it successfully reduces its body width for over 30%; secondly, it does so while maintaining balance, allowing the drone to fly strictly in the X-Y plane with no more than 5 cm deviation in the Z-direction over the 8 second horizon.

Obstacle Avoidance Of course, the controller's ability to reconcile all three tasks at once, altitude, position and deformation control is not limited to hovering. It can also, for instance, track a certain attitude precisely, maintain a velocity and deform into certain shape. This makes it powerful enough for our drone to travel through restricted terrains by deforming in ways that reduces its length in a certain direction. In the example shown in the last row of 3.7, we explicitly require the pedal 2 and 4 to contract so that it can pass through a gap between two walls that is parallel to the x axis and emerge out of it at the top. So the controller need to reconcile three different tasks. First it needs to close up the two wings in coordination. Second it needs to tile sideways. Third it needs to provide enough thrust to maintain a motion along the x,y axis, but reduce motion along the z axis, which will make it bump into the wall. As seen in the pictures, our controller does a satisfactory job. First it successfully contracted its wings, reducing its width along its body frame z axis for over 30%. Secondly it did so while maintaining balance, allowing the drone to fly strictly in the +x, +y direction, while only causing 5 cm of deviation in the z axis over the 8 second long passage, as seen in the middle graph.

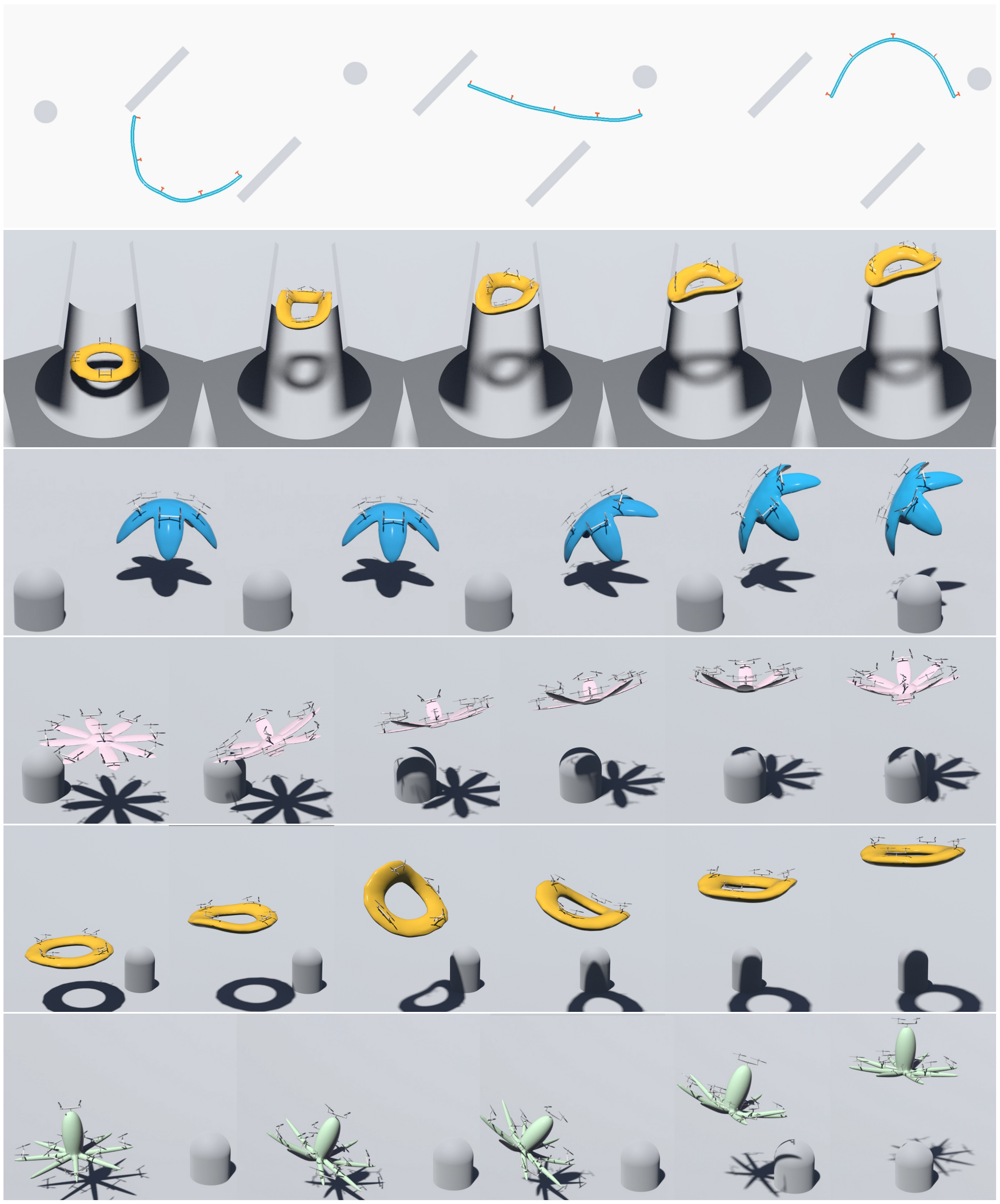


Figure 3.8: Visualization of more test results; Top 2: Obstacle avoidance animation; Bottom: Locomotion animation;

Chapter 4

Learning multi-body constraint systems

4.1 Position Based Framework

The key idea of a standard position-based framework is to move (or ”project”) a set of points according to a constraint to satisfy the constraint.

According to Macklin and Müller’s position based fluid [Macklin and Müller, 2013], for a fluid system, the constraint is defined as a constant density for each particle at each time step to enforce the incompressibility. Our algorithm shares the same workflow with PBF to update the predicted position iteratively until the density for each particle reaches the desired value.

Algorithm 1 Simulation loop

```
1: for particles $i$ do
2:     Apply external forces $\mathbf{v_i^*} \leftarrow \mathbf{v_i} + \Delta t\mathbf{a}$
3:     Predict position $\mathbf{x_i^*} \leftarrow \mathbf{x_i} + \Delta t\mathbf{v_i^*}$
4: end for
5: while $iter < solverIterations$ do
6:     for particles $i$ do
7:         Find Neighboring particles Ni
8:         Calculate $\lambda_i$
9:     end for
10:    for particles $i$ do
11:        Calculate $\Delta p_i$
12:        Update position $\mathbf{x_i^*} \leftarrow position\mathbf{x_i^*} + \mathbf{\Delta p_i}$
13:    end for
14: end while
15: for particles $i$ do
16:     Update velocity $\mathbf{v_i} \leftarrow \frac{(\mathbf{x_i^*} - \mathbf{x_i})}{\Delta t}$
17:     Apply viscosity and update $\mathbf{v_i}$
18:     Update position $\mathbf{x_i} \leftarrow \mathbf{x_i^*} + \Delta\mathbf{p_i}$
19: end for
```

4.2 Fluid with a Constraint

4.2.1 Apply the Constraint in Simulation

In our simulation process, to update the position, we follow the algorithm in PBF.

$$\rho_i = \sum_{j \in P} m_j W(p_i - p_j, h) \tag{4.1}$$

$$C_i(p_1, ..., p_n) = \frac{\rho_i}{\rho_0} - 1 \tag{4.2}$$

For kernel function W, Poly6 is used:

$$W_{poly6}(P_i - P_j, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - |P_i - P_j|^2)^3, & if |P_i - P_j| < h \\ 0, & otherwise \end{cases} \tag{4.3}$$

where h is the cutoff for W. Particles would be regarded as neighbors only if their distances are closer than h.

The mass for every particle is the same. Changing particle mass without changing the rest density would change the rest distance among particles.

To ensure that $C_i \approx 0$ all the time, given P, we want to find a ΔP along the gradient that satisfies:

$$C(P + \Delta P) = 0 \tag{4.4}$$

$$\approx C(P) + \nabla C^T \Delta P \tag{4.5}$$

$$\approx C(P) + \nabla C^T \nabla C_p \lambda \tag{4.6}$$

According to these equations, λ could be calculated as:

$$\lambda = -\frac{C_i(p_0, ..., p_n)}{\sum_k |\nabla_{p_k} C_i|^2 + \epsilon} \tag{4.7}$$

In practice, we use autograd of PyTorch to compute the **gradient of relative positions**. Only the particle itself and its neighbors would contribute to the constraint. The gradient of other particles would be zero.

Then the updated positions are calculated as:

$$\Delta p_i = \frac{1}{\rho_0} \sum_{j \subset N_i} (\lambda_i + \lambda_j) \nabla W(p_i - p_j, h)/2 \tag{4.8}$$

When there are not enough particles around, clustering will happen to satisfy the constraint (especially at boundaries). An artificial pressure term is incorporated to avoid particle collision, creates surface tension, and lower the neighborhood requirements when updating positions:

$$\Delta p_i = \frac{1}{\rho_0} \sum_{j \subset N_i} (\lambda_i + \lambda_j + s_{corr}) \nabla W(p_i - p_j, h)/2 \tag{4.9}$$

$$s_{corr} = -k\left(\frac{W(p_i - p_j, h)}{W(\Delta q, h)}\right)^n \tag{4.10}$$

where $|\Delta q| = 0.1h$, k = 0.1, and n = 4.

The artificial pressure term prevents the constraint (defined in Equation 4.1) to literally become 0, as well as introduce difficulties in designing a general network to learn the constraint without strong prior knowledge, which would be discussed in later sections.

To validate this workflow, we initialize particles on a 8*8 grid. We then apply a gravity that is always pointing to the origin from everywhere in the space. By this means we get rid of container representation and collision calculation, and focus only on the interaction of particles themselves.

The simulation result is shown in Figure 4.2.1 The surface tension created by the would make sure that the overall shape of the fluid dynamics would become circle-like after enough frames, and the gravity would move the fluid to the origin.

We then create training data by generating particles on a grid with random offsets that are smaller than half-cell length. The initial positions of the grids are also randomized.

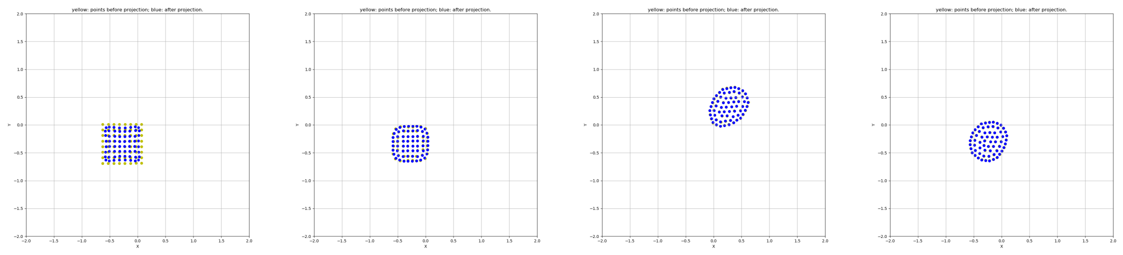


Figure 4.1: Frame 1, 3, 50 and 100 of the GIF generated by the simulator. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

4.2.2 Learning the Constraint

In this section, we will discuss how we design the network.

Basically, we are trying to learn how the **relative positions** of the neighbors of a particle contribute to its constraint.The input data is the predicted positions given in line 3 of algorithm 1. The output is the updated positions, which should satisfy the constraint.

Learn the kernel function

In the first experiment, to make sure that the network is able to learn the constraint, the label we use to compute loss is given by equation (8), which **is not using the correction term**. This makes sure that we already know the explicit constraint computing function, and the constraint is 0 after iterations for each projection.

In this case, we use the network to replace equation 3. In other words, the network is learning the kernel function directly.

During the process, we also found that using the relative position converges slowly, and the result shows a projection preference of axis and direction. Using the distance gives us a better result.

However, this method is not using the actual simulation data, and learning only

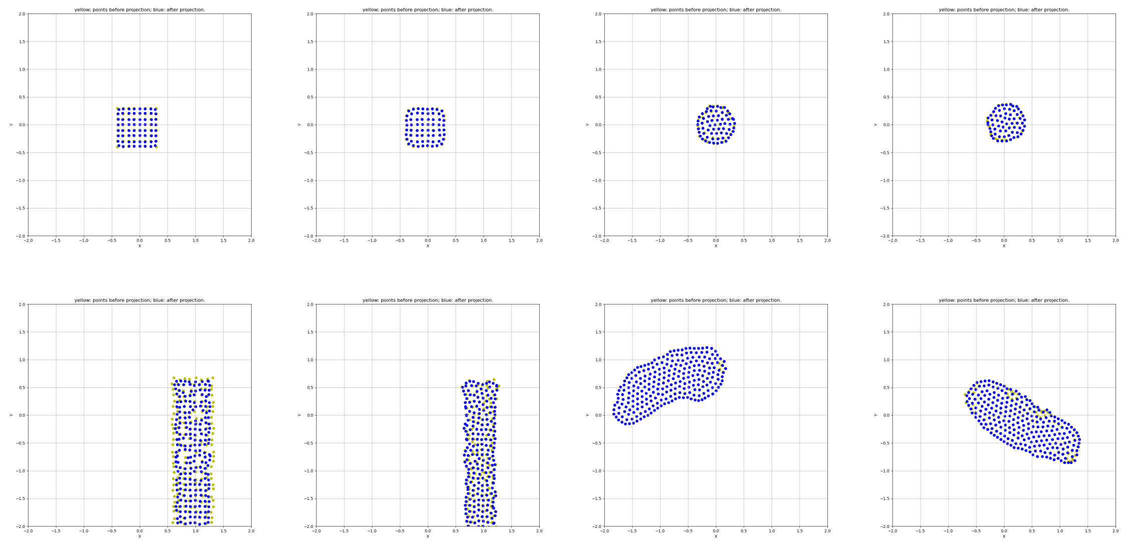


Figure 4.2: Frame 1, 3, 50 and 100 of the GIF generated by the learning both the kernel function and the correction; 64 particles in the Top line, 256 particles in the Bottom line. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

the kernel function is too boring.

Learn the kernel function and the correction

We use two networks to calculate the result. One network learns the kernel function (by using the distance directly), and the other network uses w to learn the correction term.

Figure 4.2.2 shows that it can actually learn the correction term in this way. Other visualization show that the result works for different number of particles.

By using the distance information directly instead of using the relative position, the result is less likely to show a preference of direction (e.g. some previous networks would move the particles more along y-axis than along x-axis).

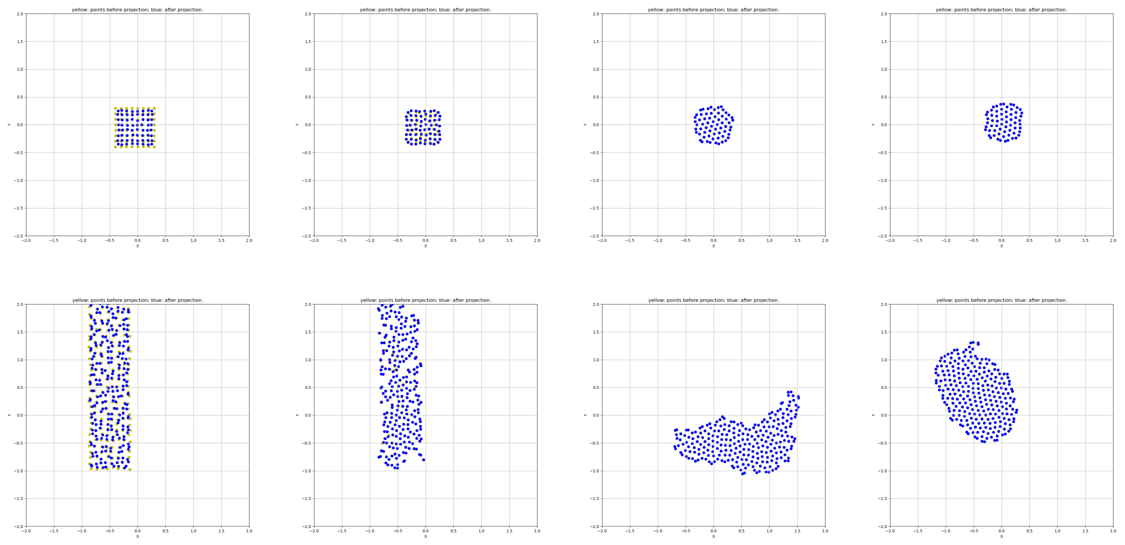


Figure 4.3: Frame 1, 3, 50 and 100 of the GIF generated by learning an abstract constraint. 64 particles in the Top line, 256 particles in the Bottom line. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

Find an abstract constraint

This method tries to learn a single constraint that is more representative than density, and could define the fluid directly without the artificial pressure term.

Our first attempt is to get rid of the artificial pressure term directly, and everything else stays the same as the previous network. This didn't give us any satisfying result.

With the information of the original position before applying velocity, the movement of particles becomes more stable even without viscosity.

Again, by directly using the relative positions, the result does not converge, but by using the distances directly we could get a much better result.The problem occurs at the boundaries – particles are clustering and the surface tension is not correctly learned.

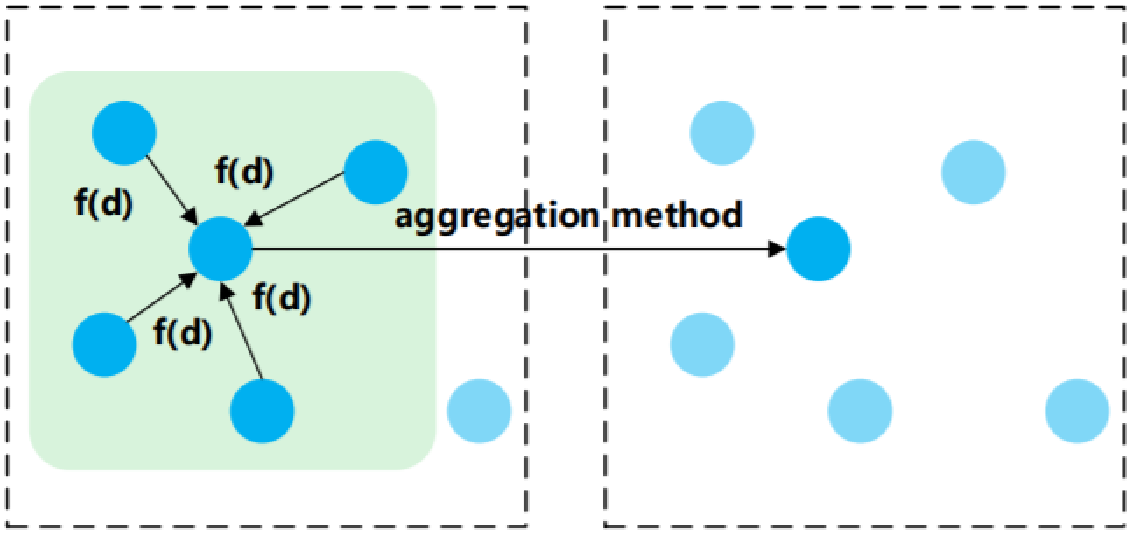


Figure 4.4: A Classic Message-passing GNN framework

Implementation details

Using GNN

Previous section strictly follows the workflow of PBF, and the result is close to the ground truth. However, there are two main drawbacks: 1. too much prior knowledge is given, and the projection function is too complicated; 2. the projection is only about positions; viscosity is not taken into consideration.

In this section, we explored methods to solve the issues. To make the projection function more scalable, we use GNN to learn the fluid dynamics; Similar layers are used to learn the pair-wise relationship. To learn the viscosity at the same time, we tried to learn the velocity by adding more features to our input.

The workflow of aggregating information from neighbors and updating a feature is very similar to a graph neural network (GNN) [Scarselli et al., 2009], and can be directly migrated to a GNN framework (See Figure 4.4).

By assuming that information passing among neighbors and iterations would make sure that at last the constraint is satisfied for every particle, we move a particle towards the direction that makes its own constraint approach 0.

In other words, the updated position is computed as:

$$\Delta p_i = \lambda_i \nabla C \tag{4.11}$$

We also tried to update the position towards the direction that reduce the constraint of the sum of its neighbors, but the training result failed to converge:

$$\Delta p_i = \sum_{j \in N_i} \lambda_j \nabla_i C_j \tag{4.12}$$

λ is computed as the same way in Equation 4.7.

GNN + Linear Layer

$$C1_i = \sum_{j \in N_i} f(X_i - X_j) \tag{4.13}$$

$$C2_i = g(C1_i) \tag{4.14}$$

- Edge input features: relative positions, relative velocities, distances
  Add up edge outputs for each node
- One more Layer to process the output (Linear + ReLu layers)

2 Layers of GNN

$$C1_i = \sum_{j \in N_i} f(X_i - X_j) \tag{4.15}$$

$$C2_i = \sum_{j \in N_i} g(dist_{ij}, C1_i, C1_j) \tag{4.16}$$

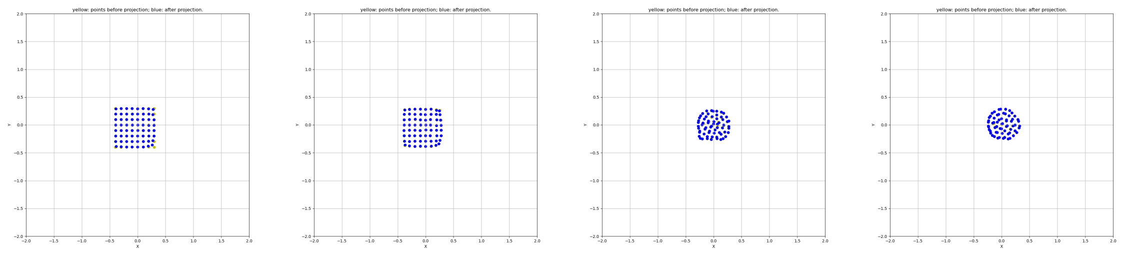


Figure 4.5: Frame 1, 3, 50 and 100 of the GIF generated by the GNN + linear layer network. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

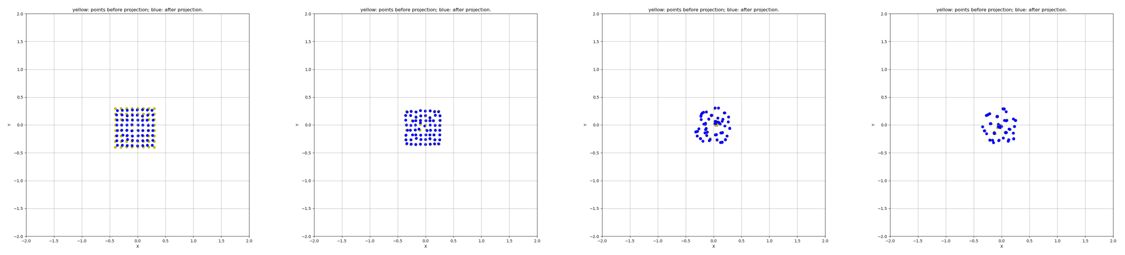


Figure 4.6: Frame 1, 3, 50 and 100 of the GIF generated by 2 Layers of GNN. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

where C1 is an intermediate feature; C2 is the output (considered as the constraint).

- Edge input features: relative positions, relative velocities

  For each node, add up outputs of adjacent edges

- Edge input features: last output of P_i, last output of P_j, distance

  Add up edge outputs for each node

As is shown in Figure 4.2.2, the correction term to avoid clustering is not learned by this method.

Considering Viscosity

During visualization, we simply subtract positions between frames and divide it by time step to calculate velocity; viscosity is not applied.

Viscosity would influence the relative velocity among neighboring particles so particles that are close to each other would be prompt to have the same velocity (but not exactly the same).

In order to take this into our consideration, we decide to use the positions predicted without considering velocity as the position to be projected), and the positions of the previous frame to be another input parameter, and then to compute the output position.
The input features becomes:

$$\hat{X}^{f+1} = 2X^{f} - X^{f-1} \tag{4.17}$$

$$X^{f+1} = P(\hat{X}^{f+1}, X^{f}) \tag{4.18}$$

No stable result is learned so far by this way. The visualization of particles could be found in Figure 4.2.2.

Implementing Details

We use L1 loss function to compute the error of the predicted position and the actual position.

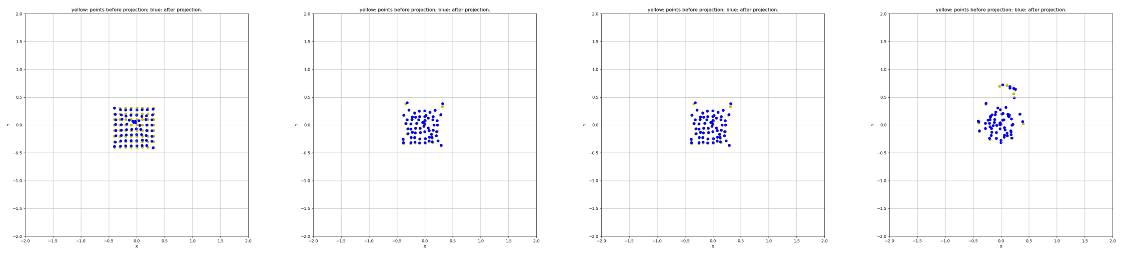


Figure 4.7: Frame 1, 3, 35 and 45 of the GIF generated by the network considering viscosity. Yellow dots shows particle positions before projections, and blue dots shows positions after projection.

4.3 Soft Body with a Constraint

With our framework, we also tried to learn the constraint of a rope. The constraint of a rope would be the bending and a fixed length between neighbors. Experiments show that our framework successfully learns the fixed-length constraint and the learned model could be applied to ropes expressed by arbitrary number of particles. However, in this case we need to specify the connection manually.

4.4 Conclusion and Future work

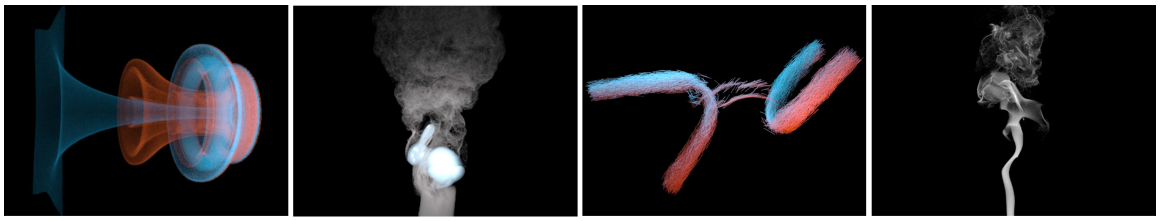
Our work shows the potential of learning fluids purely from observation data, and apply it to another fluid system with similar properties. Experiments show that the learned models work fine with different particle number, or different forces.

However, compared with other position based methods, our method is still not simple and clean enough. The network still needs some pre-computed data (e.g., distances among particles) or prior knowledge, (e.g., h value for the neighbor range).

To evaluate the efficiency of our methods, experiments with larger number of particles should be conducted.

Chapter 5

Modeling multi-body vortical flow motion



Various fluid phenomena simulated using our vortex segment method. (Far Left) Leapfrogging vortices. (Middle Left) Turbulent smoke flowing past a rotating bunny. (Middle Right) Reconnected vortex tubes from two intersecting ones. (Far Right) Cigarette smoke.

5.1 Vortex segment method

Various categories of numerical methods are developed to explore different kinds of complex and changing flow phenomena. From the perspective of computing variables, they are classified as solving velocity [Xiong and Yang, 2019a, 2020], auxiliary velocity [Kuz'min, 1983; Cortez, 1996; Saye, 2016; Hao et al., 2019], vorticity [Cot-

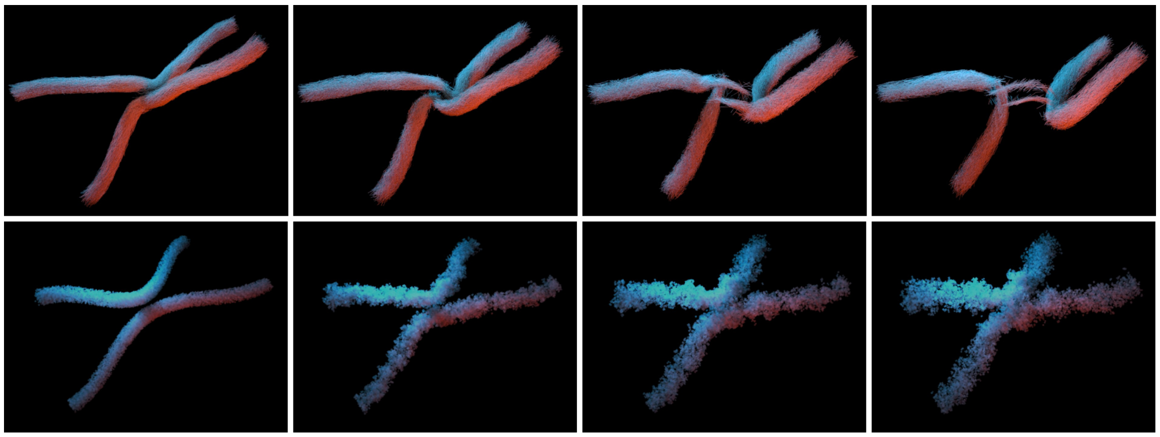


Figure 5.1: Comparison of splitting and reconnection of intersecting vortex tubes with the vortex segment method and the vortex particle method. Top/bottom 4 pictures show frames with vortex segment/particle method at 1, 100, 200 and 300 respectively.

tet and Koumoutsakos, 2000], and wave function [Chern et al., 2016, 2017], and so on. From the perspective of discretization forms, they are classified as Lagrangian method [Monaghan, 1992; Weißmann and Pinkall, 2010], Eulerian method [Rogallo, 1981; Xiao et al., 2020; Qu et al., 2019], and Lagrangian–Eulerian hybrid method [Van Rees et al., 2011; Wang et al., 2020]. With their own unique advantages, these methods can be used for numerical simulations of various flows.

The Lagrangian vortex method, a Lagrangian method based on vorticity dynamical equation, has a special advantage in the simulation of unsteady flows where the vortex structures play a leading role, such as dynamical evolution of a forced shear layer [Ghoniem and Ng, 1987], an impulsively started rigid body [Pepin, 1990], and some interacting coherent vortical structures [Weißmann and Pinkall, 2010; Padilla et al., 2019]. Firstly, this method shares all the merits of Lagrangian method, namely, with the material elements themselves being discrete elements, it doesn't have to construct complicated Eulerian meshes for flows with complex geometries. Also, it dramatically decreases numerical dissipations [Ploumhans and Winckelmans, 2000] by avoiding interpolating between grids with the physical quantities stored on material elements. Secondly, The Lagrangian vortex elements can be placed adaptively (that is, with the number of vortex elements being placed proportional to the vorticity strength) in the process of discretizing vorticity field. In the end, a corresponding relationship between discretized vorticity field and continuous incompressible velocity field can be built using Biot–Saviot law (BS law), enabling reconstructing a continuous solenoidal velocity field that fills the entire computational domain with relatively fewer discretized vortex elements needed for dynamical evolution [Wu et al., 2015].

Back in 1990, Chorin described in his pioneering work [Chorin, 1990] that *"a physical vortex is approximated by a cloud of tubular vortices."* The vortex segment method he proposed in this work, in which vorticity is carried on a set of segments and evolved by calculating their interactions, was the predecessor of the modern vortex particle method (e.g., [Cottet and Koumoutsakos, 2000]). Following this work, [Chorin, 1993] switched the data representation from discrete segments to a segment mesh, to reduce the redundant vertex storage and hence improve the computation efficiency, which laid the foundation of the modern vortex filament method [Weißmann and Pinkall, 2010]. These two pieces of classical work yield an insightful mathematical model that *"an incompressible flow can be approximated by a 'polymeric' model, which consists of an ensemble of stretched, folded, and pinched vortex tubes"* [Chorin, 1990], which serves as the motivation for our numerical paradigm design.

Motivated by Chorin's work, we devise a structure-enriched and connectivity-free Lagrangian method to model vortical flow featured by its anisotropic geometry and dynamics. Specifically, we build a generalized particle representation based on segment clouds with each particle consisting of two-point samples. From a computational perspective, discrete segments, or a generalized Lagrangian representation with each particle carrying two-point samples possess a series of inherent computational advantages when modeling anisotropic vortical flows. Numerical merits include ease of

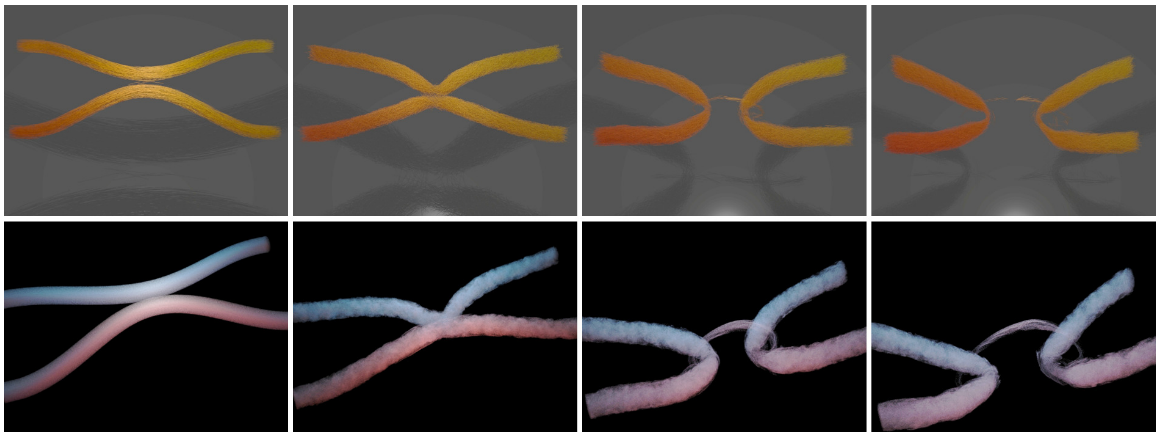
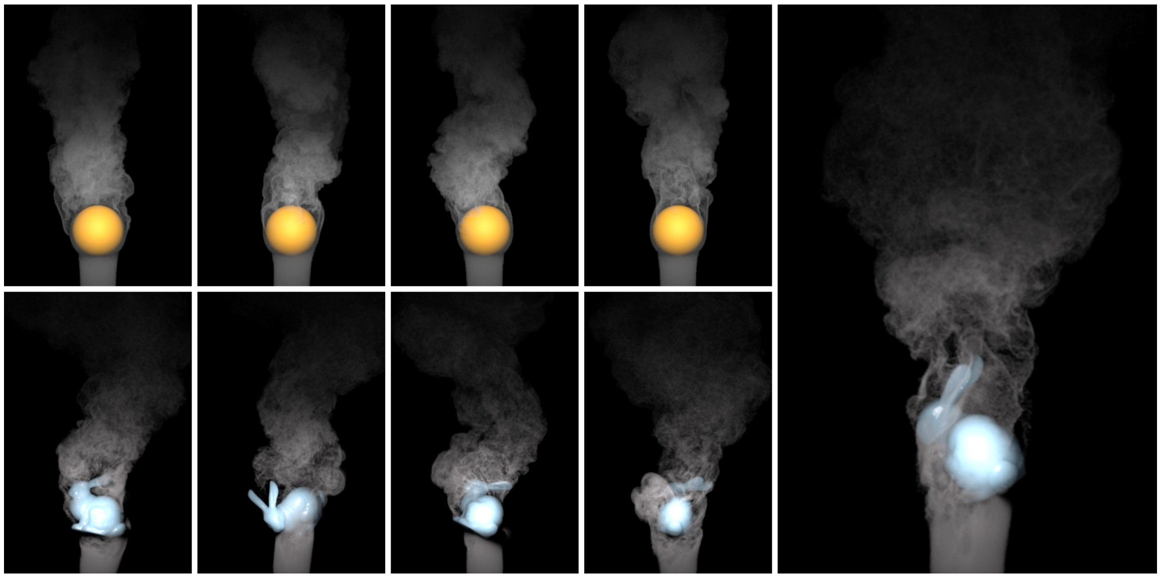


Figure 5.2: The splitting and reconnection of quasi-parallel vortex tubes. Left to right columns: frames 1, 100, 300, and 400. The pictures are visualized by vortex segments clouds (top 4 pictures) and tracer particles (bottom 4 pictures).



Turbulence above a static sphere and a rotating bunny. the top-left four pictures: the static sphere frames at 100, 200, 240, and 400, respectively; the bottom-left four pictures: the rotating bunny frames at 83, 93, 163, and 184, respectively; the right picture: the rotating bunny frame at 271.

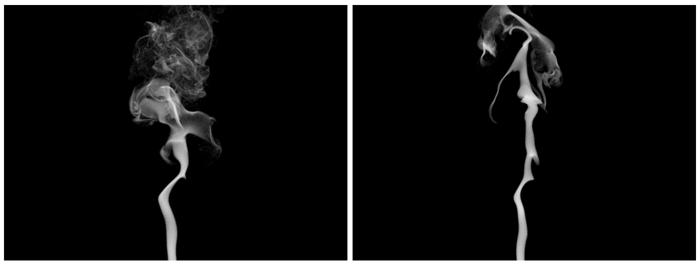


Figure 5.3: Comparison of the simulation of cigarette smoke using vortex segment and particle methods.

modeling local, vortical stretching [Zhang and Bridson, 2014], enforcing adaptivity [Fernandez et al., 1996], and to handling topological changes robustly [Weißmann and Pinkall, 2009].

To accommodate the various types of anisotropic geometrical and topological evolution on a segment cloud, we build a set of discrete reseeding operations enhanced by each segment's orientation. These reseeding operations consist of merging, splitting, and deleting, which are combined to mimic the conventional particle reseeding procedure. All of these operations leverage the anisotropic and oriented features of the segment primitives. Moreover, these operations are local, parallelizable, and connectivity-free, facilitating a high-performance code implementation to leverage the modern parallel computer architectural intricacies. As demonstrated in our examples, the proposed method accommodates the parallel computation of large-scale vortex phenomena on modern computing hardware, which boosts the capability of the method in solving strongly anisotropic and topologically complicated flows.

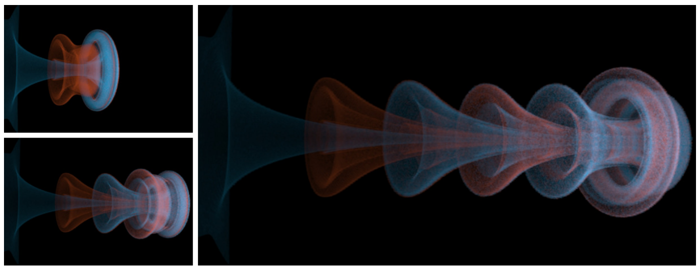


Figure 5.4: Leapfrogging vortices with vortex segments showing frames 100 (top left), 200 (bottom left), and 300 (right).

5.2 Physical Model

Considering the incompressible fluid in a domain Ω, the fluid dynamics can be described by

$$\begin{cases} \frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho}\boldsymbol{\nabla} p + \boldsymbol{f}, \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} = 0, \end{cases} \tag{5.1}$$

with proper initial and boundary conditions. Here $\boldsymbol{u}(\boldsymbol{x}, t)$ is the velocity field, $D/Dt = \partial/\partial t + \boldsymbol{u}\cdot\boldsymbol{\nabla}$ is the material derivative, t denotes the time, p is the pressure, ρ is the density, and $\boldsymbol{f}$ is the body force. If we assume ρ is a constant, taking the curl of (5.1) yields the governing equation of vorticity $\boldsymbol{\omega} = \boldsymbol{\nabla}\times\boldsymbol{u}$ as

$$\frac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega}\cdot\boldsymbol{\nabla})\boldsymbol{u} + \boldsymbol{\nabla}\times\boldsymbol{f}. \tag{5.2}$$

We discretize the vorticity field on a set of discrete vortex elements

$$\boldsymbol{\omega}(\boldsymbol{x}, t) = \sum_{j=1}^{N_v} \boldsymbol{\Gamma}_j(t) f_\delta[\boldsymbol{x} - \boldsymbol{x}_j(t)]. \tag{5.3}$$

Here N_v is the total number of vortex elements, $\boldsymbol{\Gamma}_j$ and $\boldsymbol{x}_j$ are the vorticity strength and the central position of the j^{th} vortex element. We use f_δ as a distribution function that satisfies $\int_\Omega f_\delta(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1$, describing the distribution of the vorticity around $\boldsymbol{x}_j$.

f_δ is designed as an isotropic mollification function in a conventional vortex particle method. Without loss of geometric generalities, the primitives of the vortex elements can be points (the classical vortex particle method), segments, triangles, etc.

Substituting (5.3) into the Biot–Savart (BS) law, the velocity field for vortex element convection can be obtained as

$$\boldsymbol{u}(\boldsymbol{x},t) = \boldsymbol{u}_\infty + \frac{\sum_{j=1}^{N_v} \boldsymbol{\Gamma}_j(t) \times \boldsymbol{F}_\delta(\boldsymbol{x},\boldsymbol{x}_j,t)}{2(N_d-1)\pi}, \tag{5.4}$$

with

$$\boldsymbol{F}_\delta(\boldsymbol{x},\boldsymbol{x}_j,t) = \int_\Omega \frac{(\boldsymbol{x}-\boldsymbol{x}')f_\delta[\boldsymbol{x}'-\boldsymbol{x}_j(t)]}{|\boldsymbol{x}-\boldsymbol{x}'|^{N_d}} \mathrm{d}\Omega', \tag{5.5}$$

where $\boldsymbol{u}_\infty$ is the background velocity, N_d is the dimension of the computational domain Ω, and $\mathrm{d}\Omega'$ is the volume element at $\boldsymbol{x}'$.

5.3 Discrete Vortex Segments

5.3.1 Geometric Representation

We discretize the vorticity field with a cloud of vortex segments. The information carried on each vortex segment includes the positions of the two endpoints $\boldsymbol{x}_j^\pm$, $j = 1, 2, \cdots, N_v$ and the vorticity strength magnitude Γ_j. The segment's midpoint can be calculated as $\boldsymbol{x}_j = (\boldsymbol{x}_j^+ + \boldsymbol{x}_j^-)/2$. The vorticity strength vector on each vortex segment is calculated as $\boldsymbol{\Gamma}_j = \Gamma_j(\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-)/|(\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-)|$. In addition, each vortex segment has a virtual radius $\mathcal{R}$ for numerical regularization. In two-dimensional space, a vortex segment will degenerate to a vortex point $\boldsymbol{x}_j$, in which case the vortex segment method will amount to a vortex particle method. We remark that a more precisely induced velocity around a vortex tube can be obtained with a cloud of vortex segments compared with vortex particles.

In three-dimension space, the position of the j^{th} vortex element is represented by the segment C_j. We have f_δ as a delta function supported on C_j. Substituting f_δ into (5.5) and (5.4), we obtain the induced velocity of the j^{th} vortex element with regard to a spatial point $\boldsymbol{x}$ as:

$$\begin{aligned}\boldsymbol{u}_j^{BS}(\boldsymbol{x}) =& \frac{\Gamma_j}{4\pi}\left(\frac{\boldsymbol{x}_j^+ - \boldsymbol{x}}{\|\boldsymbol{x}_j^+ - \boldsymbol{x}\| + \mathcal{R}} - \frac{\boldsymbol{x}_j^- - \boldsymbol{x}}{\|\boldsymbol{x}_j^- - \boldsymbol{x}\| + \mathcal{R}}\right) \\ &\cdot (\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-)\frac{(\boldsymbol{x}_j^- - \boldsymbol{x}) \times (\boldsymbol{x}_j^+ - \boldsymbol{x})}{\|(\boldsymbol{x}_j^- - \boldsymbol{x}) \times (\boldsymbol{x}_j^+ - \boldsymbol{x})\|^2 + \mathcal{R}^2},\end{aligned} \tag{5.6}$$

where $\mathcal{R}$ is a small positive number for regularization. This formula is presented as the form of an analytical expression in [Weißmann and Pinkall, 2010], while the discrete form is given in [Padilla et al., 2019]. For 2D cases, $f_\delta(\boldsymbol{x})$ becomes a conventional Dirac delta function $\delta(\boldsymbol{x})$, and the induced velocity becomes

$$\boldsymbol{u}_j^{BS}(\boldsymbol{x}) = \frac{\Gamma_j}{2\pi}\frac{\boldsymbol{e}_z \times (\boldsymbol{x} - \boldsymbol{x}_j)}{\|\boldsymbol{x} - \boldsymbol{x}_j\|^2 + \mathcal{R}^2}, \tag{5.7}$$

where $\boldsymbol{e}_z$ is the normal direction of the 2D plane. We take the summation of induced velocities of all the vortex elements and the background velocity $\boldsymbol{u}_\infty$ to calculate the velocity at $\boldsymbol{x}$ as

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{j=1}^{N_v} \boldsymbol{u}_j^{BS}(\boldsymbol{x}) + \boldsymbol{u}_\infty. \tag{5.8}$$

5.3.2 Lagrangian Advection and Vortex Stretching

According to the Kelvin's circulation theorem, the vorticity strength of a vortex filament element is conservative during the action of stretching and convection:

$$\frac{\mathrm{d}\Gamma_j}{\mathrm{d}t} = \int_{D_j}\left(\frac{\mathrm{D}\boldsymbol{\omega}}{\mathrm{D}t} - \boldsymbol{\omega}\cdot\nabla\boldsymbol{u}\right)\cdot \mathrm{d}\boldsymbol{S} = 0, \tag{5.9}$$

where D_j is the cross-section of the vortex segment in the direction of the vorticity. Thus, neither advection nor stretching changes the vorticity strength of the vortex filament. Therefore, naively updating the position of the endpoints of each vortex segment without considering the radius change of the vortex segment

$$\frac{\mathrm{d}\boldsymbol{x}_j^{\pm}}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_j^{\pm}) \tag{5.10}$$

can update the vortex element with both advection and stretching. Without considering reseeding, the position and the length of a vortex segment will change during its evolution, but its shape will remain straight.

There is no vorticity stretching in two-dimensional space. Hence, the 2D vortex convection can be simplified as

$$\frac{\mathrm{d}\boldsymbol{x}_j}{\mathrm{d}t} = \boldsymbol{u}(\boldsymbol{x}_j). \tag{5.11}$$

Similar ideas of processing vortex stretching using an explicit segment representation can also be found in the previous work of [Zhang and Bridson, 2014], where virtual segments were created on a background grid in every time step to measure the local stretching effects. Compared with this hybrid representation, our segment cloud method fully leverages the vorticity expressiveness of discrete segments and naturally evolves the system's motion in a pure Lagrangian way.

5.3.3 Topological Changes with Segments

One of the most salient features of our segment cloud method is its capability of processing local topological changes with simple and parallel segment operations. Motivated by [Chorin, 1990] on removing hairpin segments and the various particle reseeding and local re-meshing techniques in computer graphics (e.g., [Ferstl et al.,

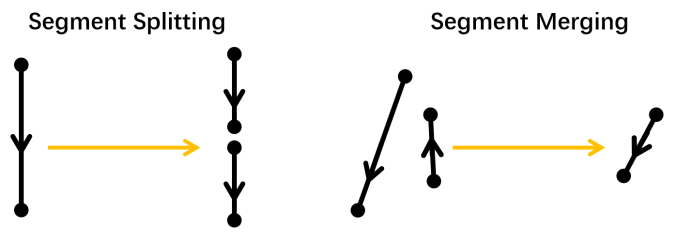


Figure 5.5: Splitting and merging of vortex elements

2016; Wang et al., 2020]), we devise three local segment reseeding operations including segment splitting, merging, and deletion. We showcase that the combination of these three operations can facilitate our simulation system to automatically handle complicated topological changes of vortical fluid such as vortex tube reconnection. At the same time, these segment operations enable our system to always maintain a reasonable number of the segment during the simulation.

Segment splitting We employ a segment splitting operator (see the left of Figure 5.5) as splitting a segment into two as they both keep stretching. By setting a max length threshold for a segment, we split it into two new segments with ends when the segment with ends $(\boldsymbol{x}_j^-, \boldsymbol{x}_j^+)$ is greater than the threshold:

$$[\boldsymbol{x}_j^-, (\boldsymbol{x}_j^+ + \boldsymbol{x}_j^-)/2] \text{ and } [(\boldsymbol{x}_j^+ + \boldsymbol{x}_j^-)/2, \boldsymbol{x}_j^+]. \tag{5.12}$$

Segment merging We devise a segment merging operation (see the right of Figure 5.5) to avoid two parallel segments getting too close. We check two criteria before merging a pair of segments. First, we check if the central positions of the two segments are close enough (i.e. the absolute value $|\boldsymbol{x}_i - \boldsymbol{x}_j| < \lambda$ with λ as a threshold). Second, we check if the vorticity directions of the two segments are almost opposing each other. In particular, we check if the angle between the two vorticity vectors (specified by the vorticity magnitude and the segment endpoints) is almost π (i.e.

$\boldsymbol{x}_i \cdot \boldsymbol{x}_j / |\boldsymbol{x}_i||\boldsymbol{x}_j| < \cos\theta$ with θ as a given threshold). We will merge two segments if they are both close and their vorticities are pointing to roughly opposite directions. The center, length, and vorticity of the merged segment are updated by the average of the quantities stored on the two original segments as

$$\begin{cases} \boldsymbol{x}_{ij} = \dfrac{\boldsymbol{x}_i + \boldsymbol{x}_j}{2} \\ L_{ij} = \dfrac{|\boldsymbol{x}_i^+ - \boldsymbol{x}_i^-| + |\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-|}{2} \\ \boldsymbol{\Gamma}_{ij} = \dfrac{2[\Gamma_i(\boldsymbol{x}_i^+ - \boldsymbol{x}_i^-) + \Gamma_j(\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-)]}{|\boldsymbol{x}_i^+ - \boldsymbol{x}_i^-| + |\boldsymbol{x}_j^+ - \boldsymbol{x}_j^-|}. \end{cases} \tag{5.13}$$

Segment deletion We delete a segment if its vorticity falls below a threshold.

Accommodated by these operations, the realization of disconnection and reconnection of vortex tubes can happen automatically while stabilizing the computational algorithm. We demonstrate the efficacy of the combination of these two basic segment operations by simulating the various vortex tube reconnection examples (see Figures 5.1 and 5.2) that are infeasible for a pure particle method or potentially complicated for a mesh-based method to process their topological changes.

5.4 Temporal evolution

We initialize each segment's position randomly and its vorticity strength based on a given vorticity distribution. We use the fourth-order Runge–Kutta method for the time integration of the vortex segments.

We carry out the temporal evolution of the vortex segment cloud using the following steps:

(a) Calculation of induced velocity using (5.8);

(b) Advection of vortex segments using (5.10) for 3D flows and (5.11) for 2D flows;

(c) Splitting of vortex segments using (5.12) for 3D flows;

(d) Merging of vortex segments using (5.13) for 3D flows;

(e) We delete segments if their vorticity falls below a threshold.

5.5 Visualization of Vortex Segments

To visualize our results, we take the output of our simulation and rendered them in Houdini as smokes. The simulation results are represented as a set of particles. For most of the examples, the only property of the particles is the three dimensional vector – position. For the leapfrogging example, we add another property that specifies which vortex ring the particle originally belongs to. Different colors are assigned to the rings.

Given the particles, we create a volume and rasterize it to be voxels with density. We adjust a set of parameters including particle size, voxel size and filter size, etc. to find a desired realistic result.

During the rendering process, we meet several challenges:

- The rendering process could be very slow to get pictures with a good quality
- The smoke could either be too noisy or too blurry

We found that the number of ray samples would not affect the quality of the picture too much, and the number of samples per pixel would also have little influence over the picture quality after a threshold. To make sure that the picture is not too noisy nor too blurry, we need to make sure that we have enough points to represent the smoke. The best number of particles could vary since the structure of the smoke is different. If the smoke forms certain shapes, the number could be smaller; if the smoke fills the whole space, the number would be larger. We choose the best configurations based on observations in real life. We adjust the configuration for each example to make sure that we get reasonable results in an acceptable amount of time.

5.6 Results

Vortex tube reconnection Figures 5.1 and 5.2 show the splitting and reconnection of two intersecting and quasi-parallel vortex tubes forming another two separated U-shaped tubes [Beardsell et al., 2016; van Rees et al., 2012]. This simulation captures the main topological changes and the pinched-off vortex filaments using a mesh-free Lagrangian method, which was not feasible for any previous particle-based methods. In particular, as shown in Figure 5.1, we compared our results with the ones obtained by a conventional vortex particle method, which failed to capture such highly anisotropic and topologically complicated phenomena. This comparison showcased our method's unique ability in modeling the topological transitions and evolution of complicated vortex flow.

Vortices interacting with solids Figure 5.1 demonstrates the upward rising smoke passing a static sphere and a rotating bunny swinging from side to side forming a waving wake flow. We can see that our vortex segment method can capture the boundary vortical details effectively and transport these vortices with the advected segments in the flow field.

Cigarette smoke The simulations of rising cigarette smoke in Figure 5.3 show a comparison between vortex segments and vortex particles. In each time step, a small number of vortex elements in both examples rise from the bottom with an average velocity forming a smoke effect as the vortex elements rising with strength gradually decay to zero. The number of the simulation particles in the computation domain converges to around 400. The small-scale vortical flow details are well-preserved in the simulation produced by the segment method. In contrast, the simulation using the particle method shows less turbulent features due to the insufficient amount of particles being used.

Leapfrogging vortices As shown in Figure 5.4, two vortex rings are initialized using the same in-plane vorticity strength and different radii. The interaction between the two rings delivers a leapfrogging motion along the common axis. The bigger ring with smaller self-induced velocity shrinks and accelerates due to their mutual interaction and the smaller ring with larger self-induced velocity widens and decelerates. The rear decrescent ring then passes through the leading enlarged ring. We show that we can capture the leaping dynamics of the vortex rings in a long-term stable fashion without maintaining the segment connectivities as in a conventional vortex filament method.

Chapter 6

Conclusion

The thesis focuses on challenges of the design, modeling and control of soft dynamic systems. We show the workflow of designing a soft drone as well as providing a tool to customize a 2D drone by sketching. We then learn the soft-body controller through simulation data, decomposing the state space of the soft drone into three physics-based geometric variables, and train three respective neural networks to represent the dynamics of these variables. We show examples of incorporating the learned neural network dynamic system into an enhanced LQR controller to control a variety of tasks of soft drones.

For modeling, we propose a position based method that could learn a fluid dynamic from particle positions. We learn stable fluids following the framework of position based fluid, showing the network's ability to resolve complex dynamics. The model learned from a small number of particles could also be used to simulate fluid with larger number of particles. We also develop a full set of parallelizable vortex dynamics solvers with the support of dynamic solid boundaries to endorse the simulation of various challenging flow phenomena and further simulate a series of complex flows, such as the reconnection of vortex tubes.

Bibliography

Agrawal, K. and Shrivastav, P. (2015). Multi-rotors: A revolution in unmanned aerial vehicle. *International Journal of Science and Research*, 4(11):1800–1804.

Angelidis, A. (2017). Multi-scale vorticle fluids. *ACM Trans. Graph.*, 36:1–12.

Angelidis, A. and Neyret, F. (2005). Simulation of smoke based on vortex filament primitives. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–96.

Anzai, T., Zhao, M., Nozawa, S., Shi, F., Okada, K., and Inaba, M. (2018). Aerial grasping based on shape adaptive transformation by halo: Horizontal plane transformable aerial robot with closed-loop multilinks structure. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6990–6996. IEEE.

Aref, H. and Flinchem, E. (1985). Dynamics of a vortex filament in a shear flow. *J. Fluid Mech.*, 148:477–497.

Barnat, A. and Pollard, N. S. (2012). Smoke sheets for graph-structured vortex filaments. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 77–86.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510.

Beardsell, G., Dufresne, L., and Dumas, G. (2016). Investigation of the viscous reconnection phenomenon of two vortex tubes through spectral simulations. *Phys. Fluids*, 28:095103.

Bernard, P. S. (2009). Vortex filament simulation of the turbulent coflowing jet. *Phys. Fluids*, 21:025107.

Bieze, T. M., Largilliere, F., Kruszewski, A., Zhang, Z., Merzouki, R., and Duriez, C. (2018). Finite element method-based kinematics and closed-loop control of soft, continuum manipulators. *Soft robotics*, 5(3):348–364.

Bouabdallah, S., Noth, A., and Siegwart, R. (2004). Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2451–2456. IEEE.

Brochu, T., Keeler, T., and Bridson, R. (2012). Linear-time smoke animation with vortex sheet meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–95.

Chern, A., Knöppel, F., Pinkall, U., and Schröder, P. (2017). Inside fluids: Clebsch maps for visualization and processing. *ACM Trans. Graph.*, 36:142.

Chern, A., Knöppel, F., Pinkall, U., Schröder, P., and Weißmann, S. (2016). Schrödinger's smoke. *ACM Trans. Graph.*, 35:77.

Choquin, J. P. and Huberson, S. (1990). Computational experiments on interactions between numerical and physical instabilities. *Int. J. Numer. Meth. Fl.*, 11:541–553.

Chorin, A. J. (1973). Numerical study of slightly viscous flow. *J. Fluid Mech.*, 57:785–796.

Chorin, A. J. (1990). Hairpin removal in vortex interactions. *J. Comput. Phys.*, 91:1–21.

Chorin, A. J. (1993). Hairpin removal in vortex interactions ii. *J. Comput. Phys.*, 107:1–9.

Cortez, R. (1996). An impulse-based approximation of fluid motion due to boundary forces. *J. Comput. Phys.*, 123(2):341–353.

Cottet, G. H. and Koumoutsakos, P. (2000). *Vortex Methods: Theory and Practice*. Cambridge University Press.

Deng, Y., Zhang, Y., He, X., Yang, S., Tong, Y., Zhang, M., DiPietro, D., and Zhu, B. (2020). Soft multicopter control using neural dynamics identification. *arXiv preprint arXiv:2008.07689*.

Dong, W., Liu, J., Xie, Z., and Li, D. (2019). Adaptive neural network-based approximation to accelerate eulerian fluid simulation. New York, NY, USA. Association for Computing Machinery.

Du, T., Schulz, A., Zhu, B., Bickel, B., and Matusik, W. (2016). Computational multicopter design.

Eberhardt, S., Weissmann, S., Pinkall, U., and Thuerey, N. (2017). Hierarchical vorticity skeletons. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 6.

Fernandez, V. M., Zabusky, N. J., Liu, P., Bhatt, S., and Gerasoulis, A. (1996). Filament surgery and temporal grid adaptivity extensions to a parallel tree code for simulation and diagnosis in 3d vortex dynamics. In *ESAIM: Proceedings*, volume 1, pages 197–211.

Ferstl, F., Ando, R., Wojtan, C., Westermann, R., and Thuerey, N. (2016). Narrow band flip for liquid simulations. In *Computer Graphics Forum*, volume 35, pages 225–232.

Floreano, D., Mintchev, S., and Shintake, J. (2017). Foldable drones: from biology to technology. In *Bioinspiration, Biomimetics, and Bioreplication 2017*, volume 10162, page 1016203. International Society for Optics and Photonics.

George Thuruthel, T., Ansari, Y., Falotico, E., and Laschi, C. (2018). Control strategies for soft robotic manipulators: A survey. *Soft robotics*, 5(2):149–163.

Ghoniem, A. F. and Ng, K. K. (1987). Numerical study of the dynamics of a forced shear layer. *Phys. Fluids*, 30:706–721.

Hald, O. (1979). Convergence of vortex methods for euler's equations. ii. *SIAM J. Numer. Anal.*, 16:726–755.

Hald, O. and Del Prete, V. M. (1978). Convergence of vortex methods for euler's equations. *Math. Comput.*, 32:791–809.

Hao, J., Xiong, S., and Yang, Y. (2019). Tracking vortex surfaces frozen in the virtual velocity in non-ideal flows. *J. Fluid Mech.*, 863:513–544.

Hasimoto, H. (1972). A soliton on a vortex filament. *J. Fluid Mech.*, 854:477–485.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Holl, P., Koltun, V., and Thuerey, N. (2020). Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*.

Hopfinger, E., Browand, F., and Gagne, Y. (1982). Turbulence and waves in a rotating tank. *J. Fluid Mech.*, 125:505—-534.

Kim, B., Lee, S. B., Lee, J., Cho, S., Park, H., Yeom, S., and Park, S. H. (2012). A comparison among neo-hookean model, mooney-rivlin model, and ogden model for chloroprene rubber. *International Journal of Precision Engineering and Manufacturing*, 13(5):759–764.

Krasny, R. (1988). Numerical simulation of vortex sheet evolution. *Fluid Dyn. Res.*, 3:93–97.

Kulkarni, M., Nguyen, H., and Alexis, K. (2019). The reconfigurable aerial robotic chain: Shape and motion planning. *arXiv preprint arXiv:1911.10627*.

Kuz'min, G. A. (1983). Ideal incompressible hydrodynamics in terms of the vortex momentum density. *Phys. Lett. A*, 96(2):88–90.

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., and Torralba, A. (2019). Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*.

Loiseleux, T., Chomaz, J. M., and Huerre, P. (1998). The effect of swirl on jets and wakes: Linear instability of the rankine vortex with axial flow. *Phys. Fluids*, 10:1120–1134.

Lu, W., Jin, N., and Fedkiw, R. (2016). Two-way coupling of fluids to reduced deformable bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 67–76.

Lu, Y., Zhong, A., Li, Q., and Dong, B. (2018). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285.

Macklin, M. and Müller, M. (2013). Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):1–12.

Marzouk, Y. M. and Ghoniem, A. F. (2007). Vorticity structure and evolution in a transverse jet. *J. Fluid Mech.*, 575:267–305.

Mellinger, D. and Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE.

Monaghan, J. J. (1992). Smoothed particle hydrodynamics. *Annu. Rev. Astron. Astr.*, 30:543–574.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118.

Padilla, M., Chern, A., Knöppel, F., Pinkall, U., and Schröder, P. (2019). On bubble rings and ink chandeliers. *ACM Trans. Graph.*, 38(4).

Park, S. I. and Kim, M. J. (2005). Vortex fluid for gaseous phenomena. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 261–270.

Pentland, A. and Williams, J. (1989). Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 215–222.

Pepin, F. (1990). Simulation of the flow past an impulsively started cylinder using a discrete vortex method. Technical report, CALIFORNIA INST OF TECH PASADENA GRADUATE AERONAUTICAL LABS.

Pfaff, T., Thuerey, N., and Gross, M. (2012a). Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph.*, 31.

Pfaff, T., Thuerey, N., and Gross, M. (2012b). Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph.*, 31:1–8.

Ploumhans, P. and Winckelmans, G. S. (2000). Vortex methods for high-resolution simulations of viscous flow past bluff bodies of general geometry. *J. Comput. Phys.*, 165:354–406.

Qu, Z., Zhang, X., Gao, M., Jiang, C., and Chen, B. (2019). Efficient and conservative fluids using bidirectional mapping. *ACM Trans. Graph.*, 38.

Rogallo, R. S. (1981). Numerical experiments in homogeneous turbulence. In *Technical Report TM81315, NASA*.

Rosenhead, L. (1931). The formation of vortices from a surface of discontinuity. *Proc. Roy. Soc. A*, 134:170–192.

Rus, D. and Tolley, M. T. (2015). Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR.

Saye, R. (2016). Interfacial gauge methods for incompressible fluid dynamics. *Sci. Adv.*, 2:e1501869.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Schenck, C. and Fox, D. (2018). Spnets: Differentiable fluid dynamics for deep neural networks. In Billard, A., Dragan, A., Peters, J., and Morimoto, J., editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 317–335. PMLR.

She, Z.-S., Jackson, E., and Orszag, S. A. (1990). Intermittent vortex structures in homogeneous isotropic turbulence. *Nature*, 344:226–228.

Si, H. (2015). Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2).

Skallerud, B. and Haugen, B. (1999). Collapse of thin shell structures—stress resultant plasticity modelling within a co-rotated andes finite element formulation. *International Journal for Numerical Methods in Engineering*, 46(12):1961–1986.

Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116.

Spielberg, A., Zhao, A., Hu, Y., Du, T., Matusik, W., and Rus, D. (2019). Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *Advances in Neural Information Processing Systems*, pages 8284–8294.

Takami, H. (1964). A numerical experiment with disecrete vortex approximation, with reference to the rolling up of a vortex sheet. Technical report, Stanford Univ. Calif.

Tayebi, A. and McGilvray, S. (2004). Attitude stabilization of a four-rotor aerial robot. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 2, pages 1216–1221. Ieee.

Tedrake, R. (2020). Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832), downloaded on 7, 19, 2020 from http://underactuated.mit.edu/.

Terzopoulos, D. and Witkin, A. (1988). Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6):41–51.

Urbain, G., Degrave, J., Carette, B., Dambre, J., and Wyffels, F. (2017). Morphological properties of mass–spring networks for optimal locomotion learning. *Frontiers in neurorobotics*, 11:16.

van Rees, W. M., Hussain, F., and Koumoutsakos, P. (2012). Vortex tube reconnection at $Re = 10^4$. *Phys. Fluids*, 24:075105.

Van Rees, W. M., Leonard, A., Pullin, D. I., and Koumoutsakos, P. (2011). A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers. *J. Comput. Phys.*, 230:2794–2805.

Wang, H., Jin, Y., Luo, A., Yang, X., and Zhu, B. (2020). Codimensional surface tension flow using moving-least-squares particles. *ACM Trans. Graph.*, 39(4).

Wang, Z., Akiyama, K., Nonaka, K., and Sekiguchi, K. (2015). Experimental verification of the model predictive control with disturbance rejection for quadrotors. In *2015 54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pages 778–783. IEEE.

Waslander, S. L., Hoffmann, G. M., Jang, J. S., and Tomlin, C. J. (2005). Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3712–3717. IEEE.

Weinan, E. (2017). A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11.

Weißmann, S. and Pinkall, U. (2009). Real-time interactive simulation of smoke using discrete integrable vortex filaments. pages 1–10.

Weißmann, S. and Pinkall, U. (2010). Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph.*, 29:115.

Weißmann, S., Pinkall, U., and Schröder, P. (2014). Smoke Rings from Smoke. *ACM Trans. Graph.*, 33:140.

Wu, J. Z., Ma, H. Y., and Zhou, M. D. (2015). *Vortical Flows.* Springer.

Xiao, Y., Chan, S., Wang, S., Zhu, B., and Yang, X. (2020). An adaptive staggered-tilted grid for incompressible flow simulation. *ACM Trans. Graph.*, 39(6).

Xiong, S., Tao, R., Zhang, Y., Feng, F., and Zhu, B. (2021). Incompressible Flow Simulation on Vortex Segment Clouds.

Xiong, S. and Yang, Y. (2017). The boundary-constraint method for constructing vortex-surface fields. *J. Comput. Phys.*, 339:31–45.

Xiong, S. and Yang, Y. (2019a). Construction of knotted vortex tubes with the writhe-dependent helicity. *Phys. Fluids*, 31:047101.

Xiong, S. and Yang, Y. (2019b). Identifying the tangle of vortex tubes in homogeneous isotropic turbulence. *J. Fluid Mech.*, 874:952–978.

Xiong, S. and Yang, Y. (2020). Effects of twist on the evolution of knotted magnetic flux tubes. *J. Fluid Mech.*, 895(A28).

Xu, J., Du, T., Foshey, M., Li, B., Zhu, B., Schulz, A., and Matusik, W. (2019). Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12.

Yang, C., Yang, X., and Xiao, X. (2016). Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424.

Yang, S., He, X., and Zhu, B. (2020). Learning physical constraints with neural projections. *arXiv preprint arXiv:2006.12745*.

Zhang, X. and Bridson, R. (2014). A PPPM fast summation method for fluids and beyond. *ACM Trans. Graph.*, 33:1–11.

Zhao, M., Kawasaki, K., Anzai, T., Chen, X., Noda, S., Shi, F., Okada, K., and Inaba, M. (2018a). Transformable multirotor with two-dimensional multilinks: Modeling, control, and whole-body aerial manipulation. *The International Journal of Robotics Research*, 37(9):1085–1112.

Zhao, M., Shi, F., Anzai, T., Chaudhary, K., Chen, X., Okada, K., and Inaba, M. (2018b). Flight motion of passing through small opening by dragon: Transformable multilinked aerial robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4735–4742. IEEE.