Learning to Act in Partially Structured Dynamic Environment

Chen Huang,[1] Lantao Liu,[2] Gaurav Sukhatme[1]
[1]Department of Computer Science at the University of Southern California, Los Angeles, CA 90089, USA
E-mail: {huan574, gaurav}@usc.edu
[2]Intelligent Systems Engineering Department at Indiana University - Bloomington
Bloomington, IN 47408, USA. E-mail: lantao@iu.edu

Abstract

We investigate the scenario that a robot needs to reach a designated goal after taking a sequence of appropriate actions in a non-static environment that is partially structured. One application example is to control a marine vehicle to move in the ocean. The ocean environment is dynamic and the ocean waves typically result in strong disturbances that can disturb the vehicle's motion.

Modeling such dynamic environment is non-trivial, and integrating such model in the robotic motion control is particularly difficult. Fortunately, the ocean currents usually form some local patterns (e.g. vortex) and thus the environment is partially structured. The historically observed data can be used to train the robot to learn to interact with the ocean flow disturbances. In this paper we propose a method that applies the deep reinforcement learning framework to learn such partially structured complex disturbances. Our preliminary results show that, by training the robot under artificial and real ocean disturbances, the robot is able to successfully act in complex and spatiotemporal environments.

Introduction and Related Work

Acting in unstructured environments can be challenging especially when the environment is dynamic and involves continuous states. We study the goal-directed action decision-making problem where a robot's action can be disturbed by environmental disturbances such as the ocean waves or air turbulence.

To be more concrete, consider a scenario where an underwater vehicle navigates across an area of ocean over a period of a few weeks to reach a goal location. Underwater vehicles such as autonomous gliders currently in use can travel long distances but move at speeds comparable to or slower than, typical ocean currents [Wynn et al., Smith et al.]. Moreover, the disturbances caused by ocean eddies oftentimes are complex to be modeled. This is because when we navigate the underwater (or generically aquatic) vehicles, we usually consider long term and long distance missions, and during this process the ocean currents can change significantly, causing spatially and temporally varying disturbances. The ocean currents are not only complex in patterns, but are also strong in tidal forces and can easily perturb the underwater vehicle' motion, causing significantly uncertain action outcomes.

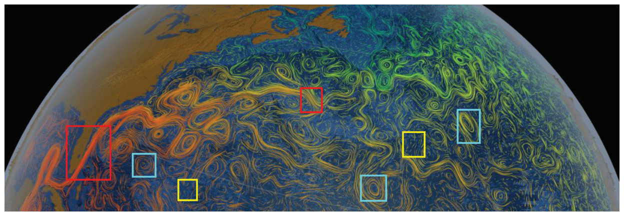


Figure 1: Ocean currents consist of local patterns (source: NASA). Red box: uniform pattern. Blue box: vortex. Yellow box: meandering

In general, such non-static and diverse disturbances are a reflection of the unstructured natural environment, and oftentimes it is very difficult to accurately formulate the complex disturbance dynamics using mathematical models. Fortunately, many disturbances caused by nature are seasonal and can be observed, and the observation data is available for some time horizons. For example, we can get the forecast, nowcast, and hindcast of the weather including the wind (air turbulence) information. Similarly, the ocean currents information can also be obtained, and using such data allows us to train the robot to learn to interact with the ocean currents.

Recently, studies on deep and reinforcement learning have revealed a great potential for addressing complex decision problems such as game playing [Mnih et al., Silver et al., Oroojlooyjadid et al.].

We found that there are certain similarities between our marine robots decision-making and the game playing scenarios if one regards the agent's interacting platform/environment here is the nature instead of a game. However, one general critical challenge that prevents robots from using deep learning is the lack of sufficient training data. Indeed, using robots to collect training data can be extremely costly (e.g., in order to get one set of marine data using on-board sensors, it is not uncommon that a marine vehicle needs to take a few days and traverse hundreds of miles). Also, modeling a vast area of environment can be computationally expensive.

Fortunately, oftentimes the complex-patterned disturbance can be characterized by local patches, where a single patch may possess a particular disturbance pattern (e.g., a vortex/ring pattern) [Oey, Ezer, and Lee], and the total number of the basic patterns are enumerable. Therefore, we are motivated by training the vehicle to learn those local patches/patterns offline so that during the real-time mission, if the disturbance is a mixture of a subset of those learned patterns, the vehicle can take advantage of what it has learned to cope with it easily, thus reducing the computation time for online action prediction and control. We use the iterative linear quadratic regulator [Li and Todorov] to model the vehicle dynamics and control, and use the policy gradient framework [Levine and Koltun] to train the network. We tested our method on simulations with both artificially created dynamic disturbances as well as from a history of ocean current data, and our preliminary results show that the trained robot achieved satisfying performance.

Technical Approach

We use the deep reinforcement learning framework to model our decision-making problem. Specifically, we use s, a to denote the robot's state and action, respectively. The input of the deep network is the disturbance information which is typically a vector field. Our goal is to obtain a stochastic form of policy $\pi_\theta(s,a) = \mathrm{P}(a|s,\theta)$ paramterized by θ (i.e., weights of the neural network) that maximizes the discounted, cumulative reward $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is a horizon term specifying the maximum time steps and r_t is the reward at time t and γ is a discounting constant between 0 and 1 that ensures the sum converges. A deep convolutional neural network is used to approximate the optimal action-value function $Q^*(s,a) = \max_{\pi} \mathrm{E}[R_t|s_t, a_t, \pi]$. More details of the basic model can be found in [Mnih et al.].

Network Design

Since the ocean currents data over a period is available, we build our neural network with an input that integrates both the ocean (environmental) and the vehicle's states. The environmental state here is a vector field representing the ocean currents (their strengths and directions). Fig. 2 shows the structure of the neural network.

Specifically, the input consists of two components: environment and vehicle states. The environment component has three channels, where the first two channels convey information of the x-axis and y-axis of the disturbance vector field. Since in the environment we need to define goal states, and there may be obstacles, thus, we use a third channel to capture such information. In greater detail, we assume that each grid of the input map has three forms: it can be occupied by obstacle (we set its value -1), or be free/empty for robot to transit to (with value 0), or be occupied by the robot (with value 1). The other component of the input is a vector that contains vehicle state information, including the vehicle's velocity and its direction towards the goal. Note that we do not include the robot's position in input because we want the robot to be sensitive only to environmental dynamics but not to specific (static) locations.

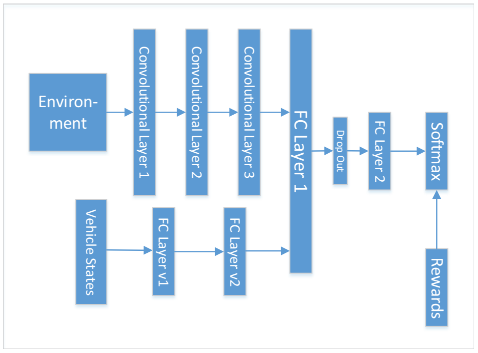


Figure 2: Neural Network Structure

The design of internal hidden layers is depicted in Fig. 2. The front 3 convolutional layers process the environment information, while the vehicle states begin to be combined starting from the first fully connected (FC) layer. The reason of such a design lies in that, the whole net could be regarded as two sub-nets that are not strongly correlated: one subnet is used to characterize features of disturbances, which is analogous to that of image classification; the other sub-net is a decision component for choosing the best action strategy. In addition, such separation of input can reduce the number of parameters so that the training process can be accelerated.

After each convolutional layer a max-pool is applied. The vehicle states will pass through 2 FC layers, and then are combined with the environmental component output from convolutional layer 3 as the input to a successive FC Layer 1. Between FC Layer 1 and 2 there exists a drop-out layer to avoid overfitting. The Softmax layer is used to normalize outputs for generating a probability distribution that can be used for sampling future actions. Additionally, the *loss funciton* is calculated using this probability distribution as well as the actual rewards.

Loss Function and Reward

We employ the policy gradient framework for solution convergence. With the stochastic policy $\pi_\theta(s,a)$ and the Q-value $Q_{\pi_\theta}(s,a)$ for the state-action pair, the policy gradient of loss function is $L(\theta)$ can be defined as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}_{\pi_\theta}\big[Q_{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(s,a)\big]. \tag{1}$$

To improve the sampling efficiency and accelerate the convergence, we adopt the *importance sampling* strategy using guided samples [Levine and Koltun].

With the objective of reaching the designated goal, our rewarding mechanism is therefore to minimize the cost from start to goal. The main idea is to reinforce with a large positive value for those correct actions that lead to reaching the goal quickly, and punish those undesired actions (e.g., those take long time or even fail to reach the goal) with small or even negative values. Formally, we define the reward r of each trial/episode as:

$$r = \begin{cases} r_s, & \text{succeeded,} \\ -(\alpha r_s + (1-\alpha) r_d), & \text{failed.} \end{cases} \tag{2}$$

where

$$r_s = \frac{1}{\sum_t \pi_\theta(s,a)||p_t - p_G||_2}, \tag{3}$$

$$r_d = 1 - e^{-D_{min}}. \tag{4}$$

where $||p_t - p_G||_2$ denotes the distance from the t-th step position to the goal state, and $D_{min} = \min_t ||p_t - p_G||_2$ is the minimum such distance along the whole path. The term r_s in Eq. (3) evaluates the state with respect to the goal state, whereas the term r_d in Eq. (4) summarizes an evaluation over the entire path. Coefficient $\alpha \in [0,1]$ is an empirical value to scale between r_s and r_d so that they contribute about the same to the total reward r. In our experiments α is set to 0.9.

Offline Training and Online Decision-Making

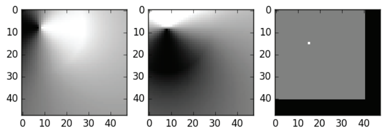
We train the robot by setting different starting and goal positions in the disturbance field, and the *experience replay* [Mnih et al., Riedmiller] mechanism is employed. Specifically, we define an *experience* as a 3-tuple (s, a, r) consisting of state s, action a, and reward r. The idea is to store those experiences obtained in the past into a dataset. Then during the reinforcement learning update process, a mini-batch of experiences is sampled from the dataset each time for training. The process of training is described in Algorithm 1, which can be summarized into four steps.

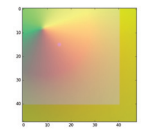
1. Following incumbent action policies, sample actions and finish a trial path or an episode.
2. Upon completion of each episode, obtain corresponding rewards (a list) according to whether the goal is reached, and assign the rewards to actions taken on that path.
3. Add all these experiences into dataset. If the dataset has exceeded the maximum limit, erase as many as the oldest ones to satisfy the capacity.
4. Sample a mini-batch of experiences from the dataset. This batch should include the most recent path. Then shuffle this batch of data and feed them into the neural network for training. If current round number is less than the max training rounds, go back to step 1.

With the offline trained results, the decision-making is straightforward: only one forward propagation of the network with small computational effort is needed. This also allows us to handle continuous motion and unknown states.

Algorithm 1: Training

```
$round \leftarrow 0$
while $round < n$ do
    Obtain reward $List\langle s,a\rangle$ of each episode.
    $experiences \leftarrow \varnothing$
    for all $\langle s,a\rangle \in List\langle s,a\rangle$ do
        $r \leftarrow get\_reward(s,a)$
        $experiences \leftarrow experiences \bigcup \langle s,a,r\rangle$
    end for
    $subset \leftarrow experiences$
    pad up $subset$ to batch size with data from dataset
    store $experiences$ into dataset
    shuffle $subset$
    feed $subset$ into neural network
    perform back propagation
    $round \leftarrow round + 1$
end while
```


(a) Input


(b) Input(mix)

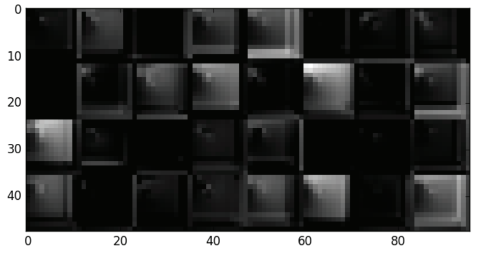

(c) Convolutional Layer 3

Figure 3: Illustration of disturbance features captured by hidden layer

Results

We validated the method in the scenario of marine robot goal-driven decision-making, where the ocean disturbances vary both spatially and temporally.The simulation environment was constructed as a two dimensional ocean surface, and the spatiotemporal ocean currents are external disturbances for the robot and are represented as a vector field, with each vector representing the water flow speed captured at a specific moment in a specific location.

The robot used in simulation is a underwater glider with a kinematic motion model with state $z = (x, y, \phi)$ including the vehicle's position and orientation in the world frame, respectively. Since the behavior of the vehicle on the 2D ocean surface is similar to that of the ground mobile robot, thus we opt to use a Dubins car model to simulate its motion. (Similar settings can be found in [Yao, Wang, and Su, Mahmoudian and Woolsey].) The dynamics can be written as:

$$\dot{x} = v\cos\phi, \quad \dot{y} = v\sin\phi, \quad \dot{\phi} = \omega, \tag{5}$$

where control inputs $u = (v, \omega)$ are the vehicle's net speed and turning rate, respectively. The dynamics are obvious nonlinear and in the discrete time case are denoted as $z_{t+1} = f(z_t, u_t)$. Such non-linear control problem can be solved using the iterative Linear Quadratic Regulator (iLQR) [Li and Todorov].

Network Training

We use Tensorflow [Abadi et al.] to build and train the network described in Fig. 2. In our experiments, the input vec-

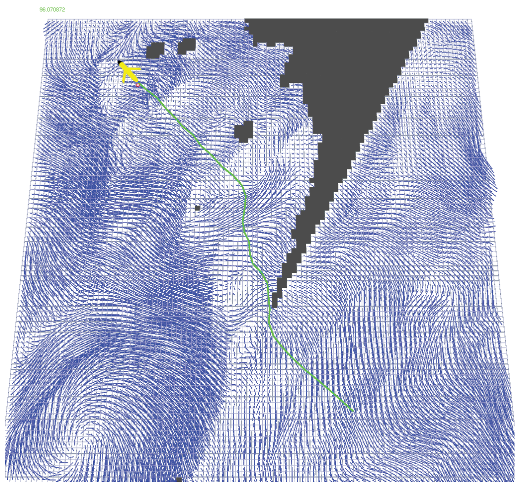


Figure 4: Demonstration of the ocean currents and a path of the robot

tor field map is 48×48, and the size of dataset for action replay is set to 10000. The learning rate is $1e-6$, and the batch size we used for each iteration is 500. We also set the length of each episode as 1000 steps.

Fig. 3 shows some features extracted from internal layers of the network. Fig. 3(a) illustrates the feature of a random disturbance vector field. Specifically, the first two channels of Fig. 3(a) are x and y components of the vector field, and the grey-scale color represents the strength of disturbance. The third channel of Fig. 3(a) is a pixel map that contains the goal point (white dot) and obstacle information (black borders).

Other grey grids denote free place. Fig. 3(b) shows a mixed view of the features, with three channels colored in red, green and blue, respectively. The picture depicts a local vortex pattern with the vortex center located near the upper left corner. Fig. 3(c) shows outputs of convolutional layer 3, from which we can observe that the hidden layers extract some local features.

Evaluations

We implemented two methods: one belongs to the control paradigm and we use the basic iLQR to compute the control inputs; the other one is the deep reinforcement learning (DRL) framework that employs the guided policy mechanism, where the policy is guided by (and combined with) the iLQR solving process [Levine and Koltun].

Artificial Disturbances We first investigate the method using artificially generated disturbances. We tested different vector fields including vortex, meandering, uniform, and centripetal patterns.

For different trials, we specify the robot with different start and goal locations, and the *goal reaching rate* is calculated by the times of success divided by total number of simulations.

The results in Table 1 show that within given time limits, both the iLQR and DRL methods lead to a good success rate, and particularly the DRL performs better in complex environments like the vortex field; whereas the iLQR framework has a slightly better performance in relatively mild environments where current speed is low, like the meander disturbance field.

Then, we test the average time costs, as shown in Table 2. The results reveal that the trials using iLQR tend to use less time than those of the DRL method. This can be due to the "idealized" artificial disturbances with simple and accurate patterns, which can be precisely handled by the traditional control methodology.

Disturbance pattern	Method	Num of trials	Num of success	Success rate
Vortex	DRL	50	48	0.96
	iLQR	50	46	0.92
Meander	DRL	50	49	0.98
	iLQR	50	50	1.00
Uniform	DRL	50	49	0.98
	iLQR	50	48	0.96
Centripetal	DRL	50	49	0.98
	iLQR	50	48	0.96

Table 1: Simulation with artificially generated disturbances

Ocean Data Disturbances In this part of evaluation, we use ocean current data obtained from the California Regional Ocean Modeling System (ROMS) [Shchepetkin and McWilliams]. The ocean data along the coast near Los Angeles is released every 6 hours and a window of 30 days of data is maintained and retrievable [Chao].

An example of ocean current surface can be visualized in Fig. 4, which also demonstrates a robot's path from executing our training result.

Because the raw ROMS ocean data covers a vast area and practically it requires several days for the robot to travel through the whole space, thus, we randomly cropped local areas to evaluate our training results. Fig. 5 demonstrates a few paths generated in such randomly selected areas.

Similar to the evaluation process for the artificial disturbances, we also looked into those aforementioned performances under the real ocean disturbances. We then evaluate the success rate and time cost, and Table. 3 shows the results (robot speed does not scale to map). Fig. 5 gives a more friendly visualization of those three areas used in our experiments. The results indicate that in most cases the DLR performs better than the basic iLQR strategy.

Fig. 5(c) and 5(d) show scenarios that can be challenging due to strong vortexes. Fig. 5(c) shows that by selecting a good path going around the vortex, the robot successfully reached the goal state. Note, in the area 3 of Fig. 5(d), a very curvy path (e.g., near the goal point) could occur due to some strong vortex in certain local areas. In this example, the ocean current around the goal area has a speed approximately equal to (or even greater than) the robot's maximal speed, but is against the robot's moving direction, so that the robot cannot easily proceed, and both DRL and iLQR eventually failed to reach the goal in this situation. A possible

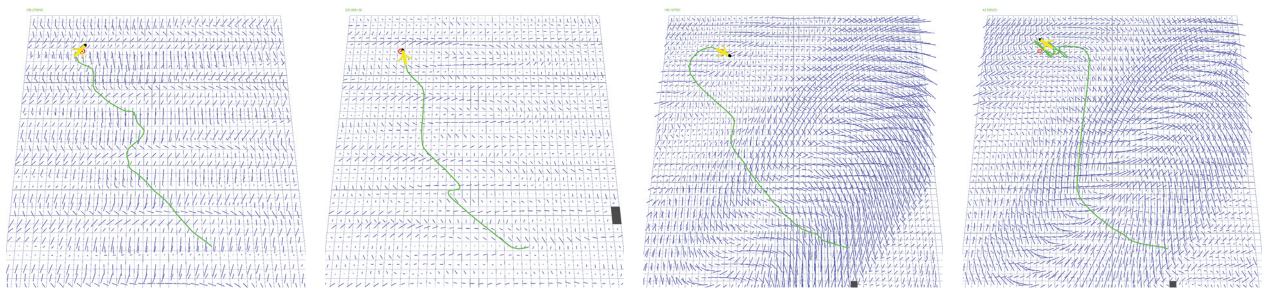

(a) area1 (b) area2 (c) area3 (d) area3

Figure 5: Examples of robot paths under different spatiotemporal disturbance patterns.

Pattern	Method	Num of trials	Average time cost
Vortex	DRL	50	20.549
	iLQR	50	14.811
Meander	DRL	50	16.926
	iLQR	50	15.367
Uniform	DRL	50	17.667
	iLQR	50	17.803
Centripetal	DRL	50	20.220
	iLQR	50	14.792

Table 2: Average time cost under artificial disturbances

Area	Method	Num of trials	Success rate	Average time cost
Area 1	DRL	15	1.00	13.787
	iLQR	15	0.93	16.375
Area 2	DRL	15	1.00	14.998
	iLQR	15	1.00	15.530
Area 3	DRL	15	0.60	22.875
	iLQR	15	0.80	19.546

Table 3: Average time cost under ocean disturbances

solution is to manipulate the robot's maximal speed to be larger (this however may be against the reality).

From Table 1 to Table 3, we can conclude that the DRL framework is particularly capable of handling complex and (partially) unstructured environments.

Conclusions

In this paper we investigate applying the deep reinforcement learning framework for robotic learning and acting in partially-structured environments. We use the scenario of marine vehicle decision-making under spatiotemporal disturbances to demonstrate and validate the framework. We show that the deep network well characterizes local features of varying disturbances. By training the robot under artificial and real ocean disturbances, our simulation results indicate that the robot is able to successfully and efficiently act in complex and partially structured environments.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Chao, Y. 2017. Regional ocean model system. http://www.sccoos.org/data/roms-3km/.

Levine, S., and Koltun, V. 2013. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 1–9.

Li, W., and Todorov, E. 2004. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, 222–229.

Mahmoudian, N., and Woolsey, C. 2008. Underwater glider motion control. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 552–557. IEEE.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Oey, L.-Y.; Ezer, T.; and Lee, H.-C. 2005. Loop current, rings and related circulation in the gulf of mexico: A review of numerical models and future challenges. *Circulation in the Gulf of Mexico: Observations and models* 31–56.

Oroojlooyjadid, A.; Nazari, M.; Snyder, L. V.; and Takác, M. 2017. A deep q-network for the beer game with partial information. *CoRR* abs/1708.05924.

Riedmiller, M. 2005. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, 317–328. Springer.

Shchepetkin, A. F., and McWilliams, J. C. 2005. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. *Ocean Modelling* 9(4):347–404.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Smith, R. N.; Schwager, M.; Smith, S. L.; Jones, B. H.; Rus, D.; and Sukhatme, G. S. 2011. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics* 28(5):714 – 741.

Wynn, R. B.; Huvenne, V. A.; Bas, T. P. L.; Murton, B. J.; Connelly, D. P.; Bett, B. J.; Ruhl, H. A.; Morris, K. J.; Peakall, J.; Parsons, D. R.; Sumner, E. J.; Darby, S. E.; Dorrell, R. M.; and Hunt, J. E. 2014. Autonomous underwater vehicles (auvs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology* 352:451 – 468.

Yao, P.; Wang, H.; and Su, Z. 2015. Uav feasible path planning based on disturbed fluid and trajectory propagation. *Chinese Journal of Aeronautics* 28(4):1163–1177.