Case Study: Safety Verification of an Unmanned Underwater Vehicle

Diego Manzanas Lopez, *Vanderbilt University*
Patrick Musau, *Vanderbilt University*
Nathaniel Hamilton, *Vanderbilt University*
Hoang-Dung Tran, *Vanderbilt University*
Taylor T. Jonhson, *Vanderbilt University*

***Abstract*—This manuscript evaluates the safety of a neural network controller that seeks to ensure that an Unmanned Underwater Vehicle (UUV) does not collide with a static object in its path. To achieve this, we utilize methods that can determine the exact output reachable set of all the UUV's components through the use of star-sets. The star-set is a computationally efficient set representation adept at characterizing large input spaces. It supports cheap and efficient computation of affine mapping operations and intersections with half-spaces. The system under consideration in this work represents a more complex system than Neural Network Control Systems (NNCS) previously considered in other works, and consists of a total of four components. Our experimental evaluation uses four different scenarios to show that our star-set based methods are scalable and can be efficiently used to analyze the safety of real-world cyber-physical systems (CPS).**

***Index Terms*—Safe AI, Data-driven Methods, Cyber-Physical Systems, Learning-Enabled Components, Autonomous Vehicles**

I. Introduction

Over the last decade, advancements in Artificial Intelligence (AI) have allowed us to re-imagine the ways we organize our cities, communicate, and move around. In fact, AI has been lauded to be one of the most influential and disruptive set of methodologies of our era. Underpinning these advancements are the success of artificial neural networks whose adroit pattern recognition competencies have allowed for technologies such as Amazon's Alexa, Apple's Siri, and DeepMind's AlphaGo to flourish and enter everyday conversation. Despite these successes, AI methodologies have not had the same level of impact in safety critical systems due to the looming apprehension that it is often impossible to identify the specific factors that led to a neural network's prediction. In other words, utilizing a "black box" model in a safety critical system, constitutes the highest form of technical debt [1]. Moreover, in a famous study, Christian Szegedy et al. demonstrated the fragility of neural network models by showing that a hardly perceptible modification to an input could cause a well-trained network to produce an incorrect classification [2].

In light of the potential to revolutionize the development of robust and intelligent systems, the last several years have seen a significant increase in the development of verification, testing, and falsification methods for systems that make use of neural networks. A comprehensive review of these techniques can be found in the survey by Liu et al. [3] and Xiang et al. [4]. While a great deal of these methods deal with networks in isolation, in recent years several methods have been proposed for verifying neural network control systems [5]–[9]. Neural network control systems commonly appear in safety critical systems where the neural network controller is generated through the use of reinforcement learning and learning by demonstration [6]. In this realm, the safety verification problem is postulated as a reachability problem. Here the challenge is to estimate the set of reachable states of the closed loop system where the plant is modeled using linear ordinary differential equations and the controller is a feed-forward neural network. Despite significant progress in neural network control systems verification, developing a scalable methodology remains a key challenge. Recently, analyses by Ivanov et al. [10] and Tran et al. [11], have explored the limits of the existing verification approaches, and the trade-offs between scalability and precision. Building on these achievements, the following work seeks to examine the efficacy of star-set based methods through the analysis of a *Neural Network Control System* that represents a more complex system than those previously considered in other works [12].

Considering this challenge, this paper examines the problem of verifying the safety of an unmanned underwater vehicle whose mission is to autonomously navigate without collisions. Our approach is based on the use of the star-set, which determines the exact output reachable set of the closed-loop system [13]. The utilization of the star-set in this work is largely due to the fact that the star-set is a computationally efficient set representation adept at characterizing large input spaces, and supports cheap and efficient computation of affine mapping operations, and intersections with half-spaces [14]. Our experimental evaluation using four different scenarios demonstrates that our star-set based methods are scalable and can be efficiently used to analyze the safety of real-world *Cyber-Physical Systems* (CPS) for up to 30 second time windows.

A. Problem Formulation

Research in recent years has demonstrated that acquiring pipeline inspection data from *Unmanned Underwater Vehicles* (UUV) offers superior data quality and consistency since UUV's often move in a highly efficient and stable manner. Thus, the unmanned methodology represents a paradigm shift in offshore geophysical survey and pipeline inspection. By utilizing a small UUV, the carbon footprint per inspection

line kilometer can be reduced dramatically. A traditional host vessel may use 10,000 - 15,000 liters of fuel per day of operations, whereas a small unmanned craft uses up to 95% less fuel [15].

In this paper, the problem we wish to consider is verifying that the UUV operates safely by successfully avoiding obstacles while executing its mission of inspecting a given pipeline. The obstacles we consider are static obstacles of varying sizes, and the safety specification we wish to consider is that the UUV does not move within a δ radius of a given obstacle. Mathematically, the safety specification, p, is satisfied if at every time step $t \in [t_0, t_f]$ in the bounded range $[t_0, t_f]$, governed by the start time t_0 and end time t_f,

$$\forall t \left(||X_v(t) - X_o(t)||_2 > \delta \right), \tag{1}$$

where X_v and X_o are the vehicle and obstacle positions, respectively.

Verification of the safety specification occurs at design time. Given an environment, an initial state and location for the UUV and an obstacle placed in front of it, our experiments evaluate the safety of the UUV.

II. System Architecture

The simulation experiments considered in this work were run using the *Robotics Operating System* (ROS) [16] and visualized using the Gazebo robot simulator [17]. The simulation package that we utilized is called the *Unmanned Underwater Vehicle Simulator* (UUVS) [18], which provides a rich set of Gazebo plugins that are used to describe the dynamics and realistic simulation environments for surface and underwater vehicles. Additionally a variety of Gazebo plugins were modified and added in an effort to more accurately capture hydrodynamic disturbance events and randomly place objects along the UUV path.

The UUV system in the Unmanned Underwater Vehicle Simulator, which we refer to as the *UUVS Model*, is made up of several components, shown in Figure 1. There is a *Learning Enabled Component* (LEC), FNN Controller, which determines the control output for each time step. It utilizes two inputs, the distance to an object and the *Closest Point of Approach* (CPA), that are calculated based on observations from the forward looking and side scan sonars. The controller then outputs 4 values, 2 regarding the heading change and 2 regarding the speed commands, which are then fed through a highly nonlinear normalization function, mapping these 4 values to a control command made up of a desired heading change (rad) and desired speed (m/s). These control commands are then processed by a low-level PID controller that converts these commands into actuation commands for the UUV.

Combining the nonlinearities of the PID, the normalization function, and the sonar sensors, makes the task of formally verifying the *UUVS Model* highly complicated. These complexities make performing safety verification via reachability analysis intractable. Therefore, we created a modified model that approximates the performance of the *UUVS Model* and allows us to perform reachability analysis more conveniently. We will refer to this model as our *Verifiable Model* and it is composed of the four parts shown in Figure 4, *Feed Forward Neural Network Sensor*, *Feed Forward Neural Network Controller*, *Feed Forward Neural Network Normalization*, and a plant model.

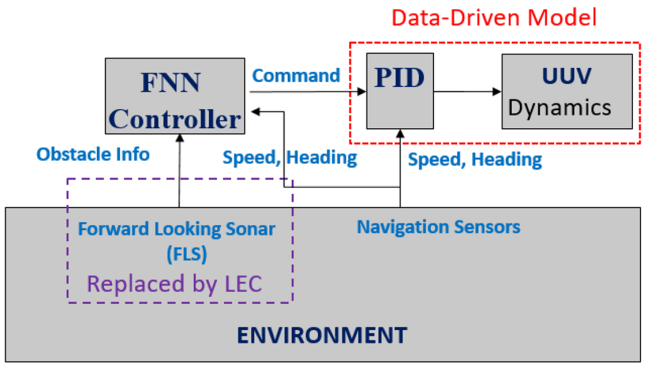

Fig. 1. UUV system architecture with the proposed modification for reachability analysis suitability with current available methods.

A. Feed Forward Neural Network Sensor

We replace the Forward Looking Sonar (FLS) with a feedforward neural network with 3 inputsm 7 hidden ReLU layers consisting of 10 neurons each, and a linear output layer with 2 neurons. This change corresponds to the purple dashed rectangle in Figure 1. From the plant's outputs and the obstacle location, the neural network is able to compute both values needed for the controller's input, i.e. the distance to the obstacle and the CPA.

B. Feed Forward Neural Network Controller

The Feed Forward Neural Network Controller (FNN Controller) is the same in the *UUVS Model* and in the *Verifiable Model*. The FNN Controller is a Reinforcement Learning controller that was trained using the Vanilla Policy Gradient Method and optimized to maximize the a reward function enforcing the certain conditions. If there is no object in the UUV's path then the UUV is expected to maintain its current heading. Any deviation from this heading results in a penalty. However, if there is an obstacle within the UUV's path then the UUV is penalized with respect to the closest point of approach and actions that change the UUV's heading are incentivized. Thus, the reward is associated with a given state, i.e. the input to the FNN Controller.

This neural network consists of 2 hidden layers of 32 fully connected, ReLU activated neurons. The two inputs are the distance to the obstacle and the closest point of approach. The linear output layer is made up of two values related to the heading change (rad) and two related to the speed (m/s).

C. Feed Forward Neural Network Normalization

The output of the FNN Controller is connected to this neural network in order to transform the 4 values provided into 2 outputs commands, the desired heading change (rad) and the desired speed (m/s). This feedforward neural network consists of 3 ReLU hidden layers with 50 neurons total and a linear

output layer with 2 neurons. In Figure 1, we add this neural network following the controller as a substitute of the highly nonlinear normalization function.

D. Plant Model

We replace the highly nonlinear UUV dynamical model with a simpler 2-dimensional linear discrete-time data-driven model obtained via system identification techniques [19]. We are able to consider a planar model (no z component) due to the constant depth that the UUVS is able to maintain due to the fixed constant depth command sent to the low-level PID controller. The data used in the system identification was obtained by performing a series of diverse maneuvers in the simulator and recording the data at fixed intervals. Because of the way the data was collected, the model of the vehicle's dynamics also includes the low-level PID controller, which further makes the reachability analysis more amenable. The result is our plant model, which is defined by a set of discrete linear difference equations of the form

$$x(t+1) = Ax(t) + Bu(t) \tag{2}$$

$$y(t) = Cx(t) + Du(t), \tag{3}$$

where t is the time step, and A,B,C and D are constant matrices of size 8x8, 8x2, 3x8, and 3x2 respectively. The model is learned using classical system identification methods [19], and treat the dynamics to be learned as a *black-box* model. Initially, none of the system parameters are known [20].

We validate our plant model with respect to the results from following the same control inputs used by the Gazebo-simulated vehicle, choosing a different scenario from the previously collected trajectories. The recorded trajectory and the respective simulated one from our data-driven model are shown in Figure 2.

The results show that, given the same inputs and initial state, our linear model captures the behavior of Gazebo vehicle very well. For simplicity, we do not include any obstacles in the graph, only the (x, y) trajectories of the simulated vehicle and the identified vehicle. Normally, an average error of $\approx 0.6m$ would be considered large. However, the UUV modeled is $2m$ long and travels at an average of $1.5m/s$. Therefore, the error is relatively small compared to the size of the problem.

III. Reachability Analysis of Neural Network Control Systems

A standard architecture of a reachability analysis problem for a *Neural Network Control System* (NNCS) is displayed in Figure 3 and is formulated as follows. Starting from an initial set of states X_0 for our model P, the controller C takes the output set of the plant Y_p as an input to compute the controller output set $U = F(Y_p)$. Then, the control set (controller output set) is applied to the plant to compute the next set of states $X_{k+1} = AX_k + BU_k$, where $k \in [0, t]$.

This process is then repeated iteratively to obtain a sequence of reachable sets of states, $X_0, X_1, X_2, ..., X_t$, where X_0 is the initial state and X_t are the reachable states at time t. To obtain precise reachable sets for the NNCS, we compute

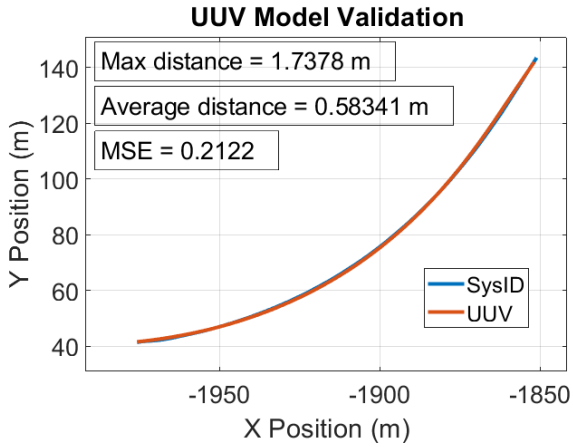

Fig. 2. Model validation: the orange line denotes the UUV trajectory recorded by the ROS simulation. The blue trajectory denotes the trajectory of the data-driven model obtained via system identification. This is the model we use in our MATLAB experiments. We include the average and maximum distance between each x-y position of the ROS trajectory and the MATLAB simulation at each point in time. We also show the mean-square error (MSE) between both trajectories.

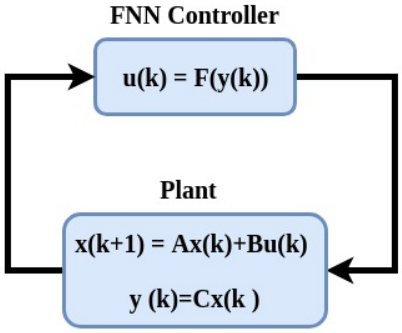

Fig. 3. Neural network control system (NNCS).

the exact control set U given the output set Y_p. Also, we compute the exact set of reachable states X_i given the previous set of states X_{i-1} and the corresponding control set U_{i-1}. To compute these exact reachable sets we utilize the star-set set representation to describe them, since the star-set is computationally less expensive than other common set representations such as polytopes and zonotopes [13]. We refer the reader to the following papers for an in-depth discussion of star-set based techniques [11], [13].

At each time step, given the states of the learned model of the UUV, the location of the obstacle, the distance to the obstacle, the closest point of approach to the obstacle with respect to the UUV trajectory, we pass these values to FNN Controller, and compute its output set. It is worth noting that if an obstacle is observed by the FLS then the distance to the obstacle is returned otherwise a constant value is returned for the closest point of approach and the distance to the object. The controller output set is fed through normalization component, the other NN of the system, which computes an output set in terms of heading change and speed. This is then fed through the model dynamics, which computes the reach sets for the outputs. From these, we only take the x and y position intervals, as these are the only ones needed to calculate the data for the feedback loop back to controller.

A. Reachability algorithm for NNCS

We are able to compute the exact reachable set of the NNCS by computing the exact control set and the exact plant output set at each timestep. However, in obtaining the reachable sets for a complex system, the number of sets increase quickly over time, making it computationally more and more expensive to obtain reachable sets for successive time steps. Therefore, this process is very time-consuming even through the use of time optimizations such as parallel computing.

To avoid the explosion of the number of sets needed to represent the reachable set, we take a single convex hull of the reachable sets of states of the plant after each step, and then we compute the the output reachable set of the plant $(Y = CX)$, which is then fed back to the next component as a single star set. However, the tradeoff here is that the solution to this problem is no longer exact.

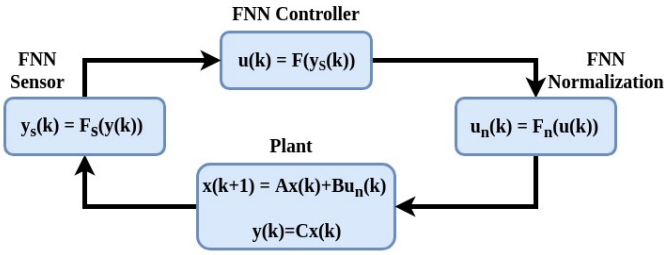

Fig. 4. Modified neural network control system for the analysis of the unmanned underwater vehicle. We compute the exact output reach set of all 3 FNN, but over-approximate the plant's output.

B. Reachability algorithm for UUV System

Our *Verifiable Model* uses three LEC's and a singular plant model connected as shown in Figure 4. Due to the increased number of LEC's in our system, we modified the general algorithm for NNCS to add all the components in the system according to Algorithm 1.

We have four main steps in the reachability analysis computation. (1) We compute the output set of the plant given the initial state. (2) Using the location of the UUV output from the plant, we compute the exact output set of the *sensor* neural network, FNN Sensor. (3) We compute the exact output set of the controller, divided into 2 operations, the main controller, FNN Controller, and the normalizing neural network, FNN Normalization. (4) We compute the set of approximate states of the plant at the next time step.

IV. Evaluation: Collision Avoidance

For our underwater vehicle system, we present several different scenarios in which we modify the initial states and the location of the obstacle to be avoided. The obstacle is located in front of the vehicle at the beginning of the experiment, and the objective is for the vehicle to detect it and avoid a collision with it.

Based on the assumption that our vehicle maintains a constant depth, we only consider the x and y positions of the obstacles. The obstacles placed in the UUV path are boxes with 1 meter long edges. The safety specification p, formally defined in equation 1, that we analyze is that the vehicle will never travel within a $\delta = 2$ meter radius of the center of the object.

Algorithm 1: Reachability Algorithm UUV System.

% k_{max}: Number of steps
% A, B, C: plant's matrices
% I: Set of initial states for the plant
% X: Set of current states for the plant
% Y: Output set of the plant
% R: Output reachable set of the plant
% F: Neural network controller
% N: Neural network normalizing output controller
% S: Neural network as sensor function
% D: Set of unsafe states

INITIALIZE
R = $cell$(1,k_{max} +1);
$R\{1,1\}$ = I;
X_1 = $R\{1,k\}$;

REACH COMPUTATION
for $k = 1 : k_{max}$ **do**
  Y_k = CX_k;
  S_{out} = S(Y_k);
  F_{out} = F(S_{out});
  N_{out} = N(F_{out});
  for j = *1:length(N_{out})* **do**
    X^j_{k+1} = AX_k + BN^j_{out};
  end
  X_{k+1} = IntervalHull(X_{k+1});
  $R\{1,k+1\}$ = X_{k+1};
  if $(X_{k+1} \cap D) \neq \emptyset$ **then**
    return;
end

Given an initial state set and an obstacle location, we calculate the output reachable sets for a k_{max} steps until the vehicle violates the safety specification or until the vehicle has cleared the obstacle and the safety specification has not been violated.

V. Results

Here we experiment with four different scenarios where the vehicle has been placed at a different location with different initial states, as well as modifying the location of the obstacle [1]. These four scenarios generally represent a wide range of the scenarios the UUV might encounter. Since the FNN Controller makes decisions based solely on the distance between the UUV and the obstacle and the CPA, these scenarios can generalized to any case where the UUV is at the same distance with the same CPA.

[1]Code to reproduce the results can be found at https://github.com/verivital/nnv/tree/master/code/nnv/examples/Submission/WAAS2020

In all four scenarios, we pick an initial point, (x, y, yaw), from a trajectory generated in UUVS. From this starting point, we estimate the initial states of the plant using MATLAB's System Identification Toolbox [20]. We take these 8 initial states[2] (estimated) and set them as the initial point for the UUV trajectory in UUVS, which is marked in blue. Additionally, the starting point of the UUV in UUVS is used to generate the no object path $(N.O.P.)$, which is the magenta dashed line representing the projected path the UUV would travel if it were to keep its orientation, (yaw), constant. Displaying the N.O.P. helps visualize how the FNN Controller directs the UUV away from the obstacle. For the reachability analysis, we take the estimated initial states and create lower and upper bounds around these points by adding -0.0001 and +0.0001 to each state respectively. The blue boxes in the result figures visualize the reachable states at each time step.

A. Experiment 1: A Minor Course Correction

In Experiment 1, shown in Figure 5, the UUV starts with a trajectory that does not intersect with the obstacle. Therefore, little to no course correction is needed to avoid the obstacle.

We observe this behavior in Figure 5 as the UUV meets the specification and clears the obstacle after 24 seconds.

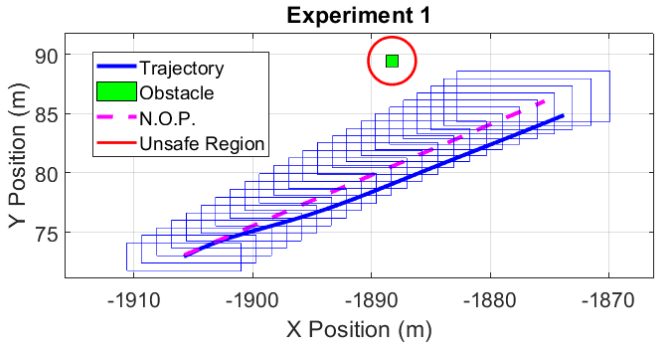

Fig. 5. The UUV is initialized at *(-1905.577, 73.085)* pointing *23.115* degrees from horizontal. The obstacle is generated at *(-1888.260, 89.443)*. The actual trajectory of the UUV stays within the computed reachable sets and the UUV passes the obstacle after 24 seconds.

B. Experiment 2: Immediate Course Correction

In Experiment 2, we initialize the UUV with a path directed towards the middle of the obstacle. In order to avoid a collision, the UUV must make an immediate course correction. The UUV is able to detect the object and as a result, it manages to turn and deviates its trajectory to avoid the obstacle in a safe manner as shown in Figure 6.

C. Experiment 3: Unsafe Scenario

In Experiment 3, we initialize the UUV within 17m of the obstacle and heading towards it. This represents a scenario where the UUV does not satisfy the safety condition. We can see that at the eleventh step, 11 seconds, the intersection of the vehicle's reachable set and the unsafe set is not null. However, based on the over-approximation method used in the reachabilty analysis, these results are inconclusive.

[2]Since the state values lack a direct physical meaning, we refer the reader to Appendix A where all the initial state values related to all 4 experiments are presented.

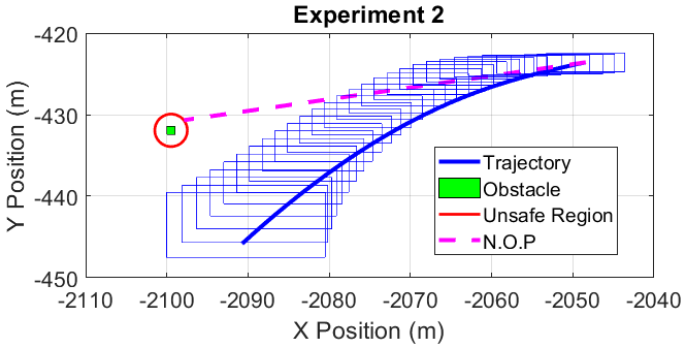

Fig. 6. The UUV is initialized at *(-2051.187, -423.978)* pointing *-170.256* degrees from horizontal. The obstacle is generated at *(-2099.480, -431.911)*. The actual trajectory of the UUV stays within the computed reachable sets and the UUV passes the obstacle after 30 seconds.

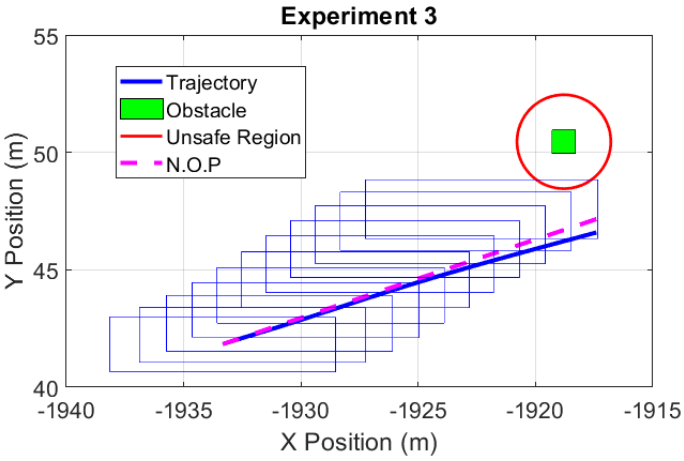

Fig. 7. The UUV is initialized at *(-1933.337, 41.836)* pointing *18.478* degrees from horizontal. The obstacle is generated at *(-1918.790, 50.456)*. The actual trajectory of the UUV stays within the computed reachable sets and the UUV passes the obstacle after 11 seconds.

D. Experiment 4: Choosing a New Path

In Experiment 4, the UUV is initialized far enough away from the obstacle that a small change in the initial heading finds a clear and open path that does not require further adjustments. This highlights the learned behavior of the FNN Controller to minimize heading changes while maximizing the distance between the UUV and the obstacle. Behavior like this could be considered undesirable and is the reason why designing a reward function is so important to the performance of a reinforcement learning controller[3].

However, we clearly observe that the UUV avoids the obstacle and does not get closer than 12 meters to the center of the obstacle.

VI. Conclusion

This paper presents an efficient over-approximate reachability scheme that consists of an exact method for the reachability analysis of neural networks, and an over-approximate method used in the plant reachability analysis in order to consider

[3]For more information about this problem, look into work discussing *Reward Shaping*.

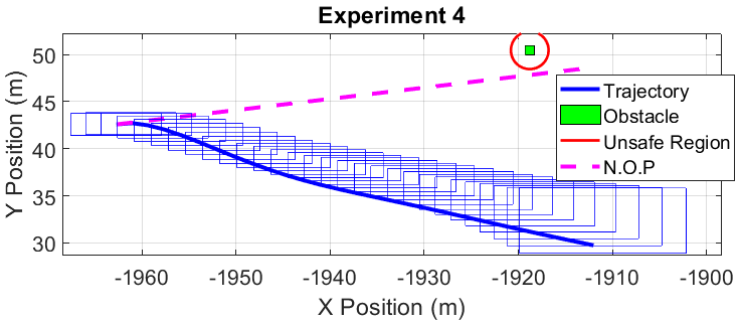

Fig. 8. The UUV is initialized at *(-1948.779, 43.165)* pointing *6.807* degrees from horizontal. The obstacle is generated at *(-1918.790, 50.456)*. The actual trajectory of the UUV stays within the computed reachable sets and the UUV passes the obstacle after 29 seconds.

the safety verification of a cyber-physical system with an RL controller. The safety specification is defined based on the location of obstacles and the underwater vehicle. We have shown that our method is scalable for real-world applications in our analysis of the obstacle avoidance capabilities of an unmanned underwater vehicle.

In future work we hope to extend the proposed methods for nonlinear NNCS with neural networks that contain other types of nonlinear activation functions such as Tanh or Sigmoid. Additionally, while the current paper deals with a discrete-time plant, future endeavors will consider continuous-time dynamical plants. Furthemore, we seek to examine other UUV tasks and safety specifications, such as a safety and performance analysis of the UUV's pipe-following performance in conjunction with obstacle avoidance.

Acknowledgment

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers SHF 1910017 and FMitF 1918450, the Air Force Office of Scientific Research (AFOSR) through contract number FA9550-18-1-0122, and the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, DARPA, or NSF.

References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2503–2511. [Online]. Available: http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv e-prints*, p. arXiv:1312.6199, Dec 2013.

[3] C. Liu, T. Arnon, C. Lazarus, C. W. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019. [Online]. Available: http://arxiv.org/abs/1903.06758

[4] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. A. Rosenfeld, and T. T. Johnson, "Verification for machine learning, autonomy, and neural networks survey," *CoRR*, vol. abs/1810.01989, 2018. [Online]. Available: http://arxiv.org/abs/1810.01989

[5] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: Verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '19. New York, NY, USA: ACM, 2019, pp. 169–178. [Online]. Available: http://doi.acm.org/10.1145/3302504.3311806

[6] S. Dutta, X. Chen, and S. Sankaranarayanan, "Reachability analysis for neural feedback systems using regressive polynomial rule inference," in *Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '19. New York, NY, USA: ACM, 2019, pp. 157–168. [Online]. Available: http://doi.acm.org/10.1145/3302504.3311807

[7] X. Sun, H. Khedr, and Y. Shoukry, "Formal verification of neural network controlled autonomous systems," *CoRR*, vol. abs/1810.13072, 2018. [Online]. Available: http://arxiv.org/abs/1810.13072

[8] W. Xiang, D. M. Lopez, P. Musau, and T. T. Johnson, "Reachable set estimation and verification for neural network models of nonlinear dynamic systems," pp. 123–144, 2019.

[9] W. Xiang, H.-D. Tran, J. Rosenfeld, and T. T. Johnson, "Reachable set estimation and verification for a class of piecewise linear systems with neural network controllers," in *American Control Conference (ACC 2018), Special Session on Formal Methods in Controller Synthesis I*. IEEE, Jun. 2018.

[10] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Case Study: Verifying the Safety of an Autonomous Racing Car with a Neural Network Controller," *arXiv e-prints*, p. arXiv:1910.11309, Oct. 2019.

[11] H.-D. Tran, F. Cei, D. M. Lopez, T. T. Johnson, and X. Koutsoukos, "Safety verification of cyber-physical systems with reinforcement learning control," in *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)*. ACM, October 2019.

[12] D. M. Lopez, P. Musau, H.-D. Tran, and T. T. Johnson, "Verification of closed-loop systems with neural network controllers," in *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 61. EasyChair, 2019, pp. 201–210. [Online]. Available: https://easychair.org/publications/paper/ZmnC

[13] H.-D. Tran, P. Musau, D. M. Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis for deep neural networks," in *23rd International Symposium on Formal Methods (FM'19)*. Springer International Publishing, October 2019.

[14] S. Bak and P. S. Duggirala, "Simulation-equivalent reachability of large linear systems with inputs," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 401–420.

[15] J. Bellingham, "Platforms: Autonomous underwater vehicles," in *Encyclopedia of Ocean Sciences (Second Edition)*, second edition ed., J. H. Steele, Ed. Oxford: Academic Press, 2009, pp. 473 – 484. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012374473900730X

[16] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 2)*, 1st ed. Springer Publishing Company, Incorporated, 2017.

[17] N. P. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004.

[18] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV simulator: A gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, sep 2016. [Online]. Available: https://doi.org/10.1109%2Foceans.2016.7761080

[19] L. Ljung, *System Identification (2nd Ed.): Theory for the User*. USA: Prentice Hall PTR, 1999.

[20] M. S. I. Toolbox, *(R2019b)*. Natick, Massachusetts: The MathWorks Inc., 2019.

APPENDIX

As mentioned in Section 5, we will present the exact initial state (x_0) intervals used for each experiment as well as the exact location of the obstacles (P_{obs}), where

A. Experiment 1

(Figure 5)

$$x_0 = \begin{bmatrix} [-0.044174880802800 \quad , \quad -0.043974880802800] \\ [0.007009187707955 \quad , \quad 0.007209187707955] \\ [0.003998288293794 \quad , \quad 0.004198288293794] \\ [-0.066590984864864 \quad , \quad -0.066390984864864] \\ [0.076075384291567 \quad , \quad 0.076275384291567] \\ [-0.063045980165905 \quad , \quad -0.062845980165905] \\ [-7.781701966436715e-04 \quad , \quad -5.781701966436714e-04] \\ [0.095613905722162 \quad , \quad 0.095813905722162] \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -1888.260 \\ 89.443 \end{bmatrix}$$

B. Experiment 2

(Figure 6)

$$x_0 = \begin{bmatrix} [-0.039780351857121 \quad , \quad -0.039580351857121] \\ [0.062835250855941 \quad , \quad 0.063035250855941] \\ [0.151063041362560 \quad , \quad 0.151263041362560] \\ [0.829311305915308 \quad , \quad 0.829511305915308] \\ [-0.628460030147656 \quad , \quad -0.628260030147656] \\ [0.283328195630901 \quad , \quad 0.283528195630901] \\ [-0.042414499938101 \quad , \quad -0.042214499938101] \\ [-0.304932444834164 \quad , \quad -0.304732444834164] \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -2099.480 \\ -431.911 \end{bmatrix}$$

C. Experiment 3

(Figure 7)

$$x_0 = \begin{bmatrix} [-0.044316470241293 \quad , \quad 0.044116470241293] \\ [0.010611741142870 \quad , \quad 0.010811741142870] \\ [0.028460902915747 \quad , \quad 0.028660902915747] \\ [-1.020282211033473e-04 \quad , \quad 9.797177889665276e-05] \\ [-9.329615809094399e-05 \quad , \quad 1.067038419090560e-04] \\ [-2.667311115387110e-04 \quad , \quad -6.673111153871095e-05] \\ [-7.128280405342853e-05 \quad , \quad 1.287171959465715e-04] \\ [-1.422996186804232e-04 \quad , \quad 5.770038131957683e-05] \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -1918.790 \\ 50.456 \end{bmatrix}$$

D. Experiment 4

(Figure 8)

$$x_0 = \begin{bmatrix} [-0.044991857837650 \quad , \quad -0.044791857837650] \\ [0.010793055476382 \quad , \quad 0.010993055476382] \\ [0.017457598529114 \quad , \quad 0.017657598529114] \\ [0.022151317193675 \quad , \quad 0.022351317193675] \\ [0.101951191993133 \quad , \quad 0.102151191993133] \\ [-0.280253273679023 \quad , \quad -0.280053273679023] \\ [0.172975572623461 \quad , \quad 0.173175572623461] \\ [0.195338830333452 \quad , \quad 0.195538830333452] \end{bmatrix},$$

$$P_{obs} = \begin{bmatrix} -1918.790 \\ 50.456 \end{bmatrix}$$