2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

ReachFlow: An Online Safety Assurance Framework for Waypoint-Following of Self-driving Cars

Qin Lin[1], Xin Chen[2], Aman Khurana[1], and John M. Dolan[1]

***Abstract*— Learning-enabled components have been widely deployed in autonomous systems. However, due to the weak interpretability and the prohibitively high complexity of large-scale machine learning models such as neural networks, reliability has been a crucial concern for safety-critical autonomous systems. This work proposes an online monitor called ReachFlow for fault prevention of waypoint-following tasks for self-driving cars. It mainly consists of two components: (a) an online verification tool which conservatively checks the safety of the system behavior in the near future, and (b) a fallback controller which steers the system back to a desired state when the system is potentially unsafe. We implement ReachFlow in a self-driving racing car governed by a reinforcement learning-based controller. We demonstrate the effectiveness by rigorously verifying a safe waypoint-following control and providing a fallback control for an unsafe situation in which a large deviation from the planned path is predicted.**

I. Introduction

Conventional control approaches have reached a plateau in attempting to deal with highly complex and unpredictable environments. Learning-enabled components (LECs) in autonomous systems such as self-driving cars are equipped with AI components like neural network controllers to increase autonomy by integrating perception, planning, and low-level control. However, the first drawback of LECs is their opaqueness, which leads to a weak *interpretability* that makes it difficult for users or even developers to verify the correctness of the system [1]. The second drawback is the handling of exceptions. Autonomous systems are commonly working in a highly uncertain environment where exceptions can hardly be taken into account in the design phase. For example, a UAV is flying along a planned route in a windy area, and the wind disturbance cannot be easily described due to its rapid changes. This paper provides a solution to overcome the second drawback. We describe an online framework to prevent a given autonomous system from reaching any unsafe state, and apply it to self-driving cars with waypoint-following controllers.

Waypoint-following is a common control task for an autonomous vehicle to track a series of goal points on-road in the real world. Fig. 1 is a high-level illustration of a waypoint-following controller in a closed-loop system. The controller takes the state error (the difference between the vehicle state and the reference state) as the input and generates velocity (if we use a velocity control) and steering angle to control the plant. In this paper, a self-driving RC car is used as a testing platform. Safety verification for the waypoint-following in a closed-loop system can be interpreted sequentially using reachability. We try to inspect whether the system is able to stay in the safe region for a finite amount of time (e.g., the car's future position never deviates more than a specified amount from the planned path computed using waypoints) within a given time horizon $[t, t+T]$, where t is current time, and T is the time horizon in interest.

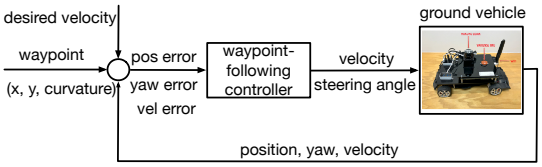

Fig. 1: A waypoint-following controller in a close-loop system. The mapping from the state to the observation has been omitted for simplicity.

Depending on the length of T, we are interested in *one-step verification* and *multiple-step verification*. Reachability for *one step* can be easily done by assuming the control action stays constant during a sampling step. *Multiple-step verification* is challenging when the controller in Fig. 1 is a black box, i.e., its formal model is unknown.

In this paper, we deal with a closed-loop control system such that the controller is given by a black box and the plant is defined by a nonlinear Ordinary Differential Equation (ODE). The model is shown in Fig. 1. The ground vehicle is an RC car, the testbed on which we implement and test ReachFlow. The Lidar and the IMU are used for localization. The WiFi component is responsible for the data transmission between the RC car and the computer.

The RC car might not move along the planned path even with a correctly designed waypoint-following controller due to the exceptions or disturbances from the environment. For example, the deviation from the path caused by uneven friction from the ground may accumulate during the car's motion. Such a case can hardly be taken into account in the design of the controller, since we are not able to formalize all possible environments for the system. Instead, we propose to use an online framework which performs an auxiliary control for the system under exceptions. That is, the framework monitors the current state of the system and predicts the system behavior in the near future. If a potential unsafe state is detected, then the fallback controller will take over

[1]Qin Lin, Aman Khurana, and John Dolan are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA qinlin,amankh,jdolan@andrew.cmu.edu
[2]Xin Chen is with the Department of Computer Science, University of Dayton, Dayton, OH 45469, USA. xchen4@udayton.edu

the system control and steer the system back to a desired state. In our application, when a potential large deviation is detected, the fallback controller will steer the car back to a reference state which is close to the planned path, and then give the control back to the original controller. Since the safety verifier and the fallback controller are working independently from the system, we do not need to know the exact definition of the original controller.

The rest of this paper is organized as follows. Section II provides a discussion of the related work. Section III presents the details of the online safety assurance framework. The experimental evaluation is given in Section IV, and the paper is concluded in Section V.

II. Related work

The online control of autonomous systems has been widely studied and has given birth to many algorithms for motion planning, unsafety avoidance, and exception handling. However, very few existing works provide formal guarantees on the results obtained. The main reason is the complexity of autonomous systems. Formal verification requires an explicit formal definition of the system in consideration, and the reachability problem, i.e., verifying whether a given state is reachable, is decidable only on a very limited category of Cyber-Physical Systems (CPS) [2]. Although a great amount of effort has been devoted to developing the over- and under-approximation techniques for computing the reachable sets of CPS [3], [4], [5], [6], [7], [8], very few of the methods can be applied to producing reachable set predictions online for a system with nonlinear dynamics due to the low efficiency.

Nevertheless, some recent publications do provide solutions to the online verification problem on subclasses of autonomous systems. In [9], a reachability analysis algorithm for linear systems is applied to verify the safety of automated road vehicles in real time. In [10], the authors propose an approach to online-check whether it is possible to prevent a linear system from reaching an unsafe state. This work is later extended to handle data-driven models with probability [11]. In [12], a decentralized real-time safety verification approach is presented for distributed CPS. Besides, some theorem proving-based methods are also applied to solve the online verification problem for CPS [13], [14].

In this paper, our online framework uses a formal verifier to conservatively estimate the safety of the system behavior in the near future, so that it essentially solves a bounded model checking problem on a nonlinear system online but only for a small number of steps. Since we need to take the noise from the environment into account and implement the framework in the testbed, the tool FLOW* [5], [15], which handles uncertain nonlinear ODEs and is implemented in C++, is the best candidate for the verifier.

III. Main Framework of ReachFlow

Our online verification & control framework consists of the components in the dashed region shown in Fig. 2. For any waypoint-following control system, the framework works cooperatively with the existing components of the system such that in each control step, the verification component checks the safety of the system execution in a few future steps, if an unsafe state is potentially reachable, the fallback controller will take over the control and steer the system to a "safer" state. For example, if the car can be potentially pushed away from the planned path at a future time t by an environmental disturbance, the fallback controller will be used to keep the car being as close as possible to the path.

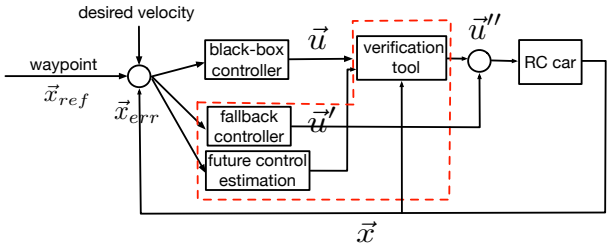

Fig. 2: Closed-loop control system studied and our framework in this paper.

The details of the framework is shown by Algorithm 1. In each control step, the future control estimation component generates, based on a Gaussian process, a sequence of control inputs $\vec{u}_1, \ldots, \vec{u}_k$ which are the prediction of the control inputs used in the next k steps. Then the verification tool FLOW* computes the reachable set overapproximations for the next k steps from the current system state based on the control input prediction. If no unsafe state is contained the overapproximations, then the control input produced by the original controller will be sent to the plant. Otherwise, the fallback controller takes over the control of the system, and computes a more conservative input for the system. Here, our fallback controller is a Linear Quadratic Regulator (LQR) which steers the car to a reference on the path.

Algorithm 1 Main Framework

Input: Control input $\vec{u}$ from the black-box controller
Output: New control input $\vec{u}'$
for *each control step* **do**
  Obtaining the control input $\vec{u}$ from the original controller;
  Obtaining the future control estimate;
  Computing the reachable set for the next k steps;
  Verifying the safety of the reachable set.
  if *It is safe* **then**
    return $\vec{u}$;
  else
    Computing $\vec{u}'$ using the fallback controller;
    return $\vec{u}'$;
  end
end

A. Taylor Model Flowpipe Construction

We briefly revisit the *Taylor Model (TM) flowpipe construction* technique [16] for computing reachable set over-approximations for continuous systems which are defined by nonlinear ODEs in the form of $\vec{x} = f(\vec{x}, \vec{u})$ where

$\vec{x}, \vec{u}$ are the collective representations of the state variables $(x_1, \ldots, x_n)$ and control input $(u_1, \ldots, u_m)$ respectively.

Given a continuous system defined by $\vec{x} = f(\vec{x}, \vec{u})$ along with an initial condition $\vec{x}(0) \in X_0 \subset \mathbb{R}^n$ and a closed and bounded set $\mathcal{U}$ for $\vec{u}$, the *reachable set* of the system in a given time interval $[0, T]$ is defined by the set $R_{[0,T]} = \{\varphi_f(\vec{x}_0, \vec{u}_0, t) \,|\, \vec{x}_0 \in X_0, \vec{u}_0 \in \mathcal{U}, t \in [0, T]\}$ where $\varphi_f(\vec{x}_0, \vec{u}_0, t)$ is the solution of the ODE with the initial state $\vec{x}_0$ and under the control input $\vec{u}_0$. Since it is not possible to exactly compute the set $R_{[0,T]}$ in general, we seek to compute an overapproximation of it such that a large deviation in the actual system behavior can be reflected by a nonempty intersection of the overapproximation and a corresponding unsafe set.

The method of TM flowpipe construction is to compute a finite set of TM flowpipes $(p_1, I_1), \ldots, (p_N, I_N)$ based on a time stepsize $\delta > 0$ such that for each $1 \leq i \leq N$, (1) p_i is a polynomial over the variables $\vec{x}_0, \vec{u}_0$ and t, and I_i is an interval; (2) the range of the TM (p_i, I_i), i.e., $\{p_i(\vec{x}_0, \vec{u}_0, t) + \vec{z} \,|\, \vec{x}_0 \in X_0, \vec{u}_0 \in \mathcal{U}, t \in [0, \delta]\}$ is an overapproximation of the reachable set $R_{[(i-1)\delta, i\delta]}$. Hence, the union of the flowpipes forms an overapproximation of the reachable set $R_{[0,N\delta]}$.

The algorithm to compute TM flowpipes is called TM integration, which consecutively computes TMs for reachable set segments from the initial set. In our framework, we use the tool FLOW* to compute TM flowpipes for predicting the system behavior in the near future. The safety checking on the TMs can also be handled by the tool.

Although we have no knowledge of the future control inputs in computing the reachable set predictions, we may compute an estimate for their range.

B. Uncertainty Estimation for Control Inputs

The explicit value of future control is unavailable without sequentially querying the controller. However, intuitively over a short predictive horizon, the change of control is normally bounded. In the following, we will introduce three approaches to estimating this bound.

Physical constraint-based approach. The range of the control inputs is constrained by the capacity of actuators, and therefore, we may use this physical range to predict the reachable sets. A weakness of such a method is that the range is usually too large and it can cause a tremendous overestimation in the TM flowpipes.

Sampling distribution-based approach. The range is computed based on a set of historical changes in the control inputs. We collect the changes of the past control inputs and apply a probability distribution, e.g., Gaussian distribution, to fitting the data, and then obtain a range for the data based on a given confidence, which can be considered as a belief of the estimate.

Gaussian Process-based approach. An estimate for the range of the future control inputs may also be obtained using machine learning-based approaches. Here, we use Gaussian Process (GP). If we do not take large noises or disturbances into account, the control inputs generated by the controller are usually similar if they are computed based on similar system states, since most of the feedback controllers perform a continuous mapping. The historical data with the state as input and the maximum control change over T as output is trained using a Gaussian Process approach. In the online process, the new state is fed into the model to obtain the range of future control actions.

In this work, an extended scenarios-related state consisting of the relative distance to the waypoint and the curvature is defined as the input. The maximum change of the control action over T is defined as the output. The curvature information is appended to the input because intuitively stationary waypoint-following for a straight line has different (actually less) control change compared with scenarios like making a turn. The difficulty is finding a suitable explicit machine learning model to describe such a mapping. A Gaussian Process does not assume a specific model. Instead, GP uses the correlation between data. The covariance between two data vectors is defined as $k(\vec{s}', \vec{s}'') = \exp\left[\sum_{i=1}^{d} \frac{-(\vec{s}_i' - \vec{s}_i'')^2}{2l^2}\right]$, where d is the dimension of the input. The n training input data are $[\vec{s}^1, \vec{s}^2, \cdots, \vec{s}^n]^T$, and the n training output data are $[y^1, y^2, \cdots, y^n]^T$. The new input datum received online is $\vec{s}_*$. The estimated output datum and its uncertainty is:

$$y_* = K_* K^{-1} \mathbf{y} \tag{1}$$

$$\mathrm{var}(y_*) = K_{**} - K_* K^{-1} K_*^{\mathrm{T}} \tag{2}$$

where $K_* = \begin{bmatrix} k(\vec{s}_*, \vec{s}^1) & k(\vec{s}_*, \vec{s}^2) & \cdots & k(\vec{s}_*, \vec{s}^n) \end{bmatrix}$, $K_{**} = k(\vec{s}_*, \vec{s}_*)$, and K is a $n * n$ covariance matrix for the training input data. Note that the output dimension of GP is 1, thus we need to construct two GP models for v and δ_f. K is only related to training data, so the computation for such a big matrix's inverse can be done offline. The parameter estimation for l can also be done offline by optimizing a negative marginal log-likelihood function. Readers are referred to [17] for more details about GP inference. The control action is subject to $\mathcal{U}$ represented by an interval $[a, b]$, where a and b are from $y_* \pm m * \mathrm{var}(y_*)$, and m is a user-defined parameter related to confidence. Such a scenario-based uncertainty estimation using Gaussian Process is called SUE-GP in this work.

IV. EXPERIMENTAL RESULTS

A. Vehicle dynamical models

We briefly introduce two vehicle models used in this work: a simplified but more general *kinematic bicycle model* and a more complex *dynamic bicycle model* considering tire dynamics.

1) Kinematic bicycle model: The nonlinear equations describing the dynamics are as follows:

$$\begin{aligned} \dot{x}_{pos} &= v\cos(\psi + \beta) \\ \dot{y}_{pos} &= v\sin(\psi + \beta) \\ \dot{\psi} &= v\sin(\beta)/l_r \\ \beta &= \arctan\left(l_r \frac{\tan(\delta_f)}{l_f + l_r}\right) \end{aligned} \tag{3}$$

where x_{pos} is the position along the x-axis and y_{pos} is the position along the y-axis. They are coordinates in a "global" initial frame. $\dot{x}_{pos}$ and $\dot{y}_{pos}$ are the velocity and ψ and $\dot{\psi}$ are the yaw and the yaw rate. β is the angle of the current velocity of the center of mass with respect to the longitudinal axis of the car, and l_f and l_r represent the distance from the center of the mass of the vehicle to the front and rear axles, respectively.

The model has three states $[x_{pos}, y_{pos}, \psi]^T$; β is an auxiliary variable. The control input is two-dimensional: $[v, \delta_f]^T$, where v is the desired speed, and δ_f is the steering angle.

2) Dynamic bicycle model - linear tire model: The kinematic bicycle model is widely used owing to its simplicity - only two distance parameters need to be identified. However, this model is limiting because it assumes that the velocity vectors at the wheels are in the direction of the orientation of the wheels, implying that all slip angles are zero. Intuitively, this assumption holds when the lateral forces are small at relatively low speed.

In the following, we introduce a dynamic bicycle model taking lateral force into account.

$$\dot{v}_x = \dot{\psi}\dot{y} + \frac{1}{m}(F_{xr} - F_{yf}\sin\delta_f)$$
$$\dot{v}_y = -\dot{\psi}v_x + \frac{1}{m}(F_{xr}\cos\delta_f + F_{yr})$$
$$\ddot{\psi} = \frac{1}{I_z}(l_f F_{yf} - l_r F_{yr})$$

The dynamic bicycle model has the same dynamic form about $\dot{x}_{pos}, \dot{y}_{pos}$, and $\dot{\psi}$ as the kinematic bicycle model in Equations 3. Note that $v_x \neq \dot{x}_{pos}$ and $v_y \neq \dot{y}_{pos}$. v_x (resp. v_y) is the velocity of the vehicle with respect to the longitudinal (resp. lateral) axis of its body frame, i.e., a "local" frame. m is the mass of the vehicle, and F_{xr} and F_{yr} are the longitudinal and lateral force of the rear tire respectively. F_{yf} is the lateral force of the front tire. I_z is the moment of inertia about the z-axis.

In a linear tire model, the longitudinal forces and lateral forces on the tire are defined as

$$F_{xi} = C_x\kappa$$
$$F_{yi} = -C_{\alpha_i}\alpha_i$$

where $i \in \{f, r\}$, α_i is the tire slip angle, C_x is the tire stiffness and C_{α_i} is the tire cornering stiffness. We use the same front and rear tires for the RC car, i.e., $C_\alpha := C_{\alpha r} := C_{\alpha f}$. κ is the slip value defined as

$$\kappa = \frac{R\omega - v_x}{v_x}$$

where R is the radius of the wheel and ω the angular velocity of the wheel. $\kappa = 1$ when the vehicle is slipping; $\kappa = \infty$ if the vehicle is skidding; and $\kappa = 0$ if the vehicle has no slip or skid. α_f and α_r are defined as:

$$\alpha_f = \arctan\left(\frac{v_y + l_f\dot{\psi}}{v_x}\right) - \delta_f$$
$$\alpha_r = \arctan\left(\frac{v_y - l_f\dot{\psi}}{v_x}\right)$$

B. System implementation

General reinforcement learning uses a trial-and-error learning strategy for optimal control policy search. In this work, we use Constrained Policy Optimization (CPO) [18], [19], a local policy search method, to find the optimal policy that satisfies all constraints. An iterative simulation environment is set up for policy training. The learned policy is stored and transferred to the real RC car. Note that in theory, CPO can cooperate with the user-defined safety constraints with probabilistic guarantees. However, it is observed that due to the gap between simulation and the real world, e.g., sensor and actuator delays, control frequency, and other variables, we are not able to ensure the perfection of the policy from the simulation. The demanding safety verification in this work will verify the control policy before the execution. The code for custom simulation environments, training scripts used to train policies, and the running trained policies[1] on our robot are available online. We implemented three scenarios: *straight-line*, *circle*, and *rounded square* (see Fig. 4).

C. Reachability evaluation

The idea to evaluate the effectiveness of reachability is straightforward: we inspect whether all positions of the car fall inside the historically predicted reachable set. The number of successful instances over the total number of executions is defined as the hit rate. We also compare the average uncertainty size of the control change. A higher hit rate and small average uncertainty size indicate that we can obtain a stronger guarantee of the predicted states with a tighter uncertainty. The results are shown in Tab. I. The bound for physical constraints is a user-defined parameter. The confidence level for the distribution-based approach is set to $\mu + 1 * \sigma$. We also set a $1 * \sigma$ confidence level for SUE-GP. We can observe that SUE-GP has slightly less v uncertainty and significantly less δ_f uncertainty. For SUE-GP, only 5 positions over 1000 executions are found not to fall into the reachable set. Fig. 3 shows two representative counterexamples; 3 other examples are near these two points with a 0.1-0.2 second difference. The offsets of these two points from the predicted reachable set are only 0.09m and 0.03m, respectively. The number immediately increases to 100% by increasing the confidence level from 1 σ to 2 σ or increasing the localization uncertainty from 0.1 m to 0.15m.

The performance of the linear model is the worst. The problem is that the system's model, i.e., the Jacobian matrix from linearization, is derived from the current state. However, in this experiment multiple-step reachability is computed. The state has evolved but the model does not get updated accordingly. This problem becomes more serious when the car is turning. It has been validated that when we try to do a single-step verification for a 0.1-second ahead prediction, the hit rate for the linear model significantly increases and reaches 99.9% using SUE-GP. The dynamic bicycle model achieves a higher hit rate than the bicycle model using identical SUE-GP parameters.

[1] https://github.com/r-pad/aa_planner

TABLE I: Reachability performance comparison among physical constraints-approach, sampling distribution approach, and SUE-GP with different dynamic models for multiple-step prediction (0.5 s)

	Phy.	Samp.	SUE-GP Bicycle	SUE-GP Dynamic	SUE-GP Linear
Hit rate (%)	100	100	99.95	99.99	58.9
v uncertainty	0.6	0.04	0.038	0.038	0.038
δ_f uncertainty	1.1	1.272	0.673	0.673	0.673

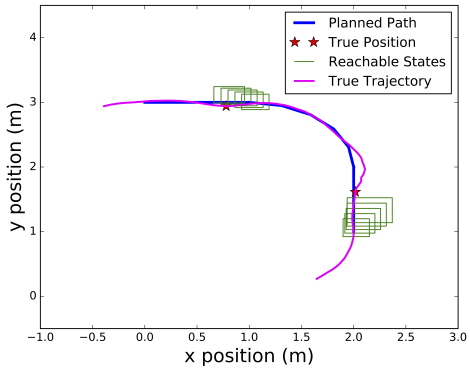

Fig. 3: Counter examples: the righthand point has position offset 0.09m, left hand position offset 0.03m

D. Safety Verification

1) Safety specification: ReachFlow allows *linear safety specifications* and *polynomial safety specifications*. The linear specification is defined by linear inequalities in a negative null form, i.e., $H\vec{x} - \vec{b} <= 0$, which is the joint of linear half-space. The polynomial specification is written again in a null form as $p(\vec{x}) <= 0$. In the specific waypoints task, $\vec{x}$ can be defined as the relative distance between the predicted position and the reference position.

2) Fallback control experiment: To validate the verification of ReachFlow, we artificially replace the original control action (turning left) with a control action tending to keep driving straight at the position $(1, 0)$ (see the rounded square in Fig. 4). The pre-planned perfect path is represented as $(x-1)^2 - (y-1)^2 = 1$. The unsafe condition can be defined as an inequality $(r+\epsilon)^2 - (x-1)^2 - (y-1)^2 <= 0$ recognizable for ReachFlow by inflating or deflating. In the examples of Fig. 5, Fig. 6, and Fig. 7, ϵ is set to 0.25 to

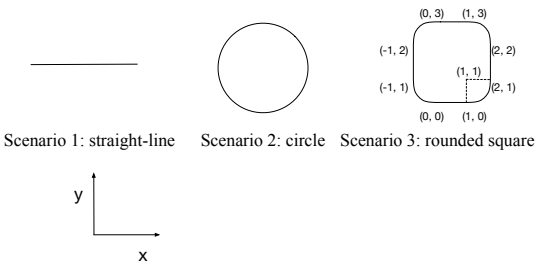

Fig. 4: Three testing scenarios: straight-line, circle, and rounded square. CPO is able to plan arbitrary path following. These are just test cases for validating ReachFlow.

verify if the car will drive outside the circle. Similarly, the verification of driving inside the circle can be done by setting ϵ as a negative number. ϵ can be interpreted as a tolerance of the safe path following. The specification of an unsafe state is left as an interface to users in ReachFlow. The users can define any realistic specifications depending on the specific problems. For example, the inflating curve in Fig. 5, Fig. 6, and Fig. 7 can be a curb of a road. Then the problem will be verifying the possibility of hitting the curb in the future T horizon. Though collision avoidance is not studied in this work, an obstacle can be defined as an unsafe state by bounding it as a convex hull using linear inequalities. Then a safety verification of a collision avoidance can be addressed.

Fig. 5 shows the verification result around the position $(1, 0)$. The rectangles are reachable states (x_{pos}, y_{pos}, ψ) projecting to the (x, y) axis, where the maximum and minimum values are bounded. The black dashed line is the pre-planned path, and the red dashed line is the boundary of the safety specification. The safety is guaranteed because no rectangle crossing the boundary is detected. Fig. 6 shows the verification result of the "driving straight" control action, which is injected as a disturbance. The green rectangles represent safe states with high certainty (no intersection with boundary), blue rectangles are for "suspicious" unsafe states (having intersection with boundary), and the red rectangles are unsafe states with high certainty (already outside the boundary). The users can play with the tolerance setting and defining "suspicious" unsafe as safe or unsafe. Fig. 7 shows that the control action of the LQR fallback controller is verified to be safe. Note that in the real system, if the control action from the original controller is verified to be unsafe, we immediately switch to the fallback controller. Due to the real-time demand, the fallback control is not verified. Our assumption in this work is that the fallback controller is safe. It would be our future work to further study an optimal controller without extra verification.

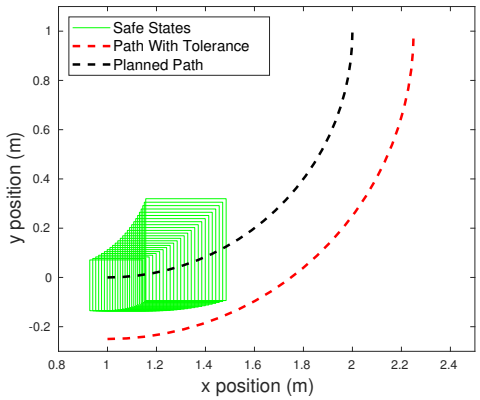

Fig. 5: Verified to be safe action (original controller)

Tab. II shows a comparison of the runtime of the whole framework using different models with prediction horizons 0.1 second and 0.5 second from 1000 runs. We can observe that model complexity does not influence the computation

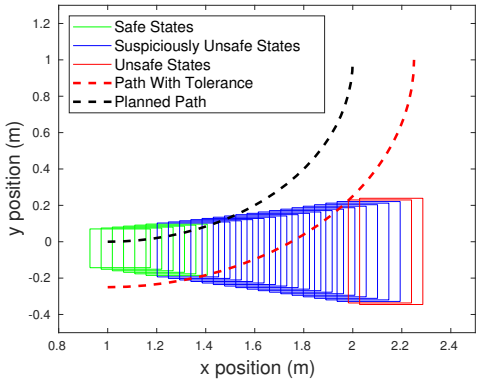

Fig. 6: Verified to be unsafe disturbance

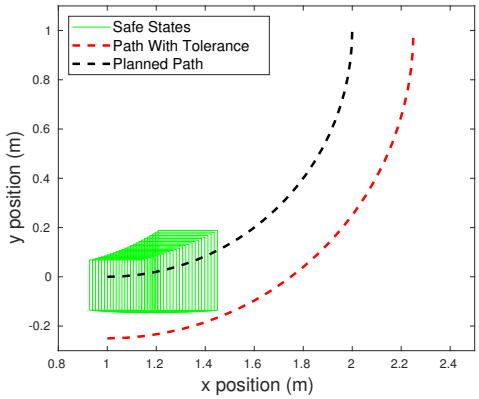

Fig. 7: Verified to be safe action (LQR controller)

cost significantly, although the dynamic model has three more states than the linear and the bicycle model. All models have satisfactory real-time performance.

TABLE II: Runtime comparison

Time	Linear (ms)	Bicycle (ms)	Dynamic (ms)
Average ($T = 0.1s$)	27.56	26.34	26.00
Std.	7.67	7.28	5.68
Average ($T = 0.5s$)	46.12	52.46	49.46
Std.	12.10	11.76	7.27

V. Conclusions

We propose ReachFlow for safety verification of waypoint-following of a self-driving car governed by a reinforcement learning controller. ReachFlow is built on an efficient prototype tool based on the flow* library. The framework also uses a Gaussian Process approach called SUE-GP to estimate future control bounds for multi-step verification. In the experiment, we demonstrate that using SUE-GP and a dynamical vehicle model, 99.99% assurance can be obtained to enclosure all future trajectories. By increasing the variance or increasing the initial position uncertainty, we can achieve 100% assurance. Unsafe verification and fallback control are also demonstrated. Future work will include studying a sophisticated fallback controller.

VI. Acknowledgment

This material is based upon work supported by the US Air Force and DARPA under contract No. FA8750-18-C-0092, and the US Air Force Research Laboratory (AFRL) under contract No. FA8650-16-C-2642. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Air Force and DARPA.

References

[1] Q. Lin, "Intelligent control systems: Learning, interpreting, verification," Ph.D. dissertation, Delft University of Technology, 2019.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.

[3] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 1997, pp. 460–463.

[4] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.

[5] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*. IEEE Computer Society, 2012, pp. 183–192.

[6] X. Chen, S. Sankaranarayanan, and E. Ábrahám, "Under-approximate flowpipes for non-linear continuous systems," in *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design (FMCAD'14)*. IEEE, 2014, pp. 59–66.

[7] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, "C2E2: A verification tool for stateflow models," in *Proc. of TACAS'15*, ser. LNCS, vol. 9035. Springer, 2015, pp. 68–82.

[8] M. Althoff, D. Grebenyuk, and N. Kochdumper, "Implementation of taylor models in cora 2018," in *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 2018.

[9] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[10] X. Chen and S. Sankaranarayanan, "Model-predictive real-time monitoring of linear systems," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE Press, 2017, pp. 297–306.

[11] H. Yoon, Y. Chou, X. Chen, E. Frew, and S. Sankaranarayanan, "Predictive runtime monitoring for linear stochastic systems and applications to geofence enforcement for UAVs," in *Proceedings of Runtime Verification 2019*, ser. Lecture Notes in Computer Science, B. Finkbeiner and L. Mariani, Eds., vol. 11757. Springer, 2019, pp. 349–367.

[12] H. Tran, L. V. Nguyen, P. Musau, W. Xiang, and T. T. Johnson, "Decentralized real-time safety verification for distributed cyber-physical systems," in *Proc. of FORTE'19*, ser. LNCS, vol. 11535. Springer, 2019, pp. 261–277.

[13] N. Fulton, S. Mitsch, J. Quesel, M. Völp, and A. Platzer, "Keymaera X: an axiomatic tactical theorem prover for hybrid systems," in *Proc. of CADE'15*, ser. LNCS, vol. 9195. Springer, 2015, pp. 527–538.

[14] B. Bohrer, Y. K. Tan, S. Mitsch, A. Sogokon, and A. Platzer, "A formal safety net for waypoint-following in ground robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2910–2917, July 2019.

[15] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," Ph.D. dissertation, RWTH Aachen University, 2015.

[16] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *Proc. of RTSS'12*. IEEE Computer Society, 2012, pp. 183–192.

[17] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[18] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 22–31.

[19] E. Ahn, "Towards safe reinforcement learning in the real world," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, July 2019.