Towards Robust Autonomous Systems through Uncertainty Quantification

Anish Muthali

Technical Report No. UCB/EECS-2024-203
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-203.html

December 1, 2024

Towards Robust Autonomous Systems through Uncertainty Quantification

Anish Muthali
anishmuthali@berkeley.edu

Thesis

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

[illegible]

Professor Claire Tomlin
Research Advisor

May 16, 2024

(Date)

* * * * * * *

[illegible]

Professor S. Shankar Sastry
Second Reader

May 15, 2024

(Date)

Abstract

We present several methods for improving the robustness of and providing assurance for control stacks that involve learning-enabled components, particularly deep learning algorithms. We leverage uncertainty quantification as a primary tool towards this goal, and we present several algorithms for translating arbitrary measures of uncertainty to practical and usable assurances. In our analysis, we separately address methods where our learning-enabled components can make predictions and receive ground truth information in real time, and methods where ground truth data would not be available at runtime. In the first case, we provide exact assurances on the behavior of our control stack, and in the second case, we empirically demonstrate robustness according to a desired operating specification. We discuss applications of these methods in practical systems, including autonomous vehicles, quadrupedal robots, and autonomous aircraft.

Acknowledgements

Firstly, I would like to thank my advisor, Professor Claire Tomlin. She has been extremely supportive over the last four years of my time in this lab, and she has been very receptive and supportive of all of my interests and ideas. She has encouraged me to be a curious and rigorous researcher, and she has supported me the entire way, from the offices of Sutardja Dai Hall to the middle of nowhere in Glasgow, Montana.

I would also like to thank all of my colleagues over the past four years of research: Forrest Laine, Haotian (David) Shen, Sampada Deglurkar, and Michael H. Lim. Their help, both in and out of the lab, has been crucial to my success as a researcher.

Additionally, I would like to thank Professor S. Shankar Sastry for his willingness to be a second reader for my thesis.

Finally, I would also like to thank my parents, and especially my brother, for their unwavering support.

Contents

Chapter 1

Introduction and Motivation

Autonomous systems' operation can be analyzed in a closed-loop manner through four primary stages: perception, prediction, planning, and control, as shown in Figure 1.1. In designing these systems, a common concern is certifying the safety of each of these modules in the hopes of obtaining a system-level safety certification. With modern autonomous systems using deep neural networks to perform the tasks of some or all of these modules, certifying systems has become a challenge.

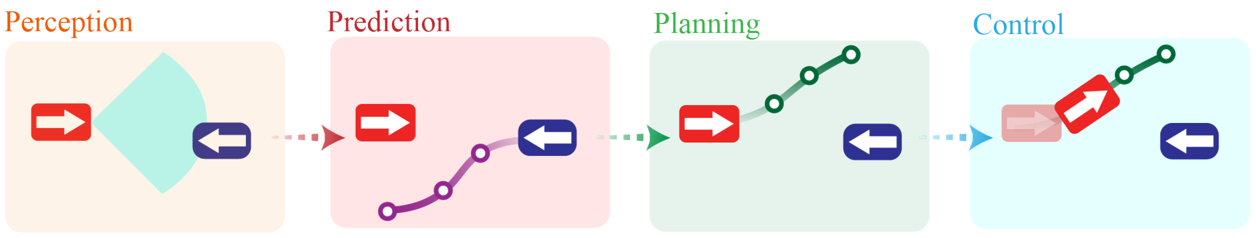


Figure 1.1: A depiction of a typical control stack. The autonomous agent is denoted by the red box with an arrow, and another (possibly human-operated) agent is denoted by the blue box with an arrow. We wish to design a safe and robust control stack that operates within some preset parameters.

At a high level, if we denote an arbitrary data-driven model as $f_\theta(\cdot)$, parameterized by θ, we can analyze the quality of the model in a few ways, including the distribution of the model's residuals (i.e., its prediction error), or the distribution of θ. This gives rise to a fundamental study in modern deep learning: uncertainty quantification (UQ). Deep learning model uncertainty is usually characterized in two groups: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty is variance in the data, i.e., the variance in residuals present even when a model is perfectly parametrized. Epistemic uncertainty, the kind of uncertainty we focus on in these works, focuses on the uncertainty in estimating the model's parameters, θ.

In lower-dimensional statistical estimation problems, we can often measure parametric uncertainty directly, such as, by making assumptions on the data distribution and/or prior over the parameter space. We can use this parametric uncertainty downstream to construct confidence intervals on our predictions. However, due to the high-dimensional nature of modern neural networks, designing theoretically-sound uncertainty measures remains a challenge. As such, most techniques leverage heuristics to approximate epistemic uncertainty. The lack of upstream guarantees on uncertainty measure correctness motivates the need for additional tools for certification. For the purpose of this work, we will primarily focus on systems that incorporate a single learned component, since the propagation of uncertainty heuristics through multiple high-dimensional, data-driven modules will significantly increase the complexity of any assurance algorithm. However, in the final section of this work, we will briefly discuss some ideas that can help us achieve safety specifications in the presence of possibly several learning-enabled components.

In this work, we primarily focus on establishing certifications on the perception and prediction components of autonomous systems, and in some cases, we discuss implications in certifying the safety of downstream planning and control. In Chapter 2 and Chapter 3, we discuss UQ in online settings, where we issue predictions and, in real time, receive ground truths corresponding to these predictions. In Chapter 4, we discuss UQ in offline settings, where the goal is to construct a more robust prediction module or attain a desired safety specification.

Below, we outline some of our contributions in incorporating uncertainty quantification for learning-in-the-loop control systems:

1. In Chapter 2, we propose a novel method to construct certifiably safe confidence sets on predictions output by a trajectory forecasting neural network. We demonstrate our application in predicting the behavior of human drivers on the road, and, using this, we present a planning framework that allows a self-driving car to safely navigate amongst human drivers. Our method achieves a deterministic bound on error rate. Namely, we construct time-indexed confidence sets $\mathcal{Y}_t$ such that our trajectory forecaster's output $\hat{y}_t$ satisfies $\frac{1}{T}\sum_{t=1}^{T}\mathbf{1}\{\hat{y}_t \notin \mathcal{Y}_t\} \leq 1-\alpha+\mathcal{O}(1/T)$.

2. In Chapter 3, we demonstrate initial steps in obtaining anytime-valid guarantees on a dynamics estimation problem. Namely, we wish to extend our results from Chapter 2 to construct confidence sets $\mathcal{X}_t$ such that $\mathbb{P}(x_t \in \mathcal{X}_t \ \forall t) \geq 1-\alpha$.

3. In Chapter 4, we discuss a robust tracking framework that incorporates neural network uncertainty which allows us to control false positive and false negative rates based on system specifications. Our method leverages a sequential hypothesis testing framework.

Chapter 2

Safe, Learning-Enabled Planning for Autonomous Vehicles

We investigate the problem of an autonomous agent interacting with multiple humans[1]. To safely maneuver the autonomous agent around people, we need reliable certification on any prediction of the humans' behavior. For example, in self-driving tasks, the autonomous car is responsible for assuring the safety of itself and the other vehicles it encounters. State-of-the-art systems try to achieve this through behavior prediction and motion forecasting models, oftentimes black-box neural networks that do not provide interpretation of their inner workings [40, 52, 22]. However, these models lack rigorous safety assurances, especially in the presence of data distribution shifts. While lower dimensional models can provide safety assurances, these methods rely on parametric assumptions on, for example, a human's rationality [19]. These assumptions can be inadequate for complex, multi-modal, and noisy decision-making scenarios. In this work, we provide safety assurances for state-of-the-art black-box trajectory forecasting methods by quantifying these models' uncertainty.

However, uncertainty quantification methods alone are insufficient for providing safety assurances. This is because they do not provide guarantees on model behavior and may be unreliable or uninterpretable [24, 59, 11]. Additionally, using prediction model uncertainty for assuredly safe decision-making in downstream planning and control is a challenging problem.

In this chapter, we first utilize conformal prediction to *calibrate* measures of uncertainty [54]. Conformal prediction is a statistical tool that uses a heuristic notion of risk to non-parametrically estimate quantiles of risk given a sequence of past observations [1]. Existing methods that leverage conformal prediction in the context of trajectory forecasting do not explicitly use interpretable metrics of prediction uncertainty [32, 15, 50, 12, 30]. We propose a method that provides rigorous confidence intervals on model error, a form of probabilistic assurance, given any interpretable heuristics on a trajectory forecasting model's prediction uncertainty. Our approach also allows us to examine the efficacy of various uncertainty quantification heuristics when attempting to predict model error. In addition, we extend our analysis to multi-agent environments to closely reflect real-world assurance cases. We analyze the problem of a single autonomous agent, commonly described as an "ego agent", interacting with other, uncontrolled agents.

By producing estimates of model error, we are able to couple statistical assurances with dynamical assurances to allow for safe downstream navigation. In particular, we turn to Hamilton-Jacobi (HJ) reachability analysis [4], which provides guarantees on dynamical systems by means of reachable sets and associated controllers. In HJ reachability, a Hamilton-Jacobi partial differential equation is solved to obtain an optimal value function and controller. The sub-zero level sets of this value function are the reachable sets (possible states of an agent at a given time) and tubes (possible states of an agent *up to* and including a given time).

Our method can be outlined as follows: given a trajectory forecasting model with an associated uncertainty heuristic, we design a quantile regression model that correlates uncertainty with prediction error, creating an approximate confidence interval on the model's prediction. We then apply conformal prediction to calibrate the confidence intervals and provide guarantees on miscoverage rate. We map the calibrated

[1]This chapter is derived from [35] and was developed in collaboration with Haotian Shen, Sampada Deglurkar, and Michael H. Lim from UC Berkeley and Rebecca Roelofs and Aleksandra Faust from Waymo Research and Google Brain respectively.

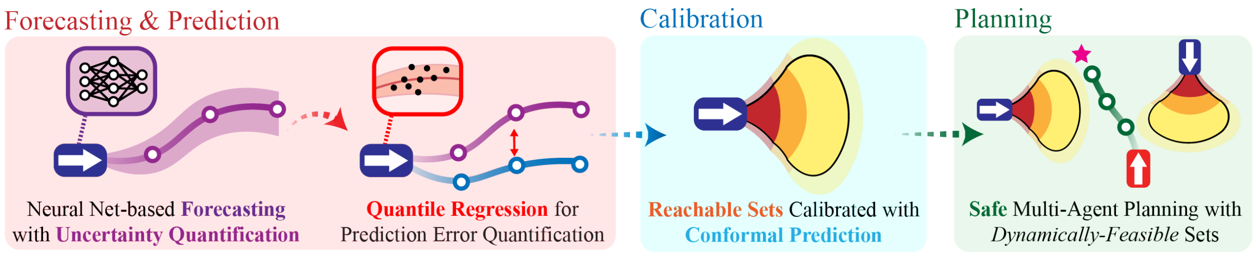

Forecasting & Prediction
Calibration
Planning
Neural Net-based Forecasting with Uncertainty Quantification
Quantile Regression for Prediction Error Quantification
Reachable Sets Calibrated with Conformal Prediction
Safe Multi-Agent Planning with Dynamically-Feasible Sets

Figure 2.1: Our method generates dynamically-feasible, probabilistic confidence sets that are derived from a conformal-calibrated quantile regression model.

intervals in control action space to sets in state space through reachability analysis, and we demonstrate the utility of these confidence sets in planning tasks. The contributions of this work include:

1. A novel way to interpret trajectory forecasting models' prediction uncertainty and obtain approximate confidence intervals (Section 2.2.2);
2. A technique to calibrate the aforementioned intervals using conformal prediction (Section 2.2.3);
3. Dynamically-feasible, probabilistic reachable sets using calibrated intervals (Section 2.3.1);
4. A planning framework that leverages assurances developed in the previous steps (Section 2.3.3).

This chapter is organized as follows: Section 2.1 discusses related works in conformal prediction and assurances in trajectory forecasting models, Section 2.2 and Section 2.3 describe the contributions outlined above, and Section 2.4 showcases the safety and performance of our methods compared to baseline methods.

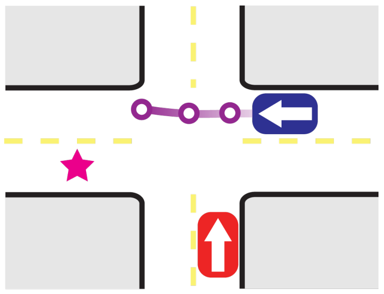
2.1 Related Works

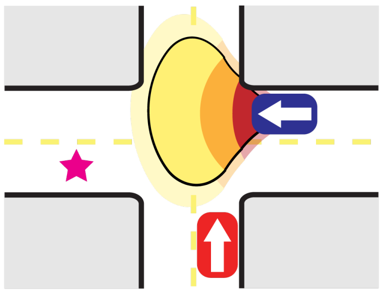
2.1.1 Conformal Prediction

Conformal prediction [54, 1] is a class of uncertainty quantification methods for constructing prediction sets that satisfy a significance level (false negative rate) requirement. Traditionally, conformal prediction creates empirical histograms of measures of risk, called non-conformity scores, and uses these to estimate prediction intervals. Classical techniques include split conformal prediction, which creates empirical histograms from hold-out sets, and full conformal prediction, which creates empirical histograms using all available data [54]. Inductive conformal prediction, a variant of split conformal prediction, uses a non-conformity score that measures distance between train and test data [9]. These methods require that the data are identically distributed and exchangeable (any permutation of data points are identically distributed). Methods such as [49] relax the requirement for identically distributed data, and [58, 5] relax the exchangeability requirement. Adaptive Conformal Inference [21] and Rolling Risk Control (RollingRC) [17] have been proposed to relax all assumptions by further calibrating the significance level to match a desired error rate. We adapt RollingRC to provide safety assurances in any multi-agent scenario.

2.1.2 Probabilistic Reachability Frameworks

In this work, we introduce a method to generate probabilistic reachable sets to account for agents' dynamics. Previous work in this space typically involves randomly generating inputs and observing corresponding outputs of a dynamics model, with some associated guarantees in the sampling process [14]. Other methods, much like ours, leverage neural network uncertainty [36]. Specifically, the method of Nakamura and Bansal [36] uses Gaussian mixture models (GMMs) output by a trajectory forecasting model to generate parametric confidence intervals on control actions, which are then used as control bounds in reachability calculations. In our work, we attempt to relax assumptions that the control actions follow any parametric distribution by applying non-parametric inference techniques.


(a) Output of Trajectron++ in a simple scene with two vehicles.


(b) Approximate (opaque) and calibrated (translucent) reachable sets.

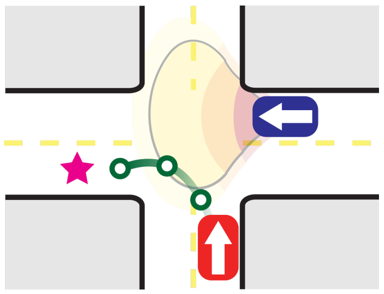

(c) Probabilistically-safe plan generated by the reachability-based planner.

Figure 2.2: Visualization of the running example. The autonomous ego vehicle is shown in red, and the human driver is shown in blue. The ego vehicle aims to navigate to the pink star while avoiding a collision with the human-driven vehicle. Confidence sets for the next three prediction steps are shown. In Figure 2.2b, the redder regions represent confidence sets for earlier prediction timesteps, and the translucent regions represent conformal prediction's calibration effect.

2.1.3 Safety Assurances in Trajectory Prediction

Various methods for incorporating uncertainty quantification have been examined for the purposes of providing safety assurances in trajectory prediction problems. Some methods provide probabilistic assurances by inferring parameters of a distribution on an agent's control actions [3, 19]. Methods such as [32] and [15] estimate confidence intervals with conformal prediction, implicitly leveraging prediction uncertainty through the non-conformity measure. Specifically, the method of Luo et al. [32] uses split conformal prediction to create a warning system, alerting drivers of "dangerous" situations. These warnings can be transformed into confidence sets, as shown in [15], which additionally eliminates exchangeability assumptions and incorporates trajectory optimization, much like our approach. Other methods emphasize the design process of the trajectory prediction neural networks, for instance by opting to use ReLU networks [12] or by opting to incorporate conformal prediction in the neural network's loss function [50]. In contrast to our approach, none of these methods consider the dynamic feasibility of confidence sets, and some methods that investigate conformal prediction, such as [12] and [32], assume exchangeability. Our method relaxes these assumptions while providing interpretability in the uncertainty quantification process and dynamic feasibility in confidence sets.

2.2 Assurances from Uncertainty

Our approach can be summarized in four primary steps: trajectory forecasting with uncertainty quantification (Section 2.2.1), leveraging uncertainty to obtain approximate prediction intervals (Section 2.2.2), calibrating approximate prediction intervals (Section 2.2.3), and obtaining dynamically feasible prediction sets (Section 2.3.1). We summarize our algorithm in Section 2.3.2, and we apply our approach to ego agent planning tasks in Section 2.3.3.

Running Example: *To motivate and illustrate our method, we introduce a simple running example with two vehicles at an intersection, one of them designated as the "ego" vehicle. In Figure 2.2, the autonomous ego vehicle, shown in red, aims to safely navigate to the pink-colored star while avoiding the blue vehicle.*

2.2.1 Trajectory Forecasting Model

We start by assuming access to a known dynamics model for each agent and a trajectory forecasting model capable of predicting an agent's control input. This trajectory forecasting model may maintain the capability to predict control actions for multiple agents at once, while considering interactions between agents. We denote the model as $f_T(\cdot) : \mathcal{X} \to \mathcal{U}$, where $\mathcal{X}$ is some arbitrary input space and $\mathcal{U}$ is a space over control actions. The network predicts $\mathbf{u}_{t:t+h} \in \mathcal{U}$, which is a collection of control action vectors indexed by timesteps t through $t+h$, for each of N total agents. Here, h is a fixed prediction horizon. We also assume the existence of an uncertainty measure on the network's outputs, denoted as $\sigma_T(\cdot) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^d$, with d as the dimension of uncertainty representation. For example, a variance prediction or a variance estimate from inference-time dropout [24] is a valid uncertainty measure. Additionally, some neural network architectures, such as Trajectron++ [40], provide alternative uncertainty measures. This network architecture predicts a GMM over possible control actions, leading to understandings of prediction uncertainty such as the variance of the GMM's modes.

Running Example: *Given some sequence of the other vehicle's position history, Trajectron++, our trajectory forecasting model of choice, predicts a GMM in action space, and then integrates the actions to obtain states. We assume that vehicles follow the extended Dubins' car dynamics model. The state of this system is* $\mathbf{x} = \begin{bmatrix} x & y & v & \theta \end{bmatrix}^\top$, *and the dynamics are given by* $\dot{\mathbf{x}} = \begin{bmatrix} v\cos(\theta) & v\sin(\theta) & u_1 & u_2 \end{bmatrix}^\top$. *The variance of the highest-probability Gaussian component, among other features of the GMM, are incorporated into the design of the uncertainty measure.*

2.2.2 Estimating Model Error from Uncertainty

To obtain confidence intervals on a black-box model's outputs, we estimate the neural network's confidence in an online manner, correlating its prediction uncertainty with prediction error. Quantile regression models enable us to map heuristic notions of uncertainty to an *approximate* confidence interval [26] along each action dimension. We choose a linear model since its simple parametrization allows for fast online updates and interpretability in how it perceives uncertainty. We demonstrate an example of interpretability in Section 2.4.4. Intuitively, our quantile regression models are approximately "calibrating" the network's uncertainty to obtain an estimate of its error.

As we observe new datapoints online, we collect $\mathbf{u}_{t-h:t}$, the last h ground truth control actions prior to timestep t. In practice, we estimate control actions by observing the state history of an agent, and then numerically computing derivatives to estimate actions from an assumed dynamics model. We contrast $\mathbf{u}_{t-h:t}$ with the network's previous prediction h timesteps ago, i.e., $\widehat{\mathbf{u}}_{t-h:t}$, and define $\mathbf{e}_{t-h:t} \coloneqq \mathbf{u}_{t-h:t} - \widehat{\mathbf{u}}_{t-h:t}$ as the prediction error.

Now, suppose we require a $1-\alpha$ approximate confidence interval on the ground truth control action. We can construct two quantile regression models $\widehat{q}_{\frac{\alpha}{2}} : \mathbb{R}^d \to \mathcal{U}$ and $\widehat{q}_{1-\frac{\alpha}{2}} : \mathbb{R}^d \to \mathcal{U}$ where $\widehat{q}_\varepsilon$ estimates the ε-quantile on the network's prediction error from timesteps t to $t+h$ for each agent, denoted $\widehat{\mathbf{e}}_\varepsilon$. For notational convenience, let us denote $\mathbb{P}_t(A)$ as the probability of event A conditioned on information until time t. We obtain an approximate $1-\alpha$ confidence interval as follows:

$$\begin{aligned}
&\mathbb{P}_t\big(\widehat{\mathbf{e}}_{\frac{\alpha}{2}} \le \mathbf{e}_{t:t+h} \le \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}}\big) && (2.1)\\
&= \mathbb{P}_t\big(\widehat{\mathbf{e}}_{\frac{\alpha}{2}} \le \mathbf{u}_{t:t+h} - \widehat{\mathbf{u}}_{t:t+h} \le \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}}\big) && (2.2)\\
&= \mathbb{P}_t\big(\widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{\frac{\alpha}{2}} \le \mathbf{u}_{t:t+h} \le \widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}}\big) && (2.3)\\
&\approx 1-\alpha. && (2.4)
\end{aligned}$$

Thus, our approximate $1-\alpha$ confidence interval on $\mathbf{u}_{t:t+h}$ is $\widehat{\mathcal{I}}_{t:t+h} = [\widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{\frac{\alpha}{2}}, \widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}}]$.

Traditionally, quantile regression models are trained using computationally expensive linear programs [26], so instead, we opt for a faster, online gradient descent approach. We define our loss function for the quantile regression model $\widehat{q}_\varepsilon$ to be $\mathcal{L}(y, \widehat{y}) = (y - \widehat{y})\varepsilon\mathbf{1}\{y \ge \widehat{y}\} + (\widehat{y} - y)(1-\varepsilon)\mathbf{1}\{y < \widehat{y}\}$ (the "pinball loss") [44]. We set y to be the true model error, $\mathbf{e}$, and $\widehat{y} = \boldsymbol{\beta}^\top\boldsymbol{\sigma}$, where $\boldsymbol{\beta}$ represents the weights of the regression model, and $\boldsymbol{\sigma}$ is the uncertainty measure from the trajectory forecasting model. We update the weights according to $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \zeta\nabla_{\boldsymbol{\beta}}\mathcal{L}(\mathbf{e}_{t-h:t}, \boldsymbol{\beta})$, with learning rate ζ.

2.2.3 Calibrating Approximate Confidence Intervals

Given that the confidence intervals we obtained in the previous section are merely approximate, we aim to calibrate these intervals. To this end, we apply the RollingRC algorithm [17], which perfectly adapts to the online requirements of our method. We are motivated to use the RollingRC algorithm compared to other conformal prediction methods due to a desire to remove the data exchangeability assumption, since we allow for sequentially-dependent data and potential distribution shifts. In addition, we would like to train and calibrate the quantile regression models in a sample-efficient manner. RollingRC guarantees that the error rate deviates from α as $\mathcal{O}(1/T)$, where T is the total number of datapoints provided to the algorithm.

Following the notation from the RollingRC algorithm, we define $\theta_t \in \mathbb{R}$ as our conformal parameter, and $\varphi(\cdot) : \mathbb{R} \to \mathcal{U}$ as the algorithm's "stretching function". Now, we claim that

$$\mathbb{P}_t\big(\widehat{\mathbf{e}}_{\frac{\alpha}{2}} - \varphi(\boldsymbol{\theta}) \leq \mathbf{e}_{t:t+h} \leq \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}} + \varphi(\boldsymbol{\theta})\big) \leq 1 - \alpha - \mathcal{O}(1/t) \tag{2.5}$$

Following similar steps as before, we obtain our newly calibrated confidence interval on $\mathbf{u}_{t:t+h}$ as $\mathcal{I}_{t:t+h} = [\widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{\frac{\alpha}{2}} - \varphi(\boldsymbol{\theta}), \widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}} + \varphi(\boldsymbol{\theta})]$.

2.3 Probabilistic Reachability and Planning

2.3.1 Probabilistic Reachability among Multiple Agents

In the previous sections, we have designed a method to provide confidence intervals on agents' control actions. However, for some downstream tasks, such as safe planning, confidence sets in spatial dimensions are more desirable. Hence, we use HJ reachability to obtain spatial sets, in the form of forward reachable tubes, on each agent's location given its dynamics and the probabilistic bound on control [41]. This procedure asserts that an agent's location will be contained in the produced reachable tube with probability $1 - \alpha$.

Suppose we wish to upper bound the probability that the ego vehicle collides with any agent. Let $\mathbf{x}_t^{(i)}$ be the location of non-ego agent i at timestep t and $\mathcal{S}[t]^{(i)}$ be the corresponding agent's forward reachable tube, as computed by our algorithm. We define miscoverage rate as the proportion of instances in which the ground truth position of any agent i at time t is outside $\mathcal{S}[t]^{(i)}$. We aim to obtain an upper bound on miscoverage rate, such that the ego agent can navigate in regions outside of $\mathcal{S}[t]^{(i)}$ for all $i \in \{1, \ldots, N\}$ and guarantee that the probability of collision is at most γ, a pre-specified parameter. Consequently, we set the confidence interval significance level α according to our desired total miscoverage rate γ and number of agents N.

Theorem 2.3.1 (Significance Level Correction)
Suppose that we wish to have a total miscoverage rate of γ, where total miscoverage rate is an upper bound on the probability that *any* human agent is miscovered:

$$\mathbb{P}_t\left(\bigcup_{i=1}^{N}\left\{\mathbf{x}_t^{(i)} \notin \mathcal{S}[t]^{(i)}\right\}\right) \leq \gamma. \tag{2.6}$$

We claim that the following α achieves an (asymptotic) total miscoverage rate of γ for N human agents:

$$\alpha = 1 - (1 - \gamma)^{\frac{1}{N}}. \tag{2.7}$$

The proof of Theorem 2.3.1 is available in Section A.1, which uses the fact that the N agents act independently conditioned on past information. Since α must be a fixed quantity in our algorithm, we must also fix N. Hence, we fix our algorithm to only consider the N agents closest to the ego vehicle.

Running Example: *Suppose we want a 95% probability safety assurance. Since there is only one other vehicle, we get $\alpha = 0.05$ from Theorem 2.3.1. Given the previous predictions of the blue agent's trajectory, we generate uncalibrated, time-indexed intervals on ranges of possible control actions, denoted $\widehat{\mathcal{I}}_t, \widehat{\mathcal{I}}_{t+\Delta t}, \widehat{\mathcal{I}}_{t+2\Delta t}$. We calibrate these using conformal prediction to obtain $\mathcal{I}_t, \mathcal{I}_{t+\Delta t}, \mathcal{I}_{t+2\Delta t}$. As we explain in the next subsection, HJ reachability allows us to take any sequence of intervals on control actions and generate a time-indexed set of states. In Figure 2.2b, we distinguish the effects of quantile regression and RollingRC's calibration.*

2.3.2 Full Algorithm

In Algorithm 1, we demonstrate the final algorithm to generate probabilistic reachable sets. The HJReachability function generates reachable sets given a probabilistic range of control actions, $\mathcal{I}_{t:t+\Delta t}$. Since the

Algorithm 1 Conformal Reachability Calibration.

```
1: procedure GenerateSets($\boldsymbol{\theta}$, $\widehat{\mathbf{u}}_{t:t+h}$, $\boldsymbol{\sigma}$)
2:     $\widehat{\mathbf{e}}_{\frac{\alpha}{2}} \leftarrow \widehat{q}_{\frac{\alpha}{2}}(\boldsymbol{\sigma})$ ▷ Obtain lower $\frac{\alpha}{2}$ quantile
3:     $\widehat{\mathbf{e}}_{1-\frac{\alpha}{2}} \leftarrow \widehat{q}_{1-\frac{\alpha}{2}}(\boldsymbol{\sigma})$ ▷ Obtain upper $\frac{\alpha}{2}$ quantile
4:     $\mathcal{I}_{t:t+h} \leftarrow \Big[\widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{\frac{\alpha}{2}} - \varphi(\boldsymbol{\theta}),$
           $\widehat{\mathbf{u}}_{t:t+h} + \widehat{\mathbf{e}}_{1-\frac{\alpha}{2}} + \varphi(\boldsymbol{\theta})\Big]$
5:     $\mathcal{S} \leftarrow []$
6:     for $t' \in \{t, t+\Delta t, \ldots, t+h\}$ do
7:         $\mathcal{S}[t'] \leftarrow$ HJReachability($\mathcal{I}_{t'}$)
8:     return $\mathcal{S}, \mathcal{I}_{t:t+h}$
9: procedure Update($\boldsymbol{\theta}$, $\mathbf{u}_{t-h:t}$, $\mathcal{I}_{t-h:t}$)
10:    for $t' \in \{t, t+\Delta t, \ldots, t+h\}$ do
11:        $\boldsymbol{\theta}\{t'\} \leftarrow \boldsymbol{\theta}\{t'\} + \xi\big(\mathbf{1}\{\mathbf{u}_{t'-h} \notin \mathcal{I}_{t'-h}\} - \alpha\big)$
12:    GradientDescent($\widehat{q}_{\frac{\alpha}{2}}$, $\mathbf{u}_{t-h:t}$)
13:    GradientDescent($\widehat{q}_{1-\frac{\alpha}{2}}$, $\mathbf{u}_{t-h:t}$)
14:    return $\widehat{q}_{\frac{\alpha}{2}}$, $\widehat{q}_{1-\frac{\alpha}{2}}$, $\boldsymbol{\theta}$
15: procedure Main($\gamma$, $N$)
16:    $\alpha \leftarrow 1 - (1-\gamma)^{\frac{1}{N}}$
17:    $\boldsymbol{\theta}\{t, t+\Delta t, \ldots, t+h\} \leftarrow 0$
18:    $\widehat{q}_{\frac{\alpha}{2}}, \widehat{q}_{1-\frac{\alpha}{2}} \leftarrow$ InitializeRandomWeights( )
19:    $\mathbb{I} \leftarrow \{\}$
20:    $t \leftarrow 0$
21:    while true do
22:        $\widehat{\mathbf{u}}_{t:t+h}, \boldsymbol{\sigma} \leftarrow f_T(\cdot), \sigma_T(\cdot)$ ▷ Get trajectory predictions and uncertainty from model
23:        $\mathcal{S}, \mathcal{I}_{t:t+h} \leftarrow$ GenerateSets($\boldsymbol{\theta}$, $\widehat{\mathbf{u}}_{t:t+h}$, $\boldsymbol{\sigma}$)
24:        $\mathbb{I} \leftarrow \mathbb{I} \cup \mathcal{I}_{t:t+h}$
25:        if $t \geq h$ then
26:            $\mathbf{u}_{t-h:t} \leftarrow$ ObserveHistory( )
27:            $\mathcal{I}_{t-h:t} \leftarrow \mathbb{I}[t-h:t]$
28:            $\widehat{q}_{\frac{\alpha}{2}}$, $\widehat{q}_{1-\frac{\alpha}{2}}$, $\boldsymbol{\theta} \leftarrow$ Update($\boldsymbol{\theta}$, $\mathbf{u}_{t-h:t}$, $\mathcal{I}_{t-h:t}$)
29:        $t \leftarrow t + \Delta t$
```

range can differ over time (e.g., $\mathcal{I}_t \neq \mathcal{I}_{t+\Delta t}$ necessarily), we iteratively compute time-indexed forward reachable tubes by computing the forward reachable tube over $[t, t+\Delta t]$ and using the reachable *set* at $t+\Delta t$ as the initial condition to compute the reachable tube over $[t+\Delta t, t+2\Delta t]$. We also utilize a GradientDescent function that updates the weights of the quantile regression models as described in Section 2.2.2. In Algorithm 1, ξ is the "learning rate" associated with the RollingRC algorithm.

2.3.3 Safe Planning Framework

Given the time-indexed sets $\mathcal{S}[t] \subseteq \mathcal{S}[t+\Delta t] \subseteq \cdots \subseteq \mathcal{S}[t+h]$, we desire that the autonomous agent's location at time t' is outside $\mathcal{S}[t+k\Delta t]$, where $t+(k-1)\Delta t \leq t' \leq t+k\Delta t$. We can plan by treating each agent's time-indexed forward reachable tube as a dynamic obstacle that grows with time. The obstacle-aware planning requirement motivates the application of a forward reach-avoid tube for the ego agent [18, 4]. We use this to derive an optimal control trajectory by selecting the Hamiltonian-maximizing control trajectory to a desired final state within the forward reach-avoid tube. In practice, this trajectory can involve bang-bang control, so one can track it using a tracker with a provable tracking error bound, such as a constrained iterative linear quadratic regulator. The planner's output is visualized in Section A.3.1.

Running Example: *From the previous section, we obtained $\mathcal{S}[t], \mathcal{S}[t+\Delta t], \mathcal{S}[t+2\Delta t]$ as a probabilistic occupancy region on the location of the other vehicle. Now, we can use the time-varying avoidance regions to plan a safe path to the goal in Figure 2.2c. Notice that the planner allows the ego agent to traverse in the yellow-colored region: it is aware that the ego vehicle would not violate the safety assurance as it can leave the yellow region by the time the other agent would enter it.*

2.4 Results

Table 2.1: Coverage Rates and Set Sizes for $1 - \gamma = 0.95$.

Methods	1st (.5s)	2nd (1s)	3rd (1.5s)	4th (2s)	5th (2.5s)	6th (3s)
	Coverage Rates for Prediction Step					
nuScenes Dataset						
Nakamura and Bansal	0.926 ±0.012	0.854 ±0.017	0.816 ±0.023	0.842 ±0.023	0.868 ±0.022	0.902 ±0.020
Luo et al.	1.000 ±0.000	1.000 ±0.000	1.000 ±0.000	0.998 ±0.002	0.989 ±0.006	0.968 ±0.012
Our Method	0.964 ±0.008	0.962 ± 0.011	0.968 ±0.010	0.975 ±0.008	0.981 ±0.007	0.985 ±0.007
Waymo Dataset						
Nakamura and Bansal	0.981 ±0.007	0.954 ±0.012	0.938 ±0.014	0.937 ±0.015	0.952 ±0.014	0.955 ±0.015
Luo et al.	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	0.998 ±0.002	0.985 ±0.009	0.955 ±0.015
Our Method	0.997 ±0.002	0.986 ±0.006	0.980 ±0.008	0.967 ±0.011	0.965 ±0.011	0.965 ±0.013
	Set Sizes for Prediction Step					
nuScenes Dataset						
Nakamura and Bansal	57 ±2	320 ±12	886 ±35	2060 ±85	3259 ±128	4285 ±160
Luo et al.	425 ±11	523 ±16	683 ±34	1097 ±109	1426 ±140	1814 ±186
Our Method	39 ±3	157 ± 13	462 ±39	1078 ±88	2150 ±170	3713 ±268
Waymo Dataset						
Nakamura and Bansal	64 ±7	311 ±32	951 ±96	2117 ±195	3814 ±328	5892 ±475
Luo et al.	448 ±8	736 ±43	1110 ±94	1568 ±169	2126 ±237	2687 ±301
Our Method	61 ±6	246 ±27	655 ±71	1361 ±143	2422 ±240	3885 ±365

We compare the empirical safety and efficiency of our contribution to two baselines, Online Update of Safety Assurances Using Confidence-Based Predictions by Nakamura and Bansal [36] and Sample-Efficient Safety Assurances using Conformal Prediction by Luo et al. [32]. For both baselines, we perform the significance level correction described in Section 2.3.1.

For all benchmarking purposes, we use Trajectron++ trained on the relevant datasets. We follow the same architecture and hyperparameters as [40] by using 4 seconds (8 steps) of history to predict 3 seconds (6 steps) into the future. This is consistent with the other baselines' approaches. Set sizes are shown in square meters. We use a pre-specified total miscoverage rate of $\gamma = 0.05$, and we generate predictions for the closest $N = 3$ agents, which strikes a balance between the speed of our HJ reachability calculations and the practical safety of the system.

2.4.1 nuScenes Dataset Results

We compare the coverage rate and efficiency of our method against the two baselines on nuScenes self-driving data [10]. We calculate average coverage rate and average set sizes individually for each forward prediction step $t, t + \Delta t, \ldots, t + h$ on 100 randomly sampled scenes. For each scene, we use the first 13 seconds to calibrate each method and make predictions on the last 5.5 seconds. Table 2.1 shows step coverage and set sizes at all prediction steps. Note that an ideal algorithm maintains a coverage rate over $1 - \gamma$ while providing the smallest prediction sets.

2.4.2 Waymo Open Motion Dataset Results

To demonstrate the planning safety and efficiency of each method, we also perform experiments on the Waymo Open Motion Dataset [16], coupled with the Nocturne simulator [53]. This allows us to apply control actions to the ego vehicle while all other agents replay their respective sequences of control actions from the dataset. We use the same planning method discussed in Section 2.3.3 for all three methods, since

Table 2.2: Waymo Planning Benchmarks

Method	**Progress to Goal**	**Collision Rate**	**Conservatism**
Nakamura and Bansal	0.494 ±0.029	**0.0**	**1.504** ±0.068
Luo et al.	0.305 ±0.028	0.005	1.626 ±0.072
Our Method	**0.544** ±0.028	**0.0**	1.507 ±0.068

neither of the baselines have associated planners. For each scene, we calibrate using the first 7 seconds and use model-predictive control to plan for the last 3 seconds, where the goal is the final position of the ego vehicle in the ground truth data. We measure three quantities: (1) progress to goal, defined as the ratio of the distance from the final state of the ego vehicle to the goal compared to the distance from the start to the goal, subtracted from 1; (2) collision rate; (3) conservatism of each method compared to the ego vehicle's ground truth trajectory, defined as the ratio of minimum distance between the ego vehicle to other agents at all times as a result of the planner, compared to that of the ground truth. The formulas and computations of these metrics are described in detail in Section A.2. In Section A.3, we additionally demonstrate the impact of the aforementioned theoretical guarantees by providing safety and efficiency metrics in the absence of conformal prediction. We performed the benchmarks on 200 randomly sampled scenes. Table 2.1 depicts coverage rates and set sizes for all prediction timesteps. Table 2.2 depicts average collision rate, average progress to goal, and conservatism.

2.4.3 Discussion of Results

For the nuScenes dataset, we notice that our method achieves more efficient set sizes for initial prediction steps, while Luo et al. achieves more efficient set sizes for later prediction timesteps. Nevertheless, neither of these two methods violates the miscoverage requirement of $\gamma = 0.05$. The method of Nakamura and Bansal violates the miscoverage rate, however, supporting the introduction of uncertainty calibration into the algorithm. Hence, calibrating neural network uncertainty is important, not only to provide the desired coverage rate but also to generate efficient prediction sets.

For the Waymo dataset, we notice a very similar phenomenon with set sizes and coverage rates. In the planning benchmarks, our method has the best progress to goal, likely due to the initial-timestep sets being smaller. This is also reflected in the conservatism scores, with reachability-based methods performing the best. The method of Luo et al. also encountered one collision scenario in which the produced set was very large and forced the planner to take a sharp avoid action. Thus, we note the importance of initial-timestep sets being small to allow the reachability-based methods to perform better in the planning benchmarks. This allows the ego vehicle to make some progress, whereas a large initial-timestep set would inhibit any progress regardless of the relative size of later timesteps' sets.

2.4.4 Case Studies

Understanding Uncertainty Measures

In this case study, we demonstrate the usefulness of our interpretable quantile regression model when understanding the efficacy of uncertainty metrics. Consider the scene in Figure 2.3a. We choose the uncertainty measure based on properties of the GMM, including the distance between peaks and the (co)variance of the highest-weighted mode. The learned regression model indicates a positive correlation between prediction error and variance of the most-likely GMM mode. In Figure 2.3b and Figure 2.3c, we can visually discern the benefit of including these features.

Overall, this case study shows the importance of understanding the usefulness of different components of the uncertainty measure. A more useful uncertainty metric can provide more efficient sets, since a more accurate quantile regression model would require less calibration (less "stretching" from conformal prediction). Conformal prediction cannot derive confidence intervals conditional on some input, so quantile regression's accuracy is crucial for providing efficient sets.

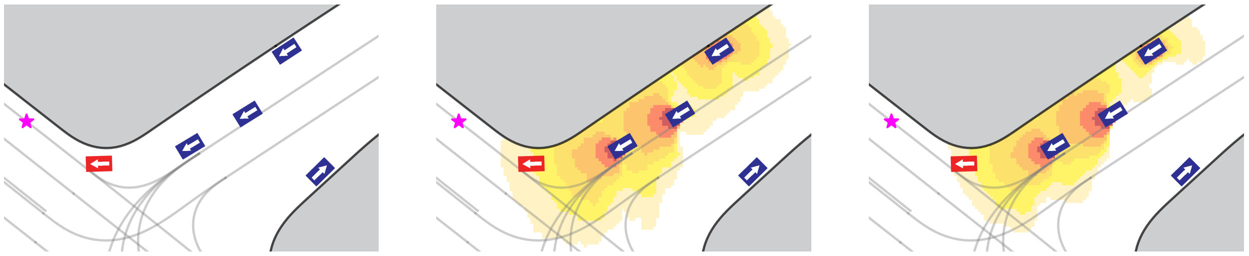

(a) Sample scenario in which we observe the reachable sets of the three agents closest to the ego vehicle. (b) Calibrated confidence sets generated by quantile regression **without** covariance features. (c) Calibrated confidence sets generated by quantile regression **with** covariance features.

Figure 2.3: Case Study of Uncertainty Metrics. We demonstrate a simple example in which the choice of uncertainty measure affects the size of sets, with coverage rate held constant.

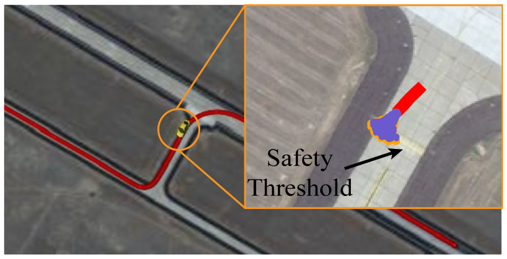


Figure 2.4: Our algorithm is applied to assure safety in potential runway incursion scenarios. Once the ground vehicle is determined to have crossed a designated safety threshold, the aircraft is cleared to land.

Safety in Aerospace Applications

In this case study, we apply our algorithm to satisfy a real-world safety assurance requirement by demonstrating our algorithm on Boeing vehicles. We consider the case of an aircraft attempting to land on a runway while accounting for potential runway incursions from ground vehicles. We use our algorithm to provide assurances on the motion of a ground vehicle on the runway. Given a fixed landing plan for the plane, we adapt the sets from our algorithm to design a warning system similar to Luo et al.'s original algorithm. If a prediction set intersects the runway, a warning is issued. A visualization of this application is shown in Figure 2.4. The ground vehicle's state history is shown in red, and its uncalibrated prediction set is shown in purple. The "stretching" effect from conformal prediction is shown in orange.

2.5 Discussion and Future Work

In this paper, we introduced a non-parametric approach to using interpretable uncertainty measures from black-box models for generating calibrated prediction intervals. We demonstrated an efficient reachability-based approach to generating prediction sets, and we showed the goal-oriented efficiency and safety of our algorithm in planning tasks for an ego agent through simulations and real-world experiments.

For future investigations, we would be interested in seeing the effects of longer planning horizons. Although many state of the art models cannot provide reliable predictions for agent behavior 1 minute into the future, for example, we would like to see the efficiency of our method compared to the other methods discussed. We would also like to generalize our method to arbitrary measures of risk, instead of only coverage rate. For example, one might want greater confidence in the behavior of nearby or fast-moving agents, than for agents that are far away or stationary. Thus, in certain practical scenarios, a heuristic measure of risk may be more appropriate than miscoverage rate. Along these lines, we would also like to explore adapting the confidence interval significance α for different agents depending on their properties with respect to maintaining safety. Finally, we are interested in decreasing conservatism across the pipeline to help ensure that the planning framework can always find a feasible solution to the problem.

2.6 Acknowledgements

This material is based upon work supported by the DARPA Assured Autonomy Program, the SRC CONIX program, Google-BAIR Commons, the NASA ULI program on Safe Aviation Autonomy, and the National Science Foundation Graduate Research Fellowship Program under Grant Nos. DGE 1752814 and DGE 2146752. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of any aforementioned organizations. We also thank James Paunicka, Blake Edwards, Dragos Margineantu, Douglas Stuart, and the entire team at Boeing for all their help and contributions in demonstrating our method on Boeing's runway incursion dataset. Finally, we thank Marius Wiggert for helping with the implementations in the HJ reachability toolbox.

Chapter 3

Robust Dynamics Estimation for Quadrupeds

In the search for certifying the robustness of learning-enabled systems, we turn our attention to a dynamics modeling problem[1]. In this chapter, we will discuss some methods for extending the results from the previous chapter, as well as some caveats associated with the assurance techniques mentioned previously. As shown in the chapter before, conformal prediction has proved to be a powerful tool in providing safety guarantees for learning-enabled systems, but existing methods are faced with several limiting factors that make them difficult to incorporate into robotics applications without strong assumptions. We outline these strong assumptions and motivate the search for new conformal prediction techniques below. This work is primarily a follow-up on the previous chapter, as well as some related work in [43]. Below, we describe ideal characteristics of safety certifications for systems:

1. We would like the safety certification algorithm to provide an assurance *conditional* on past information, rather than *in conjunction with* past information. When providing the guarantee, we do not want the guarantee itself to rely on the algorithm's accuracy in the past. Some existing conformal prediction methods provide guarantees on miscoverage error rates between an initial timestep and the present. However, they do not provide any guarantees on miscoverage rate on the current prediction alone (disregarding past information). We would like to determine a probabilistic bound on the trajectory prediction model's error only for the current prediction that it has provided, regardless of previous errors.

2. We would like the algorithm to provide valid guarantees in the presence of highly dependent actions. We recognize that each agent's actions are highly dependent, not only on their own past actions but also others' past actions. Hence, the algorithm would have to account for this dependence to provide valid probabilistic assurances. Furthermore, the algorithm would have to account for distribution shifts as well. Either the underlying prediction algorithm would have to adapt online, or the conformal prediction algorithm would have to detect distribution shift and notify downstream components.

3. In the instance of an out-of-distribution event, we would like to enable some notion of a fallback, until the aforementioned online algorithm has received sufficient data to adapt to the new distribution shift and the guarantees reflect that the algorithm is more confident. A key assumption here is that, for an unconfident algorithm, the guarantees will be far looser.

In this chapter, we focus on points 1 and 2 above. Our primary focus is to achieve an *anytime-valid* safety guarantee. That is, if the state of our dynamical system at timestep t is denoted by x_t, then we want to design a confidence set $\mathcal{X}_t$ such that $\mathbb{P}(x_t \in \mathcal{X}_t \, \forall t) \geq 1 - \alpha$. This chapter is organized as follows

[1]The work outlined in the following chapter was developed in collaboration with Haotian Shen from UC Berkeley, Rohan Sinha from Stanford University, and Rachel Luo from NVIDIA. The contents of this chapter were presented as a final project for CS 281B.

1. In Section 3.1, we discuss some extensions to the conformal prediction literature we introduce in Section 2.1. Specifically, we will highlight a type of conformal prediction that is adapted to non-exchangeable data regimes.
2. In Section 3.2, we demonstrate some experiments based on the literature we introduce in Section 3.1. We perform these experiments on a real quadruped deployed on UC Berkeley's Memorial Glade.
3. In Section 3.3, we conclude our initial experiments and describe our current work in achieving safety assurances for dynamics modeling problems.

3.1 Related Work

In this section, we will discuss in length the contributions presented in [5]. Given a sequence of calibration data points $(X_1, Y_1), \ldots, (X_n, Y_n)$ and a test point (X_{n+1}, Y_{n+1}), the method of Barber et al. devises a *weighted* conformal prediction scheme, which weights holdout points more similar to the test point highly than dissimilar points. Explicitly, if we define $Z_i \coloneqq (X_i, Y_i)$, $Z \coloneqq (Z_1, \ldots, Z_n, Z_{n+1})$, $Z^i \coloneqq (Z_1, \ldots, Z_{i-1}, Z_{n+1}, Z_{i+1}, \ldots, Z_n)$, and a non-conformity measure $R(\cdot)$, then this method constructs a non-uniform distribution of non-conformity scores as $w_1\delta_{R(Z_1)} + \cdots + w_n\delta_{R(Z_n)}$ where δ represents the Dirac delta function and the weights w_i sum to 1. If we construct a confidence set $C(X_{n+1})$ based on this distribution, this method defines the coverage gap as

$$\mathbb{P}(Y_{n+1} \in C(X_{n+1})) \geq 1 - \alpha - \sum_{i=1}^{n} w_i \, \mathrm{d}_{\mathrm{TV}}\big(R(Z), R(Z^i)\big) \tag{3.1}$$

where $\mathrm{d}_{\mathrm{TV}}(\cdot, \cdot)$ denotes the total variation distance. Hence, in general, we would like to set w_i inversely proportional to $\mathrm{d}_{\mathrm{TV}}\big(R(Z), R(Z^i)\big)$. Intuitively, for similar data points (where the total variation distance is smaller), we would like the weight to be higher than dissimilar data points (where the total variation distance is larger).

In general, measuring the desired total variation distance is challenging without restrictive assumptions on the data generating process. However, some methods estimate similarity through other metrics and compute weights using a surrogate similarity metric. HopCPT [2] estimates the weights in custom-weighted conformal prediction using Modern Hopfield Networks (MHN). MHN has an associative memory mechanism that can be used to identify parts of the time series where the conditional error distribution is similar. At time $t+1$, MHN provides an association vector $\boldsymbol{\alpha}_{t+1}$ representing the similarity in error distribution between the present and all the past time steps. HopCPT computes the $1-\alpha$ prediction interval in the following way:

$$\hat{C}_t^{\alpha} = \left[\hat{f}(X_{t+1}) + \hat{Q}\left(\frac{\alpha}{2}, E_{t+1}\right), \hat{f}(X_{t+1}) + \hat{Q}\left(1 - \frac{\alpha}{2}, E_{t+1}\right)\right] \tag{3.2}$$

where $\hat{Q}(\tau, E_{t+1})$ is the τ-quantile of the multiset E_{t+1} created by drawing n times from $[R(Z_i)]_{i=1}^{t}$ with corresponding probabilities $[a_{t+1,i}]_{i=1}^{t}$, which are elements of the vector $\boldsymbol{\alpha}_{t+1}$. This method only provides asymptotic guarantees, but under stronger assumptions of local exchangeability, provides a finite sample guarantee.

Conformal language generation [51] is an application of conformal prediction to Large Language Model (LLM) to generate a set of next possible tokens. Unlike the MHN soft selection of similar data points in HopCPT, they perform a hard selection using K-nearest neighbors in the embedding space of the LLM. In particular, they compute the weights of each neighbor based on squared L_2 distance:

$$w_k = \exp\left(\frac{\|\boldsymbol{z}_t^* - \boldsymbol{z}_k\|^2}{\tau}\right) \tag{3.3}$$

where $\boldsymbol{z}_t^*$ is the decoder hidden state at generation time, $\boldsymbol{z}_k$ is the decoder hidden state of the neighbor, and τ is a tunable "temperature" hyperparameter. These weights are directly used in the weighted conformal prediction scheme described above. This method demonstrates its success in large part to very sophisticated natural language embedding schemes, which provides a very accurate similarity metric in the L_2 sense. We show in the next section how we generate embeddings to adapt the KNN retrieval in a robotics setting.

3.2 Experiments

We demonstrate some experiments leveraging the results described in [51] on a real quadruped robot. Specifically, we aim to understand how well the k-nearest neighbors method generalizes to not only time-correlated data but also gradual distribution shifts. We train a deep learning dynamics model consisting of an LSTM with 256 hidden units and 10 timesteps of history, followed by a single hidden layer. In particular, we design our robot's state to include

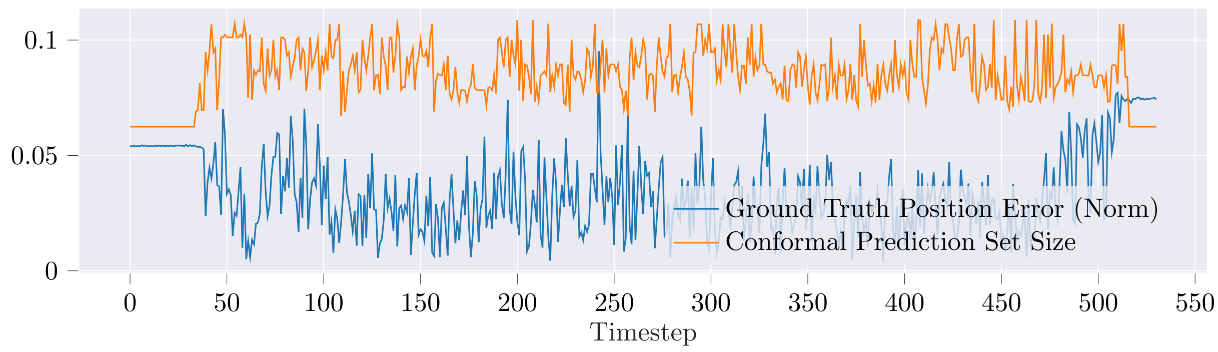
- Position, forward/lateral speed, and rotational speed
- Position and velocity data of the feet of the quadruped
- Inertial measurement unit readings such as acceleration, angular velocity, and orientation

We design our control input to mimic the input provided from the joysticks of a remote controller, i.e., the inputs provided by left joystick (corresponding to translational velocity) and right joystick (corresponding to rotational velocity). In total, our state is encoded in $\mathbb{R}^{47}$ and our control input is encoded in $\mathbb{R}^4$. Our training and evaluation data of the quadruped's movement was collected from real deployment scenarios. In total, our dataset consists of 26 different scenes with, on average, 45 seconds of data per scene. We sample data from the robot at a rate of 100 Hz.

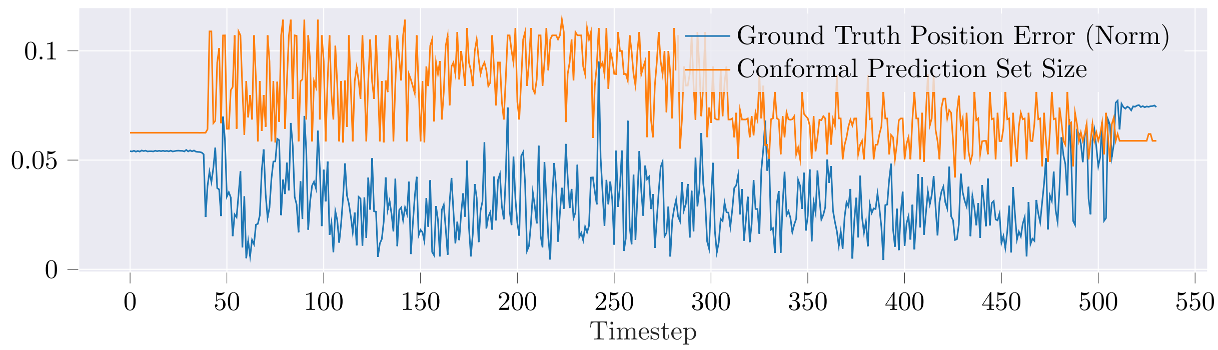
Before we apply the results from [51], we must design an embedding which represents the similarity of data points, analogous to the language-based embedding model. Since we do not have explicit labels of a distribution at our disposal, using a supervised method would be impossible. During these scenes, the robot experiences gradual distribution shift, and at several timesteps, it is impossible to accurately classify which distribution a robot is in. We opt for three choices of embeddings:

1. A manually designed embedding; this embedding consists of possibly useful signals determined from data analysis, such as the discrepancy between joystick input and realized translational/rotational

Manual Embedding: Coverage Rate 0.968



PCA Embedding: Coverage Rate 0.944



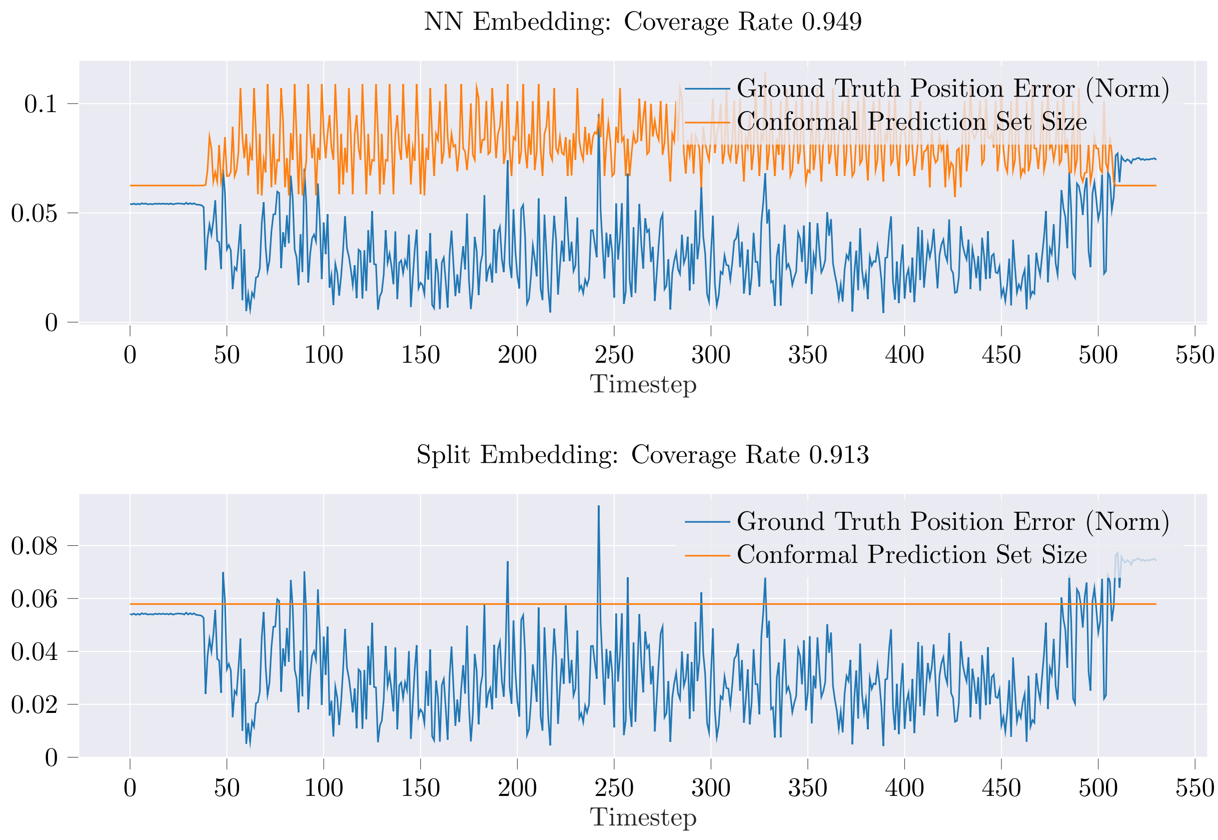


Figure 3.1: Set Sizes for "Lap Around Pool". The blue curve represents the true model error in predicting the $[x\, y\, z]$ position of the robot, and the orange curve represents the prediction set generated by each conformal prediction algorithm.

speed, the variance in robot velocity, the variance in the robot's velocity at its feet, and the robot's change in orientation.

2. An embedding derived from PCA; we collect a holdout set as is customary for split conformal, and we compute the top 10 principal component directions of the data. We then project any new data onto this subspace.
3. An embedding derived from the learned model; we take the output of the penultimate layer of the dynamics neural network (i.e., the output of the LSTM, right before the hidden layer), and use this as an embedding.

For all of these embeddings, we apply the nearest-neighbors approach with a Gaussian weighting function, arbitrarily choosing the 50 nearest neighbors. Finally, we compare the performance of all of these embeddings with vanilla split conformal, to demonstrate the importance of acknowledging data correlation. For all experiments here, we specify a miscoverage rate of $\alpha = 0.05$.

We demonstrate these results on two scenes expressing different "magnitudes" of distribution shift. The first scene, which we denote "Sidewalk to Mulch Hill", represents the robot initially walking on a sidewalk and then climbing a small hill made of mulch. The holdout set we use to generate confidence sets is the same scene applied in reverse. Coverage rates are shown in Figure 3.3, and the set sizes are shown in Figure 3.1.

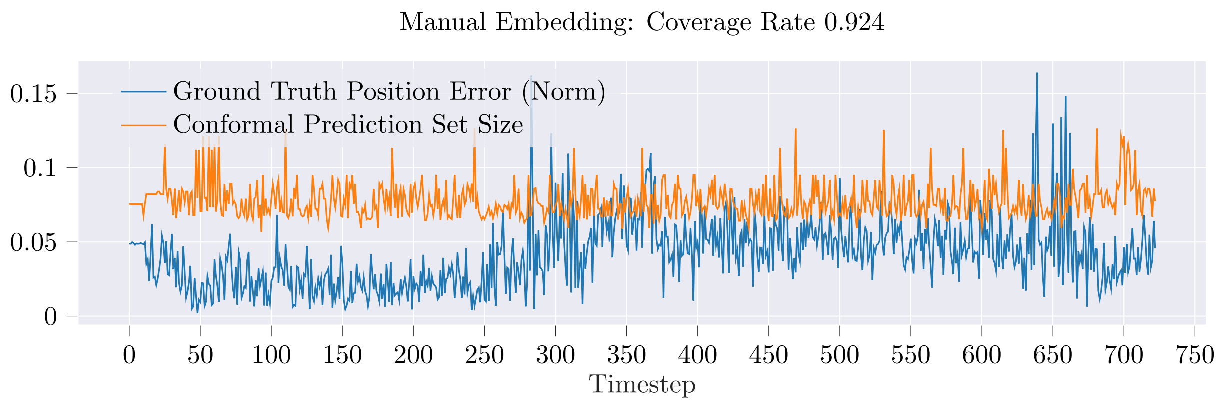
The second scene we demonstrate is denoted "Lap Around Pool", which represents the robot taking a walk around a flat ground by a pool (specifically, Memorial Pool in Berkeley). The holdout set we use here is the same lap performed in the opposite direction. Coverage rates are shown in Figure 3.4, and the set sizes are shown in Figure 3.2.

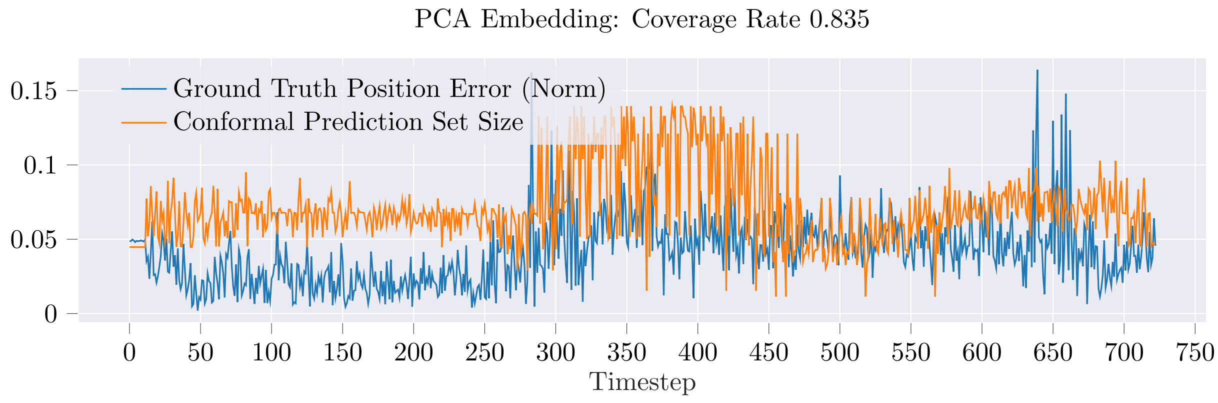
In each of these plots, the blue curve represents the true positional error of the model, i.e., its error in predicting the next $\begin{bmatrix} x & y & z \end{bmatrix}$ coordinate given the current state and control action. The orange curve represents the prediction set output by the respective conformal prediction algorithm. In general, we want the orange curve to be above the blue curve as much as possible, since this means that our conformal prediction algorithm is giving correct results. However, we would also prefer that the orange line does not extend too far above the blue line, since this would indicate larger, and hence overly conservative, prediction sets. Notice that we do not add data to conformal prediction's holdout set, contrary to the method described in Chapter 2. Hence, split conformal's set size remains constant over time. However, the highlighting characteristic is that the nearest neighbor algorithm notices that each data point holds a different "closeness" relation to data points in the calibration set. Hence, the set sizes change over time, despite not having an online component to any of the algorithms discussed here.

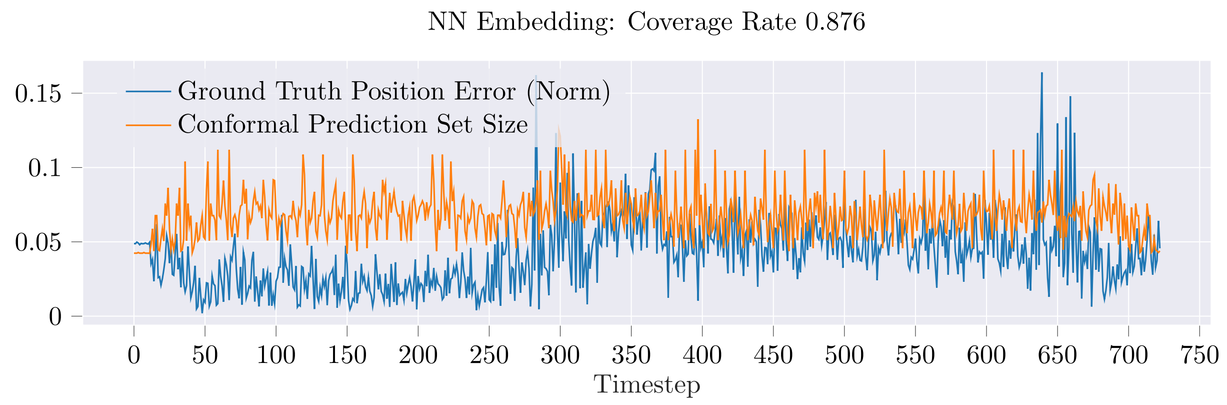
We notice that split conformal falls short of the required 95% coverage rate in all cases. We also notice that a manually-designed embedding that takes into account all failure modes of the dynamics model tends to outperform more generalized methods, such as taking the penultimate layer of the neural network or using PCA. We also notice that, in Figure 3.2, the PCA method increases set sizes drastically towards the middle of the scene, which does not entirely correspond with any drastic distribution shift since the distribution is mostly constant throughout the entire scene. The only distribution shift that occurs is the robot going faster towards the start of the scene, but this does not correspond to the perceived set size increase. However, PCA does seem to decrease the set size towards the end of the scene in Figure 3.1, which does correspond with the robot stabilizing on the top of the hill in the scene.

3.3 Conclusion and Continued Work

We notice that including an embedding that somewhat represents the similarity between data points drastically improves the quality of the sets, even if the method that incorporates this embedding is entirely a heuristic. Unfortunately, as this method is a heuristic, we still do not have a method to provide a theoretically sound guarantee. As such, we are currently exploring directions to relate the coverage gap, in terms of the total variation distance, to the weights that we compute using this heuristic.







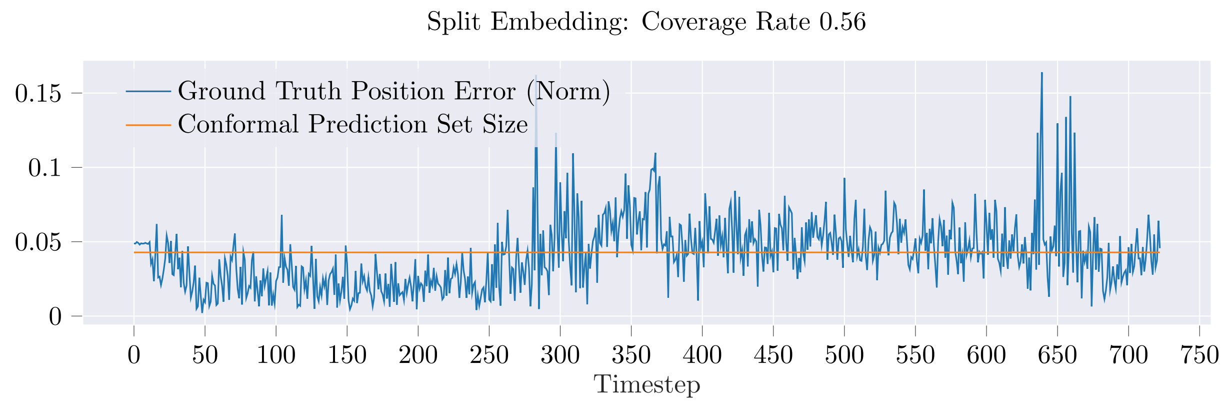


Figure 3.2: Set Sizes for "Lap Around Pool". The blue curve represents the true model error in predicting the $[x\,y\,z]$ position of the robot, and the orange curve represents the prediction set generated by each conformal prediction algorithm.

To start off this process, the work presented by [5] provides some additional insights as to how coverage gap can be computed in cases where we can measure "change points" of distributions or in cases of bounded total variation distance. One way we can confirm such a bound is to assume that our data follows some behavior according to a Markov chain. Since the data we collected specifically originates from a handful of measurable distributions, it is likely possible to encode the error dynamics of our model into a finite-state Markov chain, under some mild assumptions.

Furthermore, we also recognize that our algorithm does not operate in an online manner. The method of Barber et al. allows modifications such that we can accumulate more data in our calibration set online and still maintain probabilistically correct confidence sets. Hence, we would like to adapt a version of our online, quantile regression algorithm from Chapter 2 and apply it to the dynamics modeling problem. Although the dynamical system's state is estimated from noisy inertial measurement units in this case, we can use the real time state estimation data as the realized ground truths corresponding to our model's predictions.

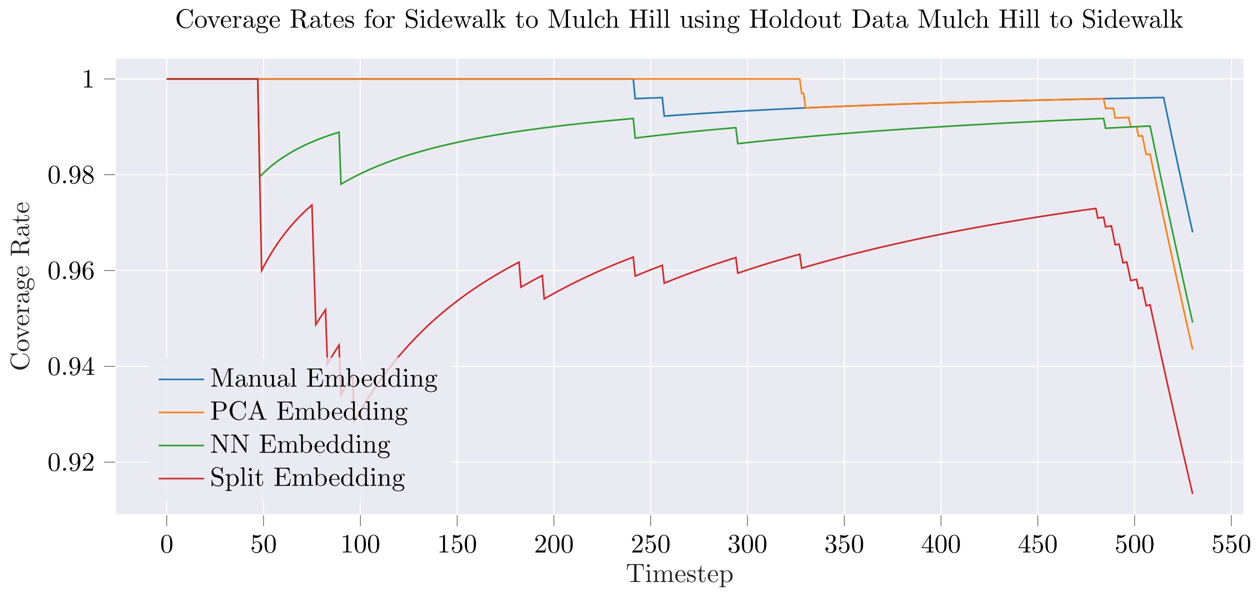


Figure 3.3: Coverage Rates for “Sidewalk to Mulch Hill”. Each curve represents a conformal prediction method’s coverage over time.

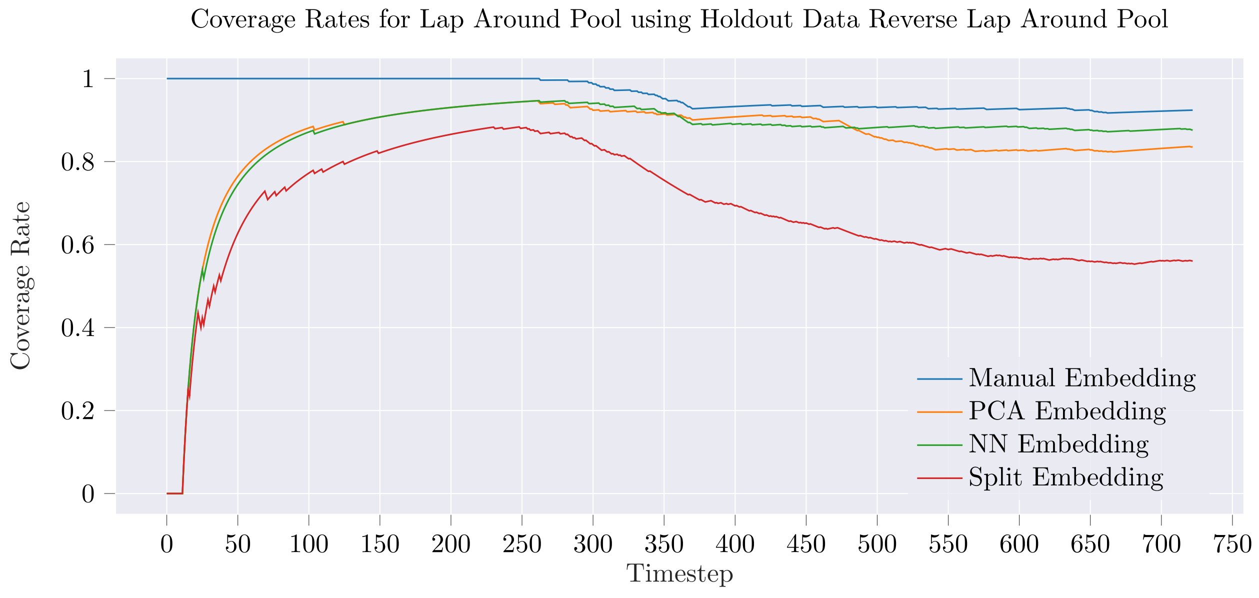


Figure 3.4: Coverage Rates for “Lap Around Pool”. Each curve represents a conformal prediction method’s coverage over time.

Chapter 4

Uncertainty-Aware Object Tracking

In many robotics applications, making predictions and observing the corresponding ground truth information is not always possible. Hence, providing assurances from a statistical perspective can be challenging, since compounding prediction error will lead to prohibitively large confidence sets. Hence, the framework of translating epistemic uncertainty into assurances is not always applicable. Nevertheless, we can use uncertainty quantification to make an empirically robust algorithm, subject to specific safety specifications. An example of such a robotics task is object tracking.

Object tracking algorithms infer the state of objects based on a sequence of sensor measurements over time. These algorithms leverage a measurement model to determine whether a new object has appeared in the field of vision of the sensor, if a currently-tracked object has received a new measurement (or detection), or if a tracked object has exited the sensor's field of vision. These measurement models are designed to minimize false positive and false negative rates, oftentimes using a prior of the measurement device to classify an object as a potential false positive or confirm it to be a true positive. For example, sensors such as LIDAR and radar are designed with false positive and true positive rate specifications which combine the physical attributes of the sensors with empirical measurements to provide a false alarm rate.

Object tracking is also often applied in computer vision applications. Classical variants of these algorithms rely on optical flow models to track the movement of pixels across an image [42], but more modern techniques rely on deep neural networks for measuring potential locations of objects [7]. In contrast to non-data-driven sensors as described previously, deep neural networks do not have reliable false positive and false negative rates. Furthermore, in computer vision applications, it is not possible to receive ground truth data about objects' positions in real time, without additional data sources such as GPS information. Hence, we can apply uncertainty quantification techniques to our vision model to achieve a more robust object tracker and translate model behavior to a desired false positive or false negative rate specification.

In this chapter, we will focus on an autonomous landing problem. Recently, airports have faced several illegal incursions, mostly due to human error. These incursions occur due to vehicles disobeying air traffic control and driving onto the runway, in the path of a landing plane, or pilots accidentally taxiing onto incorrect areas of the tarmac [47, 57]. Object tracking in these landing scenarios presents a unique challenge: when a plane is attempting to land, many road vehicles will appear tiny, even to well-equipped cameras. This motivates the application of small-object detection algorithms. Due to the difficulty of the problem and the large amount of noise in data for small object detection, object tracking algorithms struggle with state estimation, and ultimately, this magnifies the false negative rates, which can be highly devastating. In this chapter, we propose an uncertainty-aware tracker which improves robustness to the challenges presented, and we demonstrate our method on an existing incursion detection stack developed internally by Boeing[1].

This chapter is organized as follows:

1. In Section 4.1, we discuss related techniques in object tracking and the application of uncertainty quantification in object tracking. We also explain in depth two methods of incorporating neural networks' epistemic uncertainty in tracking models, derived from [34].

[1]This chapter was developed in collaboration with the Caravan team at Boeing and DARPA, through the Assured Autonomy grant.

2. In Section 4.2, we outline our method. In Section 4.2.1, we discuss the baseline system upon which we derive our improvements. In Section 4.2.2, we discuss our improvements to the baseline system.

3. In Section 2.4, we demonstrate our results on Boeing's dataset. In Section 4.3.1, we outline a case study where our uncertainty-enabled tracker performs markedly better than the baseline system.

4. In Section 4.4, we conclude our findings and discuss future directions.

4.1 Related Works

4.1.1 Multi-Object Tracking

Most classical object tracking frameworks can be decomposed into four steps:

1. Track Creation: this step is concerned with the creation of new tracks given some observations.

2. Data Association: this step associates observations to existing tracks, depending on the proximity of the observation to the track. Most modern methods use an approximate optimal assignment algorithm, such as the Hungarian algorithm [7] or a consensus-based auction algorithm [23].

3. Track Propagation: this step predicts the state from a sequence of observations and predicts the location of the track at the next timestep. This is typically accomplished using a Kalman filter update. This step requires the assumption of a dynamics model of tracked objects.

4. Track Deletion: this step deletes a track if we have sufficient belief that the track no longer exists for some reason. Some reasons for this happening include inconsistent/lack of detections (indicating that the track might be a false positive) or that the track has moved away from the tracked area of interest.

Tracking can be performed either in world-space (in which case, the objects are represented in 3D spatial coordinates) or in image-space (in which case, the objects are represented in 2D pixel coordinates). Reliable world-space tracking typically requires sensor fusion techniques, in the form of LIDAR or stereo cameras [60]. World-space trackers are harder to implement but rely on better-motivated track dynamics models, compared to pixel-space tracking. Oftentimes, trackers implement a fifth step in addition to the four described above. Trackers may choose to distinguish between a "probationary" track and a "full" track. A probationary track describes an object whose existence is uncertain, and a full track is the result of collecting enough data to confirm a probationary track's existence. In this setup, the track creation step would entail the creation of a *probationary* track. The fifth step is the promotion of a probationary track to a full track. Both probationary tracks and full tracks are eligible for deletion.

Our work is an extension of the application of Wald's sequential probability ratio test (SPRT) [56] described in [8] and [37]. These methods use optical flow-based tracking, tracking the movement of pixels between frames of a video without learning-enabled components. These methods also apply Wald's SPRT to choose between multiple hypotheses of trajectories.

Many works have also proposed end-to-end methods that do not delineate the steps of an object tracking algorithm as described above [31, 33, 13]. However, these approaches are prone to additional sources of uncertainty in tracking tasks, which are not easily decoupled like a classical object tracker. Particularly in OoD environments, these methods can produce undesirable behavior which, given the closed nature of an end-to-end system, would not be as preventable as a two-stage detection and tracking framework.

4.1.2 Combining Uncertainty Quantification and Object Tracking

The work in [34] introduces methods of uncertainty quantification applied to object tracking. This method combines Monte Carlo dropout with particle filtering to non-parametrically estimate a distribution on the neural network's weights. This method, while effective, can be computationally expensive. Some deep learning methods have also adapted to incorporate uncertainty, such as the algorithm in [31] which presents a modular deep learning approach for uncertainty-aware track labeling. Some methods have utilized uncertainty quantification techniques to achieve probabilistic guarantees [28, 46], using conformal prediction and probably approximately correct (PAC) guarantees to obtain more accurate track bounding boxes and better estimation of the state variance. In the following two sections, we will describe, in detail, two methods for incorporating uncertainty in object tracking algorithms.

4.1.3 Monte Carlo Dropout with Particle Filtering

In this section, we delve into a method we present in [34][2]. This method is based on a Bayesian perspective for incorporating uncertainty into a tracking paradigm. The development of our approach for incorporating object detector data uncertainty in a tracking framework is outlined in the following. We first characterize the approach we selected for training an object detector which can produce uncertainty estimates along with its outputs. We then describe an ideal tracking method which could account for that uncertainty. We show in this work that accounting for network uncertainty significantly improves tracking performance on out-of-distribution tracking tasks.

Neural Network Model

Our method assumes the existence of neural network trained to generating bounding boxes around detected objects. We require that, in addition to predicted bounding boxes, the network produces an estimate of the uncertainty it has in its detections. This uncertainty could arise from the relationship of the input image to the training data (epistemic uncertainty) or the ambiguity in the detection task on the particular image (aleatoric uncertainty) [25]. Due to its popularity and ease of implementation, we rely on Monte Carlo Dropout methods for producing these estimates of network uncertainty.

The object detection networks we consider here produce a list of bounding boxes, each corresponding to a location within the input image, as well as a confidence score, valued between 0 and 1 [29]. Typically a fixed threshold is chosen such that all bounding boxes with confidence score above the threshold are considered true detections. In our proposed Monte Carlo framework, we introduce dropout layers only in the network path responsible for producing bounding box locations. To introduce randomness into which bounding boxes are considered detections, we flip the interpretation of the confidence score to instead represent the threshold for considering that output a detection—for each pass of the network, a random variable is sampled from the uniform distribution $U : [0, 1]$ for each bounding box output, and if this random variable is less than the detection threshold, the corresponding bounding box is considered a detection. If not, the corresponding bounding box is not included as a detection.

Given an input image, running many passes of the network in this way emulates sampling from a conditional distribution of bounding boxes. We choose to represent this conditional distribution using a particle filter, given the high-dimensional network outputs. The description of our network as a conditional distribution also eliminates the need to explicitly differentiate false positives from true detections and false negatives from empty space. As such, we do not make any inherent assumptions about the uncertainty provided by the network.

Multiple Object Tracking with Detection Network Uncertainty

The objective of a multiple object tracking method is typically to maintain a representation of the distribution

$$\mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k}) \tag{4.1}$$

where $\mathbf{X}_{1:k}$ represents the set of all object tracks present at a discrete stage k, and $\mathbf{Z}_{1:k}$ is the set of all measurements received between the first stage and stage k. This distribution can be expressed, using Bayes rule, as

$$\begin{aligned}\mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k}) &= \frac{\mathbb{P}(\mathbf{Z}_k \mid \mathbf{Z}_{1:k-1}, \mathbf{X}_k)\mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k-1})}{\mathbb{P}(\mathbf{Z}_{1:k})} \\ &= \frac{\mathbb{P}(\mathbf{Z}_k \mid \mathbf{X}_k)\mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k-1})}{\mathbb{P}(\mathbf{Z}_{1:k})}\end{aligned} \tag{4.2}$$

[2]This work was developed in collaboration with Forrest Laine from Vanderbilt University and Boeing and DARPA through the Assured Autonomy Grant

Here we have used the fact that the measurements at stage k are independent of all previous measurements. The distribution $p(\mathbf{X}_k|\mathbf{Z}_{1:k-1})$ is the predicted distribution of object states at stage k, using measurements up to stage $k-1$. This can be expressed as

$$\mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k-1}) = \int_{\mathcal{X}} \mathbb{P}(\mathbf{X}_{k-1} \mid \mathbf{Z}_{1:k-1})\mathbb{P}(\mathbf{X}_k \mid \mathbf{X}_{k-1})\, \mathrm{d}\mathbf{X}_{k-1} \tag{4.3}$$

Combining these equations, a recursive update can be derived to relate $\mathbb{P}(\mathbf{X}_{k-1} \mid \mathbf{Z}_{1:k-1}) \to \mathbb{P}(\mathbf{X}_{k-1} \mid \mathbf{Z}_{1:k})$. This update serves as the foundation for all Bayesian object tracking methods.

In the context of tracking bounding box detections arising from a neural network, the bounding boxes produced by the network at stage k would constitute $\mathbf{Z}_k$ in this framework. However, in the framework we are proposing, we are assuming that we no longer work with particular measurements $\mathbf{Z}_k$, but rather a distribution of measurements induced by our network for a particular input. If we let o_k represent the input image to our network at stage k, we argue that the distribution we seek to represent is better represented as $\mathbb{P}(\mathbf{X}_k \mid o_{1:k})$. We can represent this distribution as

$$\begin{aligned}
&\mathbb{P}(\mathbf{X}_k \mid o_{1:k}) = \int_{\mathcal{Z}^k} \mathbb{P}(\mathbf{X}_k \mid o_{1:k}, \mathbf{Z}_{1:k})\mathbb{P}(\mathbf{Z}_{1:k} \mid o_{1:k})\, \mathrm{d}\mathbf{Z}_{1:k} \\
&= \int_{\mathcal{Z}^k} \mathbb{P}(\mathbf{X}_k \mid \mathbf{Z}_{1:k})\mathbb{P}(\mathbf{Z}_{1:k} \mid o_{1:k})\, \mathrm{d}\mathbf{Z}_{1:k} \\
&\underset{\text{eq. (2)}}{\overset{}{\underset{\uparrow}{=}}} \left(\int_{\mathcal{Z}} \frac{\mathbb{P}(\mathbf{Z}_k \mid \mathbf{X}_k)\mathbb{P}(\mathbf{X}_k \mid \mathbf{X}_{k-1})\mathbb{P}(\mathbf{Z}_k \mid o_k)}{\mathbb{P}(\mathbf{Z}_{1:k})}\, \mathrm{d}\mathbf{Z}_k\right) \cdot \left(\underbrace{\int_{\mathcal{Z}^{k-1}} \mathbb{P}(\mathbf{X}_{k-1} \mid \mathbf{Z}_{1:k-1})\mathbb{P}(\mathbf{Z}_{1:k-1} \mid o_{1:k-1})\, \mathrm{d}\mathbf{Z}_{1:k-1}}_{=\mathbb{P}(\mathbf{X}_{k-1} \mid o_{1:k-1})}\right) \\
&= \mathbb{E}_{(\mathbf{X}_{k-1}, \mathbf{Z}_k)\sim P_k}\left[\frac{\mathbb{P}(\mathbf{Z}_k \mid \mathbf{X}_k)\mathbb{P}(\mathbf{X}_k \mid \mathbf{X}_{k-1})}{\mathbb{P}(\mathbf{Z}_{1:k})}\right]
\end{aligned} \tag{4.4}$$

where we define the distribution

$$P_k = \mathbb{P}(\mathbf{X}_{k-1} \mid o_{1:k-1}) \cdot \mathbb{P}(\mathbf{Z}_k \mid o_k). \tag{4.5}$$

We note that the detections at stage k is conditionally independent of the observations from stages 1 through $k-1$ ($o_{1:k-1}$), given the observation at stage k (o_k). The tracks $\mathbf{X}_{1:k}$ are conditionally independent of the observations $o_{1:k}$, given $\mathbf{Z}_{1:k}$. The form of the resultant distribution in Equation (4.4) provides a natural framework a particle filter method.

Specifically, we base our approach on the RFS bootstrap filter described by Ristic et al.. At stage k, we simply sample N particles from the distribution P_k, meaning each particle $i \in \{1, ..., N\}$ is associated with a particular instance of the joint obstacle states $\mathbf{X}_{k-1}$ and bounding box outputs $\mathbf{Z}_k$, generated from the network with input image o_k.

Sampling bounding box outputs from the distribution $\mathbb{P}(\mathbf{Z}_k \mid o_k)$ is as simple as evaluating our network according to the process outlined in Section 4.1.3. Other than sampling a different observation $\mathbf{Z}_k$ for each particle, our method is as described in [39]. Each particle performs its own track birth and track death process, according to the update rules defined by [39].

Track Labeling and Outputting

An important aspect of the particle filtering approach is to be able to output a consistent "most-probable" object representation, which has consistent labels across discrete stages. After each particle updates its "belief" of objects' locations with new measurements and receives a weight, a new output frame needs to be generated. In traditional particle filtering frameworks, this is accomplished by sampling over the (categorical) distribution of particles. This approach is generally desirable with a very large (e.g. $N > 1000$) number of particles. However, we propose a binning and clustering method that can be applied to a particle filter with fewer particles (e.g. $N \approx 100$).

Firstly, assume that a most-probable representation of the objects were known at stage $k-1$, and that each of these tracks were labeled with a unique identifier (ID). We refer to this representation as the output frame. A given particle need not fully conform to the representation in the output frame. Particles are "binned" by the number of tracks that they contain. For example, all particles tracking five objects are placed in the same bin, separately from all particles tracking six objects, etc. A weight is computed for each bin, which is the sum of the weights of all particles in the bin. Within the bin, we apply a "clustering" process. For objects that have been previously labeled with a given ID, we choose to output the mean location of all tracks with this given ID. For unlabeled objects, we use a standard clustering method (e.g. k-nearest neighbors) and choose the means of the clusters as the locations we depict in the output frame. The unlabeled objects are labeled with new IDs, and all tracks in all particles are assigned an ID if they are sufficiently close to an object in the output frame.

4.1.4 Sampling Free Uncertainty with Likelihood Ratio Testing

The method discussed in the previous section requires significant computation due to the use of a sampling-based uncertainty scheme. For use in interactive systems, we developed an alternate method which retains the conceptual core of the idealized approach, but is amenable to actual deployment. The particular method we develop is tailored for the application of autonomous aircraft taxiing, although many of the design choices made could be generalized to other applications.

Pixel-Level Classification

A key aspect of the approach presented in the previous section is the ability to relate object detector data uncertainty with the location of detections in measurement space. While Monte Carlo generation of bounding boxes is one way to accomplish this, it relies on evaluating the neural network many times, which is not feasible for real-time systems. Therefore we turned to the efficient, sampling-free variance propagation technique [38] to generate estimates of uncertainty in a single network evaluation. This method approximates the variance of a Gaussian distribution centered at each output of the network, representing the network uncertainty for that value.

For bounding box outputs, we found Gaussian approximations of each bounding box location to be a poor representation of the distribution of possible network outputs. To address this, we propose training a pixel-level occupancy classifier, which is tasked with classifying whether each pixel in the input image corresponds to empty space or an obstacle. Variance propagation is then used to generate a 2D Gaussian distribution over the penultimate layer of the classifier, which is propagated through the final soft-max layer to generate a value which we interpret as the probability of that particular pixel being occupied.

Applying this computation to the output of every pixel generates a probability map of obstacle occupancy over the entire image. For any given input pixel with resultant probability of occupancy p, we convert that value p into a binary occupancy variable $o \in \{0, 1\}$, where $o = 1$ (representing occupied) if $p \geq 0.5$, else $o = 0$, and an uncertainty measure u, where $u = 2(1 - p)$ if $o = 1$, and $u = 2p$ if $o = 0$. In this way, the uncertainty value u is close to 1 when $p \approx 0.5$, and u is close to 0 when $p \approx 0$ or $p \approx 1$. This change of variable makes it easier to reason about these outputs in an object tracking framework.

Tracking Occupancy Grids

This transformed output can be interpreted as a binary mask, indicating the pixel locations of obstacles, along with uncertainty estimates, where lower uncertainty corresponds to higher confidence in the binary classification. This form of the network output, in theory, contains information regarding the spatial distribution of false negatives and false positives due to data uncertainty. For a particular image, if the output of the detector corresponding to a particular region of the image is classified as non-occupied, but has high uncertainty values, this region may correspond to a missed detection. Conversely, a region classified as occupied with a high uncertainty may correspond to a false positive.

The output at each pixel can be treated as an independent measurement on potential objects being tracked, and standard measurement association and measurement update methods can be applied. In particular, each pixel corresponding to a detection can be assigned to either an existing track or new track, and pixels corresponding to empty space can be assigned to all nearby tracks.

As before, we assume that objects are tracked in the image plane. We assume that the uncertainty in object tracks locations are maintained as a Gaussian distribution, and the uncertainty in object tracks

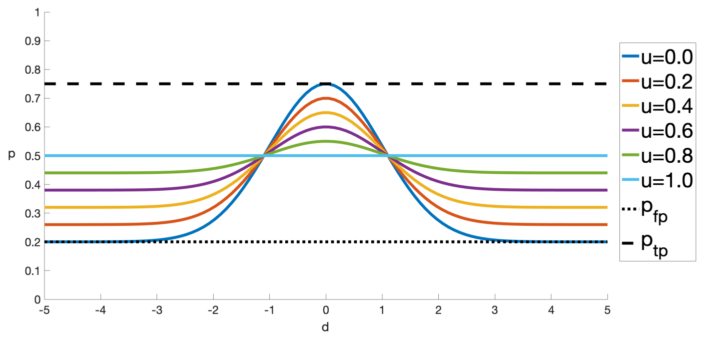


Figure 4.1: The measurement likelihood function $\mathbb{P}(o = 1, u \mid x)$ as a function of the set distance d, for various uncertainty values. Here $p_{fp} = 0.2$, $p_{tp} = 0.75$, and $\sigma = 1$. When $u = 0$, the likelihood peaks at p_{tp} when $d = 0$, and decays to p_{fp} as $|d|$ increases. When $u \to 1$, the likelihood function converges towards $\mathbb{P}(o = 1, u \mid x) = 0.5$ for all d.

is maintained as a probability of existence. Under these modeling assumptions, we propose the following method for efficiently updating these object track representations using pixel level measurements.

Assume that a representation of an object track is known at some discrete time stage k. To update this representation to reflect measurements received at stage $k + 1$, the Gaussian representation of the object state is propagated through a process model, forming a distribution over the predicted object state at stage $k+1$. Using a particle approach, many particles representing object track state vectors are sampled from this distribution. Each particle is then weighted by the product of the probabilities of measuring the independent measurements, conditioned on the object track state represented by the particle.

We define measurement likelihood for an individual pixel i with output o_i, u_i, conditioned on a particular object track state x as the following:

$$\begin{aligned}\mathbb{P}(o_i = 1, u_i \mid x) &= p_{fp} + (p_{tp} - p_{fp})\mathrm{e}^{-\frac{d_i^2}{2\sigma^2}}(1 - u_i) + \frac{u_i}{2}\\ \mathbb{P}(o_i = 0, u_i \mid x) &= 1 - \mathbb{P}(o_i = 1, u_i \mid x)\end{aligned} \tag{4.6}$$

where d_i is defined to be the set distance between the object track bounding box and the 2D region of the image plane corresponding to the pixel i, and p_{tp} is the nominal probability of a true positive detection, p_{fp} is the nominal probability of a false positive, and σ is a shaping parameter. A visualization and rationale for this measurement likelihood function is given in Figure 4.1. Contrary to the sampling-based method, the measurement model Equation (4.6) explicitly incorporates probabilities of false positive and false negatives, and can be evaluated directly in a sample-free manner.

The particle weighting w for a particle x is defined as the joint likelihood of all pixels associated to an object, where the likelihoods are assumed to be independent. Particles in the vicinity of low-uncertainty occupied pixels will have a high weight, whereas pixels with very high uncertainty will effectively assign all particles the same weight. After all particles are weighted, a posterior distribution of the track distribution is generated by fitting a Gaussian distribution to the weighted particles.

To assist in the measurement association process, we first cluster occupied pixels with neighboring (or nearly neighboring) occupied pixels. We additionally include in each cluster any unoccupied pixels which surround the cluster. We allow the unoccupied pixels to be part of multiple clusters, but occupied pixels must belong to exactly one cluster. These clusters are then either associated to existing tracks, or used to birth new tracks. For the association process, we use a greedy approach, although others could be considered.

The final step of the tracking procedure is to update the probability of existence for all tracks. The probability of existence at stage k is maintained as the ratio of probability that an object exists over probability that an object does not exist, denoted Λ_k. This ratio is updated as follows:

$$\Lambda_{k+1} = \Lambda_k \cdot \frac{\bar{w}}{\prod_{i=1}^{N} \mathbb{P}(o_i, u_i \mid \emptyset)}. \tag{4.7}$$

Here, $\bar{w}$ is the average weight taken over all particles, and

$$\begin{aligned} \mathbb{P}(o_i = 1, u_i \mid \emptyset) &= p_{fp}(1 - u_i) + \frac{u_i}{2} \\ \mathbb{P}(o_i = 0, u_i \mid \emptyset) &= 1 - \mathbb{P}(o_i = 1, u_i \mid \emptyset) \end{aligned} \tag{4.8}$$

is defined to be the probability of measuring pixel i with occupancy o_i and uncertainty u_i when there is no nearby object. In practice, when the likelihood ratio $\Lambda_{k+1} \ll 1$ after a measurement update, the object track under consideration is removed.

4.2 Method

We design our improvements on top of an existing detection and tracking stack developed by Boeing. We begin by analyzing key components of the baseline system which may provide natural implementations of neural network uncertainty. We discuss these improvements in Section 4.2.2. Our initial avenue of exploration is to follow the likelihood ratio testing idea from Section 4.1.4, where we obtain some notion of an existence probability for each track rather than a cell in an occupancy grid. We will restrict our method to be an uncertainty-aware track creation and deletion process, rather than incorporating new concepts for measurement models. In doing so, we are ensuring that our method presented here is more widely applicable across different kinds of trackers.

4.2.1 Baseline System

The baseline system can be summarized with three essential components:

- An object detection neural network which is designed and tuned to perform on small object detection tasks
- A classical object tracker which ingests the detections output by the aforementioned neural network
- A decision system that combines the tracks and declares whether the runway is clear to land

The object detection network is a fully-convolutional, one-stage object detector with no additional modifications [48]. It is trained on Boeing's proprietary aircraft landing dataset, which contains several scenes of an aircraft approaching a runway with vehicles on and around the runway. To optimize the model for the small object detection task, only detections with confidence above a fixed, hyperparameter threshold, c', are output.

The object tracker is based on a standard Gaussian process-based state estimation procedure [55], with additional hyperparameters to handle track creation, propagation, and deletion. When a series of detections are received at a particular time instant, the detections are associated to existing tracks, or contribute to spawning new tracks, according to a consensus-based auction algorithm, a derivative of the approach in [23]. When a new track is spawned, it is placed in a "probationary" state. This track is then monitored for N frames, and if it receives detections for M of N frames, then it is promoted to a full track. Otherwise, it is killed. The goal of the probationary period is to account for uncertainty in object existence, as well as to provide sufficient data for track state estimation. If a full track does not receive detections for D frames, it is deleted. Note that M, N, and D are all hyperparameters. Since these values are fixed over time, incorporating prediction uncertainty can provide improvements. For example, we may choose to increased D if we believe, through uncertainty quantification, that the neural network is encountering a high false negative rate locally in time.

The decision system maps out every track present on the runway, where the runway area is determined using the camera's homography. It examines the lifespan of each of these tracks, and if a track on the runway has been propagated for a sufficiently long period of time, the system alerts the pilot of a runway incursion. Otherwise, it advises that the pilot is clear to land.

4.2.2 Uncertainty-Aware Tracking

A natural method to approach uncertainty quantification in this instance is to incorporate a likelihood ratio testing framework. We can categorize tracks into two hypotheses: true positive and false positive tracks. By correctly classifying a track as a false positive, we can mitigate the overall number of false positives, and, by reducing the threshold c' with which we consider new detections, we can capture more potential false negatives.

Since the network we are deploying does not consider images as a sequence (it behaves in a memory-less way), considering functions of a time series of confidence scores, per object, would likely yield a useful representation of uncertainty. If we collect *all* detections that are associated with an object, whether their confidences are above c' or not, we can build a time series of confidence values per object and compute uncertainties as functions of this time series. The uncertainty we use in this instance is a rolling average of confidence scores μ_c and a rolling standard deviation of confidence score σ_c. In principle, a false positive track should have low μ_c and high σ_c, and vice versa for a true positive track. We empirically observe this relationship as we discuss in Section 4.3. In principle, any uncertainty quantification measure could replace the rolling averages and standard deviations we compute here, but we choose this measure due to its ease of computation for our real-time system. Methods such as Monte-Carlo dropout [20] would likely induce high latency.

Denote H_{FP} as the hypothesis that a track is a false positive, and H_{TP} as the hypothesis that a track is a true positive. Using the joint distribution over μ_c and σ_c, we can estimate $\mathbb{P}\left(\mu_c, \sigma_c | H_{\text{TP}}\right)$ and $\mathbb{P}\left(\mu_c, \sigma_c | H_{\text{FP}}\right)$. Next, define a likelihood ratio $\Lambda_t = \frac{\mathbb{P}\left(\mu_{c_t}, \sigma_{c_t} | H_{\text{TP}}\right)}{\mathbb{P}\left(\mu_{c_t}, \sigma_{c_t} | H_{\text{FP}}\right)}$ ($\Lambda_0 = 0$) and $S_t = \sum_{i=0}^{t} \log\left(\Lambda_i\right)$. Then, as per Wald's sequential probability ratio test [56], we can write the following decision rule for probationary tracks:

$$H = \begin{cases} H_{\text{TP}} & S_t \geq \log(\eta_1) \\ H_{\text{FP}} & S_t \leq \log(\eta_0) \end{cases} \tag{4.9}$$

Here, we define η_0 and η_1 in terms of desired false positive and false negative rates. Let α be our tracker's desired false positive rate and β be our tracker's desired false negative rate. Then, $\eta_0 := \frac{\beta}{1-\alpha}$ and $\eta_1 := \frac{1-\beta}{\alpha}$. If $\log(\eta_0) < S_t < \log(\eta_1)$, then we keep the track in its probationary state, since we have not yet collected enough data to confirm or deny its existence under the parameters we have set. When considering a similar framework for a full track, we must note that we have already "confirmed" that the track is a true positive. Hence, we are only testing whether a track now becomes a false positive (i.e., it should be deleted). To do this, we define the following decision rule for full tracks:

$$H = \begin{cases} H_{\text{FP}} & S_t \leq \log(\gamma) \\ H_{\text{TP}} & \text{otherwise} \end{cases} \tag{4.10}$$

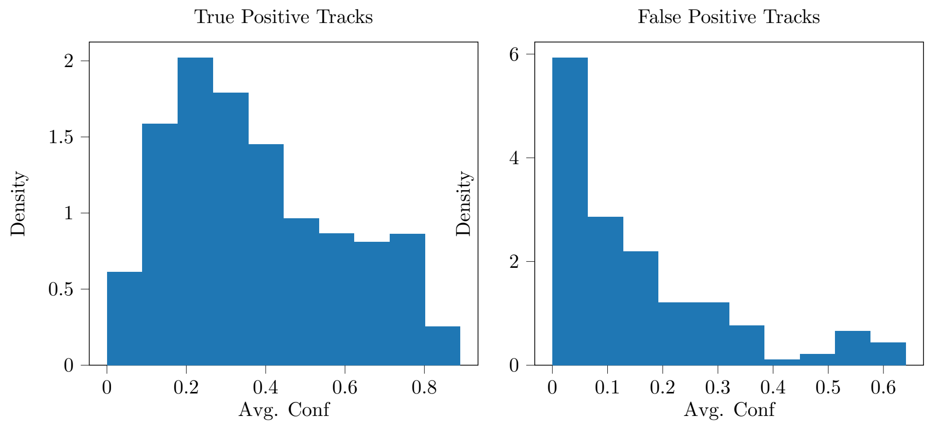
Here, we define η_0 and η_1 in terms of desired false positive and false negative rates. Let α be our tracker's desired false positive rate and β be our tracker's desired false negative rate. Then, $\eta_0 := \frac{\beta}{1-\alpha}$ and $\eta_1 := \frac{1-\beta}{\alpha}$. If $\log(\eta_0) < S_t < \log(\eta_1)$, then we keep the track in its probationary state, since we have not yet collected enough data to confirm or deny its existence under the parameters we have set. When considering a similar framework for a full track, we must note that we have already "confirmed" that the track is a true positive. Hence, we are only testing whether a track now becomes a false positive (i.e., it should be deleted). To do this, we define the following decision rule for full tracks:

$$H = \begin{cases} H_{\text{FP}} & S_t \leq \log(\gamma) \\ H_{\text{TP}} & \text{otherwise} \end{cases} \tag{4.11}$$

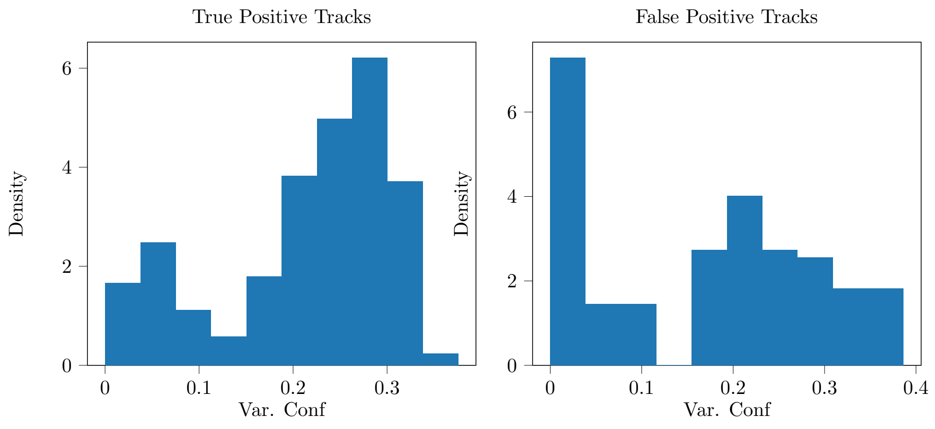
We determine γ by simulation techniques analogous to [27]. When computing the log probabilities, we apply an approximate bias correction by writing

$$\widehat{p}_{\text{new}} = \widehat{p} - \frac{1}{2n} \cdot \frac{1-\widehat{p}}{\widehat{p}} \tag{4.12}$$

where $\widehat{p}$ is the original estimated probability for a given bin of (μ_c, σ_c) and n is the number of samples in that bin. We determine this bias correction through a Taylor expansion of $\log(\cdot)$.



(a) Distributions of μ_c



(b) Distribution of σ_c

The uncertainty-aware decision system maintains key similarities with the base decision system. A key difference is that we can directly estimate a track's probability of existence, using its computed S_t. Hence, if we claim that p is the probability of a track existing, then $p = \frac{1}{1+\mathrm{e}^{-S_t}}$, where $S_t = \log\left(\frac{p}{1-p}\right)$. To find the total probability of incursion, we can assume that tracks are independent of each other, which means

$$\mathbb{P}\left(\text{incursion}\right) = 1 - \prod_{i=1}^{k}\left(1 - \frac{1}{1+\mathrm{e}^{-S_t^i}}\right) \tag{4.13}$$

where n is the number of tracks incurring on the runway and S_t^i is the value of S_t for the ith track. Using this, we can choose a threshold of an optimal probability cutoff value, above which we declare an intrusion and below which we allow the plane to land.

4.3 Results

We demonstrate the performance of our tracker and the baseline tracker on standard object tracking benchmarks and object detection benchmarks. We use Boeing's proprietary dataset which contains 52 scenes of landing sequences in various weather conditions with a diverse collection of ground vehicle behavior. The data we use is similar to the training data distribution of our neural network, so we would hope that the results are similar between the baseline and uncertainty-enabled tracker. For our tracker, we use a rolling window of 10 timesteps to compute the means and standard deviations, since this provides a good trade-off of tracking performance and real-time speed. We also choose $\beta = 0.001$ and $\alpha = 0.1$, since we are disproportionately more cautious of false negatives (a potentially fatal accident) than false positives (an inconvenience for the pilot). For the metrics shown here, we choose 22 random scenes to fit our probability distributions for the likelihood ratio test. The marginal probability distributions for rolling average confidence and rolling standard deviation in confidence are shown in Figure 4.2a and Figure 4.2b respectively. We notice that false positive tracks tend to have an average confidence much closer to zero, as expected. Furthermore, the distribution for false positive tracks tends to concentrate around zero, with a much lower variance than true positive tracks. This informs us that the network seems to be tuned to output more false negatives than false positives, since it seems more "certain" when there is no detection. However, this works against the specification that we set – requiring the system to be 100 times more cautious about false negatives than false positives. As such, we would hope to see a decrease in the number of false negatives in the uncertainty-enabled tracker compared to the baseline, likely at the expense of some additional false positives.

Our results on object tracking benchmarks [6, 45] are shown in Table 4.1. Specifically, we compute the mean multiple object tracking accuracy (MOTA), mean multiple object tracking precision (MOTP), total number of false positives, and total number of false negatives. When computing the number of false positives and false negatives, we consider tracks and objects on the "Runway Safety Area", which includes the landing strip and some portions of auxiliary taxiways. Using GPS data, Boeing's proprietary software allows us to determine the location of the Runway Safety Area in pixel coordinates.

We notice that, in both out-of-sample and in-sample data, the uncertainty-aware tracker tends to generate more false positives and fewer false negatives. This intuitively aligns with the desired safety specifications we set in our α and β parameters – we are 100 times more cautious of false negatives than false positives. Furthermore, on in-sample data, the uncertainty-aware method achieves marginally higher object tracking accuracy, and in both datasets, achieves higher object tracking precision. This is likely due to the incorporation of lower confidence detections in the uncertainty-aware method compared to the baseline method. Since detections for an object need not always appear greater than c', the tracks in the baseline tracker will not receive as many updates as the tracks in the uncertainty-aware counterpart. This will cause tracking error, since the state estimation techniques are often imprecise, and the estimation error will be further magnified by any errors in coordinate transformations between frames. Since the plane will be moving during the tracking process, the locations of tracks will change in pixel space, and, as such, each track's location needs to be adjusted using coordinate transforms. The combination of errors from these two processes means that frequent updates are almost always necessary to ensure that the tracks are reliably following their respective objects.

We also notice that there is a high number of false negatives compared to false positives. This is due to the neural network not being able to see vehicles on the runway until it approaches much closer. Hence, while the plane is closing in on the runway and has not received any detections, no tracks will be created and we log many false negatives as a result. However, this rarely leads to an instance where the system advises the pilot to land despite there being a vehicle on the runway, since the neural network can more reliably detect these objects as the plane approaches within a mile of the runway. Empirically, we also notice that the uncertainty-aware tracker generated a lot of false positive tracks due to incorrect associations to detections. Since the uncertainty-aware tracker ingests detections with much lower thresholds, it is possible for existing tracks to associate to detections that are false positives. In the next section, we present some ideas to potentially address this issue.

Table 4.1: Object Tracking Benchmarks on Boeing's Proprietary Landing Dataset

Methods	MOTA	MOTP	Total False Positives	Total False Negatives
In-Sample Data				
Baseline Tracker	0.6104	0.3605	**77**	2968
Uncertainty-Aware Tracker	**0.6368**	**0.4111**	386	**2602**
Out-of-Sample Data				
Baseline Tracker	**0.8139**	0.354	**198**	2112
Uncertainty-Aware Tracker	0.8011	**0.3716**	843	**1769**

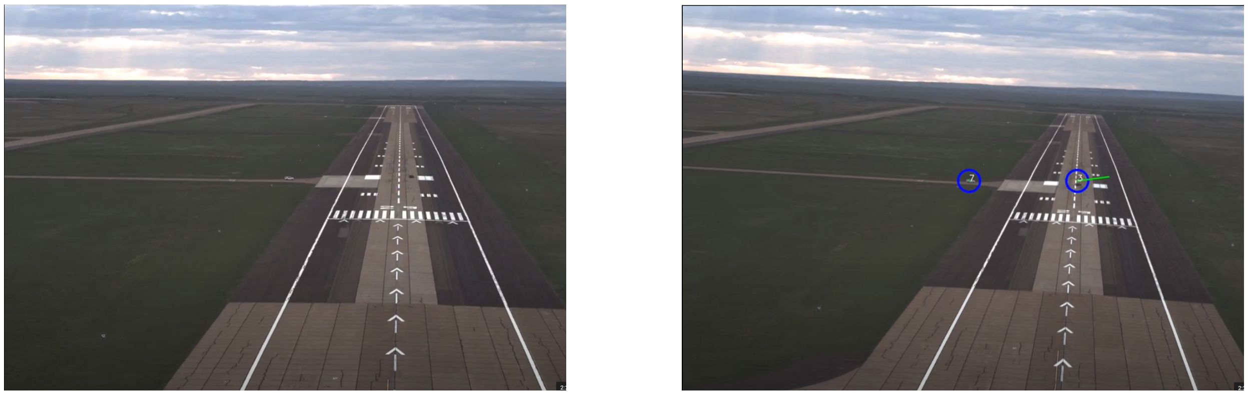

(a) Baseline Tracker (b) Uncertainty-Aware Tracker

Figure 4.2: Baseline Tracker (Figure 4.2a) vs. Uncertainty-Aware Tracker (Figure 4.2b). The blue circles represent full tracks, the numbers represent track IDs, and the green lines represent estimated velocity vectors of the tracks in pixel space.

4.3.1 Case Study: Synthetic Detector Failure

We highlight some of the benefits of our uncertainty-aware system by introducing scenarios with synthetic detector failure. We purposely disable the neural network's outputs when the plane flies below 200 feet. Since the baseline algorithm maintains a rule that any track that has not received detections for T timesteps will be deleted, we believe the baseline algorithm would perform worse than the uncertainty-aware algorithm. Specifically, if the uncertainty-aware algorithm has maintained tracks on a vehicle for an extended period of time leading up to the detector failure, it is less likely to immediately delete the track given the history of detections. This comes as a result of tracks with a lot of evidence having a very high likelihood ratio, which would need significant evidence (through a lack of detections) to delete a track. A frame of a scene is shown for the baseline tracker in Figure 4.2a and the same frame is shown for the uncertainty-aware tracker in Figure 4.2b. Notice that the uncertainty-aware tracker can more reliably keep tracks "alive", even in the presence of detector failure. This comparison of performance is indicative of our tracker being more robust due to incorporation of uncertainty.

4.4 Conclusion and Future Work

We present a tracker that uses neural network uncertainty to enable more robust tracking and reduction of false negatives and false positives. We implement a sequential hypothesis testing approach to improve a tracking algorithm's track management process, and we demonstrated the efficacy of our algorithm on Boeing's proprietary landing dataset. We demonstrate similar performance on object tracking benchmarks and superior performance on most object detection benchmarks. We also demonstrated a case study where the neural network experiences failures and our tracker outperforms the baseline tracker.

We believe that we can diminish false positives even further by incorporating uncertainty into the track association algorithm. Currently, only using uncertainty for the track creation and deletion process means that tracks may falsely associate themselves with false positives, causing errors in the track propagation algorithm. Hence, we plan on implementing a probabilistic association algorithm which takes inspiration from the analysis we performed on track creation and deletion. Fundamentally, we have translated uncertainty metrics to a track's existence probability. We can repeat a similar analysis by determining the probability of a detection being associated to a track – the higher the confidence of the detection and the closer it is to the track, the more likely it is to be associated. Currently, the association algorithm only considers distance between the track and detection, but we can make mild modifications to incorporate the detection's confidence and any other uncertainty metric we choose.

In the future, we would also like to apply online assurances in the form of confidence intervals. To avoid confidence intervals on S_t that grow unbounded over time, current methods would have to assume the correctness of S_{t-1}, which can be highly inaccurate. For example, if the neural network receives some in-distribution inputs followed by a sequence of out-of-distribution inputs, the confidence intervals we derived would not account for this distribution shift. We would like to approach the problem of certifying S_t unconditionally, while maintaining reasonably small confidence intervals that would not grow over time. Lastly, we may want to consider methods to improve the track association algorithm, ideally using nonparametric tools such as conformal prediction.

4.5 Acknowledgment

This material is based upon work supported by the DARPA Assured Autonomy Program. We would like to thank James Paunicka, Blake Edwards, Michael Schwiesow, Jose Medina, Zachary Tane, Thomas Cassidy, Dragos Margineantu, and the entire team at Boeing for all their help and contributions in demonstrating our method on Boeing's runway incursion dataset.

Chapter 5

Concluding Remarks and Future Directions

We presented several ways to incorporate neural network uncertainty in autonomous systems. We primarily focused on systems with a single learning-enabled component, such as a trajectory forecasting model, learned dynamics model, and a computer vision-based detector. We demonstrate the importance of providing probabilistic assurances and the efficacy of uncertainty quantification towards this goal. Finally, we discuss methods for incorporating uncertainty to produce a more robust control stack in the absence of ground truth information at runtime.

Most of the work we present provides robustness from the perspective of commonly-used statistical quantities, such as coverage, false positive rate, and false negative rate. However, when operating different kinds of autonomous systems, we may prefer a different metric to gauge the safety of our system and ensure that the entire system is safe. We are currently investigating methods to incorporate arbitrary notions of operating risk, and designing these risk measures for various, practical scenarios. For example, existing literature in conformal prediction allows arbitrary risk measures, but the usability of certain risk measures is not readily apparent for downstream tasks, such as planning and control. Furthermore, in methods like the ones covered in Chapter 4, any risk measure we leverage may need to be motivated from existing statistical methods, such as the sequential hypothesis test, to provide a valid guarantee.

We would also like to investigate processes to generate assurances for systems with multiple learning-enabled components. Certainly, we can use the methods that we already described here to apply assurances at the final learned component. However, this may cause our prediction intervals to be prohibitively large or guarantees to be far too loose. We would like to investigate methods that would modify the learned components to ingest auxiliary quantities from other learned components to adjust outputs in a way that would provide tighter guarantees. For example, we could modify learning-enabled components to ingest prediction intervals, or, more broadly, arbitrary measures of uncertainty from other learned modules to make a more informed prediction.

Finally, we would like to explore methods to measure the efficacy of certain uncertainty measures. It is not always the case that uncertainty measures are equally useful, and conversely, some uncertainty measures are highly central to the tightness of the bounds we empirically witness, as shown in Section 2.4.4. We would like to study statistical tools that can measure the importance of uncertainty measures without resorting entirely to empirical demonstrations. Ultimately, we would like to devise a hypothesis testing scheme that can determine the statistical significance of uncertainty metrics in providing a guarantee.

Bibliography

[1] Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint 2107.07511*, 2021.

[2] Andreas Auer, Martin Gauch, Daniel Klotz, and Sepp Hochreiter. Conformal prediction for time series with modern hopfield networks. *arXiv preprint arXiv:2303.12783*, 2023.

[3] Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Jaime F Fisac, Sampada Deglurkar, Anca D Dragan, and Claire J Tomlin. A scalable framework for real-time multi-robot, multi-human collision avoidance. In *2019 International Conference on Robotics and Automation*, pages 936–943. IEEE, 2019.

[4] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control*, pages 2242–2253. IEEE, 2017.

[5] Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, and Ryan J Tibshirani. Conformal prediction beyond exchangeability. *arXiv preprint 2202.13415*, 2022.

[6] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008.

[7] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[8] Steven D Blostein and Thomas S Huang. Detecting small, moving objects in image sequences using sequential hypothesis testing. *IEEE transactions on Signal Processing*, 39(7):1611–1629, 1991.

[9] Dimitrios Boursinos and Xenofon Koutsoukos. Assurance monitoring of learning-enabled cyber-physical systems using inductive conformal prediction based on distance learning. *AI EDAM*, 35(2):251–264, 2021.

[10] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. In *Computer Vision and Pattern Recognition*. IEEE, 2020.

[11] Bertrand Charpentier, Ransalu Senanayake, Mykel Kochenderfer, and Stephan Gunnemann. Disentangling epistemic and aleatoric uncertainty in reinforcement learning. *arXiv preprint 2206.01558*, 2022.

[12] Yuxiao Chen, Ugo Rosolia, Chuchu Fan, Aaron Ames, and Richard Murray. Reactive motion planning with probabilistic safety guarantees. In *Conference on Robot Learning*, pages 1958–1970. PMLR, 2021.

[13] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88, 2020.

[14] Alex Devonport, Forest Yang, Laurent El Ghaoui, and Murat Arcak. Data-driven reachability analysis with christoffel functions. In *2021 60th IEEE Conference on Decision and Control*, pages 5067–5072. IEEE, 2021.

[15] Anushri Dixit, Lars Lindemann, Skylar Wei, Matthew Cleaveland, George J Pappas, and Joel W Burdick. Adaptive conformal prediction for motion planning among dynamic agents. *arXiv preprint 2212.00278*, 2022.

[16] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R. Qi, Yin Zhou, Zoey Yang, Aur'elien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *International Conference on Computer Vision*, pages 9710–9719. IEEE, October 2021.

[17] Shai Feldman, Liran Ringel, Stephen Bates, and Yaniv Romano. Achieving risk control in online learning settings. *arXiv preprint 2205.09095*, 2023.

[18] Jaime F Fisac, Mo Chen, Claire J Tomlin, and S Shankar Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 11–20, 2015.

[19] David Fridovich-Keil, Andrea Bajcsy, Jaime F Fisac, Sylvia L Herbert, Steven Wang, Anca D Dragan, and Claire J Tomlin. Confidence-aware motion prediction for real-time collision avoidance. *The International Journal of Robotics Research*, 39(2-3):250–265, 2020.

[20] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[21] Isaac Gibbs and Emmanuel Candes. Adaptive conformal inference under distribution shift. *Advances in Neural Information Processing Systems*, 34:1660–1672, 2021.

[22] Junru Gu, Chen Sun, and Hang Zhao. Densetnt: End-to-end trajectory prediction from dense goal sets. In *International Conference on Computer Vision*, pages 15303–15312. IEEE, 2021.

[23] Bin Jia, Khanh D Pham, Erik Blasch, Dan Shen, and Genshe Chen. Consensus-based auction algorithm for distributed sensor management in space object tracking. In *2017 IEEE Aerospace Conference*, pages 1–8. IEEE, 2017.

[24] HM Dipu Kabir, Abbas Khosravi, Mohammad Anwar Hosen, and Saeid Nahavandi. Neural network-based uncertainty quantification: A survey of methodologies and applications. *IEEE access*, 6:36218–36234, 2018.

[25] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[26] Roger Koenker. *Quantile regression*, volume 38. Cambridge university press, 2005.

[27] Martin Kulldorff, Robert L Davis, Margarette Kolczak, Edwin Lewis, Tracy Lieu, and Richard Platt. A maximized sequential probability ratio test for drug and vaccine safety surveillance. *Sequential analysis*, 30(1):58–78, 2011.

[28] Shuo Li, Sangdon Park, Xiayan Ji, Insup Lee, and Osbert Bastani. Towards pac multi-object detection and tracking. *arXiv preprint arXiv:2204.07482*, 2022.

[29] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[30] Lars Lindemann, Matthew Cleaveland, Gihyun Shim, and George J Pappas. Safe planning in dynamic environments using conformal prediction. *IEEE Robotics and Automation Letters*, 2023.

[31] Kai Liu, Sheng Jin, Zhihang Fu, Ze Chen, Rongxin Jiang, and Jieping Ye. Uncertainty-aware unsupervised multi-object tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9996–10005, 2023.

[32] Rachel Luo, Shengjia Zhao, Jonathan Kuck, Boris Ivanovic, Silvio Savarese, Edward Schmerling, and Marco Pavone. Sample-efficient safety assurances using conformal prediction. In *Algorithmic Foundations of Robotics XV: Proceedings of the Fifteenth Workshop on the Algorithmic Foundations of Robotics*, pages 149–169. Springer, 2022.

[33] Seyed Mojtaba Marvasti-Zadeh, Javad Khaghani, Hossein Ghanei-Yakhdan, Shohreh Kasaei, and Li Cheng. Comet: Context-aware iou-guided network for small object tracking. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[34] Anish Muthali, Forrest Laine, and Claire Tomlin. Incorporating data uncertainty in object tracking algorithms. *arXiv preprint arXiv:2109.10521*, 2021.

[35] Anish Muthali, Haotian Shen, Sampada Deglurkar, Michael H Lim, Rebecca Roelofs, Aleksandra Faust, and Claire Tomlin. Multi-agent reachability calibration with conformal prediction. *2023 62nd IEEE Conference on Decision and Control*, 2023.

[36] Kensuke Nakamura and Somil Bansal. Online update of safety assurances using confidence-based predictions. *arXiv preprint 2210.01199*, 2022.

[37] Kiran Phalke and Ravindra Hegadi. Object tracking using mutiple hypothesis. In *2016 3rd International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 707–710. IEEE, 2016.

[38] Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2931–2940, 2019.

[39] Branko Ristic, Michael Beard, and Claudio Fantacci. An overview of particle methods for random finite set models. *CoRR*, abs/1602.03945, 2016. URL http://arxiv.org/abs/1602.03945.

[40] Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 683–700. Springer, 2020.

[41] Edward Schmerling. hj_reachability. https://github.com/StanfordASL/hj_reachability, 2021.

[42] Jeongho Shin, Sangjin Kim, Sangkyu Kang, Seong-Won Lee, Joonki Paik, Besma Abidi, and Mongi Abidi. Optical flow-based real-time object tracking using non-prior training active feature model. *Real-time imaging*, 11(3):204–218, 2005.

[43] Rohan Sinha, Edward Schmerling, and Marco Pavone. Closing the loop on runtime monitors with fallback-safe mpc. *2023 62nd IEEE Conference on Decision and Control*, 2023.

[44] Ingo Steinwart and Andreas Christmann. Estimating conditional quantiles with the help of the pinball loss. *arXiv preprint 1102.2101*, 2011.

[45] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, R Travis Rose, Martial Michel, and John Garofolo. The clear 2007 evaluation. In *International Evaluation Workshop on Rich Transcription*, pages 3–34. Springer, 2007.

[46] Sanbao Su, Songyang Han, Yiming Li, Zhili Zhang, Chen Feng, Caiwen Ding, and Fei Miao. Collaborative multi-object tracking with conformal uncertainty propagation. *arXiv preprint arXiv:2303.14346*, 2023.

[47] Cara Tabachnick. What to know about the recent close calls on airport runways. *CBS News*, 2023.

[48] Z Tian, C Shen, H Chen, and T He. Fcos: Fully convolutional one-stage object detection. arxiv 2019. *arXiv preprint arXiv:1904.01355*, 1904.

[49] Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. Conformal prediction under covariate shift. *Advances in Neural Information Processing Systems*, 32, 2019.

[50] Renukanandan Tumu, Lars Lindemann, Truong Nghiem, and Rahul Mangharam. Physics constrained motion prediction with uncertainty quantification. *arXiv preprint 2302.01060*, 2023.

[51] Dennis Ulmer, Chrysoula Zerva, and André FT Martins. Non-exchangeable conformal language generation with nearest neighbors. *arXiv preprint arXiv:2402.00707*, 2024.

[52] Balakrishnan Varadarajan, Ahmed Hefny, Avikalp Srivastava, Khaled S Refaat, Nigamaa Nayakanti, Andre Cornman, Kan Chen, Bertrand Douillard, Chi Pang Lam, Dragomir Anguelov, et al. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In *International Conference on Robotics and Automation*, pages 7814–7821. IEEE, 2022.

[53] Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. *arXiv preprint 2206.09889*, 2022.

[54] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*, volume 29. Springer, 2005.

[55] Niklas Wahlström and Emre Özkan. Extended target tracking using gaussian processes. *IEEE Transactions on Signal Processing*, 63(16):4165–4178, 2015.

[56] Abraham Wald and Jacob Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, pages 326–339, 1948.

[57] Evan Watson. Hillsboro airport leads in close calls, safety risks on runways in the northwest. *KGW8*, 2023.

[58] Chen Xu and Yao Xie. Conformal prediction interval for dynamic time-series. In *International Conference on Machine Learning*, pages 11559–11569. PMLR, 2021.

[59] Jiayu Yao, Weiwei Pan, Soumya Ghosh, and Finale Doshi-Velez. Quality of uncertainty quantification for bayesian neural network inference. *arXiv preprint 1906.09686*, 2019.

[60] Tao Zhao, Manoj Aggarwal, Rakesh Kumar, and Harpreet Sawhney. Real-time wide area multi-camera stereo tracking. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 976–983. IEEE, 2005.

Appendices

Appendix A

Safe, Learning-Enabled Planning for Autonomous Vehicles

A.1 Proof of Theorem 2.3.1

The theorem is restated for convenience below:

Theorem A.1.1 (Significance Level Correction)
Suppose that we wish to have a total miscoverage rate of γ, where total miscoverage rate is an upper bound on the probability that *any* human agent is miscovered:

$$\mathbb{P}_t\left(\bigcup_{i=1}^{N}\left\{\mathbf{x}_t^{(i)} \notin \mathcal{S}[t]^{(i)}\right\}\right) \leq \gamma. \tag{A.1}$$

We claim that the following α achieves an (asymptotic) total miscoverage rate of γ for N human agents:

$$\alpha = 1 - (1-\gamma)^{\frac{1}{N}}. \tag{A.2}$$

Proof To obtain this, we assume that each agent reacts to other agents entirely based on past observations. Mathematically, we define a filtration over the probability space of all agents' time-dependent actions $\mathcal{F}_0 \subset \mathcal{F}_{\Delta t} \subset \ldots \subset \mathcal{F}_{t-\Delta t} \subset \mathcal{F}_t$. We assume that any probability event corresponding to agent i's future behavior, denoted $A_{t'}^{(i)}$, is conditionally independent of any probability event corresponding to agent j's future behavior, denoted $A_{t'}^{(j)}$. Implicitly, we write $t' > t$. Specifically,

$$\mathbb{P}\left(A_{t'}^{(i)} A_{t'}^{(j)} \,\middle|\, \mathcal{F}_t\right) = \mathbb{P}\left(A_{t'}^{(i)} \,\middle|\, \mathcal{F}_t\right)\mathbb{P}\left(A_{t'}^{(j)} \,\middle|\, \mathcal{F}_t\right). \tag{A.3}$$

Intuitively, no agent makes decisions on unseen observations. Hence, we may rewrite Equation (A.1) as follows using the conditional independence and the corresponding lower bound on coverage rate for each agent:

$$\mathbb{P}\left(\bigcup_{i=1}^{N}\left\{\mathbf{x}_t^{(i)} \notin \mathcal{S}[t]^{(i)}\right\} \,\middle|\, \mathcal{F}_t\right) \tag{A.4}$$

$$= 1 - \mathbb{P}\left(\bigcap_{i=1}^{N}\left\{\mathbf{x}_t^{(i)} \in \mathcal{S}[t]^{(i)}\right\} \,\middle|\, \mathcal{F}_t\right) \tag{A.5}$$

$$= 1 - \prod_{i=1}^{N} \underbrace{\mathbb{P}\left(\mathbf{x}_t^{(i)} \in \mathcal{S}[t]^{(i)} \,\middle|\, \mathcal{F}_t\right)}_{\geq 1-\alpha-\mathcal{O}(1/T)} \tag{A.6}$$

$$\leq 1 - \prod_{i=1}^{N}(1 - \alpha - \mathcal{O}(1/T)) \tag{A.7}$$

$$= 1 - (1 - \alpha - \mathcal{O}(1/T))^N \leq \gamma. \tag{A.8}$$

Since we aim for an asymptotic bound, we take $T \to \infty$ which allows us to omit the $\mathcal{O}(1/T)$ term. To obtain the smallest value of α possible, we meet the loose inequality with equality, and find α such that $1 - (1 - \alpha)^N = \gamma$. Solving for α yields the value $\alpha = 1 - (1 - \gamma)^{\frac{1}{N}}$.

A.2 Metrics Computation

- Coverage rate: Letting previous notation prevail, we determine k^{th} prediction step coverage rate for any given scene by computing

$$\mathbf{1}\left\{\bigcap_{i=1}^{N}\left\{\mathbf{x}_{t+k\Delta t}^{(i)} \in \mathcal{S}[t + k\Delta t]^{(i)}\right\}\right\}$$

  per timestep. We average this quantity over timesteps to obtain average coverage rate for the specific scene.

- Conservatism: Define $\mathbf{x}_t^{(E)}$ to be the state of the ego vehicle in the ground truth data at time t, and define $\widehat{\mathbf{x}}_t^{(E)}$ to be the state of the ego vehicle controlled by the planner, at time t. For any given scene, we compute

$$\frac{\min\limits_{t}\min\limits_{i\in[N]}\left\|\widehat{\mathbf{x}}_t^{(E)} - \mathbf{x}_t^{(i)}\right\|_{xy}}{\min\limits_{t}\min\limits_{i\in[N]}\left\|\mathbf{x}_t^{(E)} - \mathbf{x}_t^{(i)}\right\|_{xy}},$$

  where $\|\cdot\|_{xy}$ computes the norm only along the x and y spatial coordinates of the given state vector. We average this quantity across all scenes. Intuitively, this is a measure of how close the planner is willing to get to other vehicles, compared to the ground truth ego vehicle movement.

- Progress: Let $\mathbf{x}_g$ denote the goal state of the ego vehicle, and let $\mathbf{x}_s$ denote its starting state. Next, let $\mathbf{x}_f$ denote the final state of the ego vehicle once the planner no longer provides any new plans. We define progress as

$$1 - \frac{\|\mathbf{x}_g - \mathbf{x}_f\|_{xy}}{\|\mathbf{x}_g - \mathbf{x}_s\|_{xy}}.$$

A.3 Un-Calibrated Confidence Set Metrics, Waymo Dataset

Below, we compare the coverage rate and set sizes between the un-calibrated version of our algorithm, using only quantile regression as a confidence-set-generating tool, and the calibrated version of our algorithm, leveraging conformal prediction.

Table A.1: Un-Calibrated Confidence Set Coverage Metrics

Prediction Step	Calibrated Set Coverage Rate	Un-Calibrated Set Coverage Rate
0.5	0.997 $\pm$0.002	0.968 $\pm$0.007
1.0	0.986 $\pm$0.006	0.887 $\pm$0.015
1.5	0.980 $\pm$0.008	0.839 $\pm$0.019
2.0	0.967 $\pm$0.011	0.817 $\pm$0.022
2.5	0.965 $\pm$0.011	0.818 $\pm$0.025
3.0	0.965 $\pm$0.013	0.800 $\pm$0.028

Table A.2: Un-Calibrated Confidence Set Sizes

Prediction Step	Calibrated Set Size	Un-Calibrated Set Size
0.5	61 $\pm$6	42 $\pm$4
1.0	246 $\pm$27	153 $\pm$16
1.5	655 $\pm$71	406 $\pm$46
2.0	1361 $\pm$143	815 $\pm$86
2.5	2422 $\pm$240	1434 $\pm$145
3.0	3885 $\pm$365	2300 $\pm$226

We primarily notice that conformal prediction enforces that set coverage remains above the desired 95% threshold, although this comes at the cost of providing larger sets. Additionally, conformal prediction aids in reducing the variance of coverage rates, especially for later-timestep predictions. This demonstrates the $\mathcal{O}(1/T)$ convergence guarantee on the deviation of the realized, empirical error rate from the desired error rate – with more data, the absolute deviation in conformal prediction's error rate from the desired error rate decreases, whereas this is not guaranteed for quantile regression alone.

A.3.1 Planner Visualization

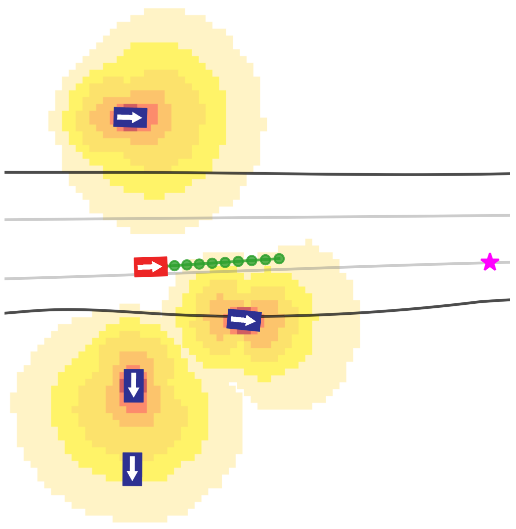


Figure A.1: Visualization of the planner. The autonomous ego vehicle is shown in red, and the human drivers are shown in blue. The plan is shown in green, representing the tracking of the original HJ-generated plan using iLQR.

A.4 Additional Algorithm Implementation Details

We opt for the version of Trajectron++ without the encoder for maps and the encoder for future ego-agent motion plans due to the lack of availability of these in the datasets. The model architecture and hyperparameters are kept the same as in [40].

We explored graph-based planning algorithms such as A* and Dijkstra but found them to be computationally intractable as a result of the high dimensionality of the Extended Dubins' car model and the presence of time-varying dynamic obstacles. Our reachability-based planner does not suffer from such issues.

In the case in which no feasible plan to the desired target state exists, for example due to the large sizes of the probabilistic reachable sets, our reachability-based planner produces a plan minimizing the distance between the final ego state and the desired target state.