Provably Safe Prediction for Autonomous Vehicles Using Uncertainty Estimations

Michael OBroin
mobroin@andrew.cmu.edu

Adrian Kager
akager@andrew.cmu.edu

13 December 2020

Abstract

With a projected annual contribution of \$800 billion "in economic and societal benefits" once fully adopted in the US alone, autonomous driving technologies have the potential to revolutionize transportation Carson et al. [2018]. Provable guarantees on safety are of the utmost importance when it comes to reaching full adoption of autonomous driving technologies. These provable guarantees are difficult to achieve, however, as these vehicles often rely on neural networks for prediction and control which are difficult to reason about due to their nature as function approximators. In our paper we specifically explore a method for proving a model that incorporates simplified pedestrian prediction using bounds on its uncertainty that hold with high probability empirically. We write a controller with 2D dynamics that takes into account the uncertainty of what it knows, in tandem with calculating how uncertain the predictions of a trained neural network are, using both Monte Carlo dropout and Ensemble methods. Finally, we wrote a simulator to demonstrate our controller and models in action.

1 Introduction

Verifying models of Cyber-Physical Systems (CPS) which make use of machine learning models is becoming increasingly important as real-world systems such as autonomous vehicles depend on them to operate. This challenge presents serious difficulties, because these neural networks lack the characteristics and explainability of traditional, non ML based, CPS.

Verification of models of autonomous vehicles is an especially important task due to the enormous potential economic and social benefits of fully autonomous vehicles and the safety critical nature of many autonomous vehicle tasks. Autonomous vehicles often make use of deep learning on a variety of tasks. One common domain is convolutional networks to do object detection, and another interesting example is trying to predict the behavior of other agents in the environment such as pedestrians. While a prediction of the future actions of a pedestrian would be invaluable, ML predictions have no formal guarantees, so these predictions cannot be relied on to be entirely accurate. One solution to this is to augment predictions with uncertainty estimates so the system can act more cautiously when there is high uncertainty. In this paper we explore predicting when a pedestrian will cross the street in a simplified model and prove the safety of the system assuming the true value lies within a range based on the uncertainty estimate. With the networks we trained and tested on the data we generated, we found that 99.97% of our predictions for unseen examples fell within four standard deviations of the ”true” label.

Specifically, we model the car and pedestrian interaction on a 2-D grid with the car traveling along the x axis and the pedestrian traveling parallel to the y axis. The pedestrian is predicted to cross the x axis at a cross walk at some time with a corresponding estimate on the prediction's uncertainty. Based on these quantities, the car chooses to either brake to a stop and let the pedestrian cross in front of it, or continue at its current velocity and travel past the crosswalk. We write this model in differential dynamic logic (dL), which allows us to logically express every part of this system. A formal definition of the syntax and semantics of this logic can be found in Platzer [2018].

Our project comes in three parts: a model of our system in differential dynamic logic, python code to generate representative data and learn a function with neural networks, and additional python code that simulates our model running. The bulk of our effort was spent in the modeling and data generation and networks components. The simulator combines the dL model and uses our ensemble neural networks to predict the crossing time of a pedestrian (with its uncertainty), and the follows the evolution of the system until the car either crashes into the pedestrian or passes them. The code for our project, along with our KeYmaera X models and proofs, is available in our github repository[1]. It is our hope that future students can build upon our code in future projects that explore this intersection of machine learning and verifying cyber-physical systems even further.

2 Related Work

We found two previous student projects from prior iterations of the course which shared important characteristics with our own. First is the RC car project from Ojha and Wang [2019]. This project investigated the implementation of a controller on a real-world RC car and dealt with the challenges of noisy sensor data and actuators. The authors' model had similar dynamics to our own system, modeling straight line motion of the car with an obstacle ahead, as well as considering the impact of

[1]`https://github.com/michael-obroin/424-project`

uncertainty in their model. Their project, however, implemented the controller on a actual physical car and used the uncertainty estimates from the actual sensors. Another prior project by Pardesi and Mahajan [2018] investigated a model of a car traveling along a multi-lane highway with a road that communicated the positions of obstacles to it . Their project considered the limit of what the car knew, but in a different fashion than our uncertainty estimation of the time. Instead, their car received location updates of the next obstacle after traveling a certain distance along the highway.

We make use of a method introduced by Segù et al. [2019] to estimate prediction uncertainty through the use of dropout in our network architecture. This method involves running multiple stochastic forward passes of a neural network that includes dropout layers and then treating the result of each forward pass as a Monte Carlo sample. A sample standard deviation can then be calculated and used as a measurement of model uncertainty. This paper goes much further and also derives a method to propagate data uncertainty through the network in order to estimate how much sensor variance effects predictions, but this is not relevant for us as in our simplified model we assume sensor data is exact.

The paper Probabilistically Safe Robot Planning with Confidence-Based Human Predictions by Fisac et al. [2018] applied similar ideas to a different domain - a real-world flying drone instead of our car model. They used more advanced Bayesian techniques to determine how correct the model has been previously and update it, as well as dealing with the problem of safe motion planning for flight.

3 Cyber-Physical Model

We modeled our environment as a two-dimensional grid, with our car and pedestrian represented as squares around a center coordinate. The car travels to the right (positive X) starting from (0,0), and the pedestrian travels up (positive Y) parallel to the y-axis starting from some point (x, -y) for x, y > 0. Our ML model predicts an estimated time at which the pedestrian will cross the road (ie the time it will be on the x axis) based on a variety of characteristics about the pedestrian explained below in the pedestrian data generation section. This is a complex enough model to capture the basics of a pedestrian and car interaction while being simple enough to model in our project time frame. See Figure 1 for a diagram of our model in its initial state.

3.1 Modeling Assumptions

We made a number of simplifying assumptions in order to reduce complexity and focus on the important aspects of the system that we wanted to model - namely, making the decisions under a bounded amount of uncertainty.

- We assumed straight line motion with constant and bounded acceleration and variable velocity in the positive x direction for the car, and straight line motion with constant velocity in the positive y direction for the pedestrian.
- The volumes of the car and pedestrian were represented as squares with side lengths of 2*carSize and 2*pedSize around their respective center coordinates.
- The initial position of the car was 0, 0, and the initial position of the pedestrian at some non-deterministic position with positive x and negative y that satisfies the preconditions.

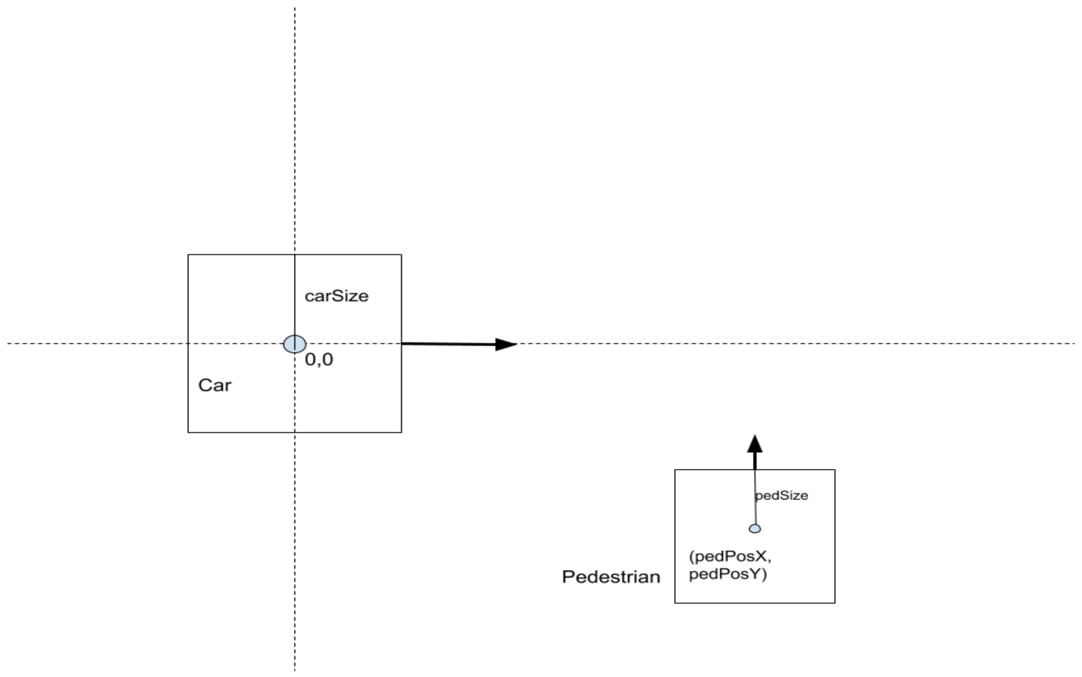

Figure 1: Diagram of our Cyber-Physical System

Under these assumptions we still have an interesting interaction between the car and pedestrian who enters the road at an unknown time, but the model is simplified enough to be manageable.

3.2 Model Explanation

The following section explains key aspects of the dL model. The full model is in Appendix I. The first key function is canStop(A, carVel) which tests whether or not the car can fully stop before reaching the crosswalk (including a buffer based on the size of the car and pedestrian). In contrast, canPassSafely(carVel) tests whether or not the car can safely pass by the crosswalk traveling at a constant velocity carVel before the earliest moment the pedestrian could enter the crosswalk based on the network prediction (CrossTime) and estimated uncertainty (EstimatedUncertainty). A key precondition of the model is canStop(-B, carVel) — canPassSafely(carVel) where B is the maximum braking acceleration. This means that there is at least one action the car can take and be safe after - either it can stop with full braking force or simply continue at its current speed.

The next key precondition check pedVelInBounds(pedPosY) checks that the pedestrian velocity (a constant named pedVal) is such that the pedestrian will reach the crosswalk in the assumed time interval [CrossTime - EstimatedUncertainty, Crosstime + EstimatedUncertainty]. This ensures that the pedestrian does not reach the crosswalk at an unexpected time which our controller would not be able to safely handle.

The final complex precondition validStartingPosition(carPosX, pedPosY) checks that the car and pedestrian do not start already intersecting and that the pedestrians starts strictly to the right of and below the car. All other preconditions simply bound the value of or indicate the sign of relevant variables and constants.

The controller assigns a random safe acceleration for the car in the range [-B, 0] where B is the magnitude of the maximum braking acceleration. These values are checked to be safe using canPassSafely(carVel) in the case of carAcc = 0 or canStop(A, carVel) in the case of carAcc < 0.

The dynamics of our system are fairly straightforward, either straight line motion with constant acceleration for the car, or constant velocity for the pedestrian. The car's changing position, velocity, and acceleration are carPosX, carVel, and carAcc respectively, and the pedestrian's position pedPosY changes at a rate pedVel. Note that carPosY and pedPosX do not change from their initial values as the car travels along the x axis and the pedestrian travels parallel to the y axis.

The postcondition didntCrash(carPosX, pedPosY) checks, well, that the car didn't end up crashing into the pedestrian. It does this by checking that the bounding squares around the car and pedestrian do not overlap. The [] around the controller means that the theorem holds if and only if the postcondition is true for all runs. This means that it is true for any value that the variables can take on, and for all lengths of time for which the differential equation dynamics can run. Therefore this postcondition is sufficient to enforce safety, as our model doesn't include any loops.

3.3 Model Proof Explanation

The proof breaks down to two main branches: one where the acceleration is set to zero and canPassSafely(carVel) holds, and another where the acceleration is negative and canStop(A, carVel) holds. In the first case, mcanPassSafely(carVel) indicates that at the earliest point the pedestrian is in contact with the road (including a buffer for the size of the car and pedestrian) the car is already past the crosswalk (also including buffers). Thus the postcondition follows immediately from solving the differential equations. In the second case we know at the constant acceleration A the car will never reach the crosswalk (again including buffers). Thus since the car never reaches the crosswalk it will never intercept with the pedestrian. Then this branch similarly closes immediately by solving the ODE's. The full proof tactic is provided in the GitHub repository (linked on page 1).

4 Code

Our code was split into three broad parts: the data generation, neural network training and testing, and a brief simulation, located in data_generation.py, networks.py, and sim.py respectively. We used seeded random number generators to generate our data. On top of the basic functionality we cover here, a lot of effort was spent writing the code in such a way that it is easy to experiment with different parameters, as well as saving and loading the data and models for reuse.

4.1 Pedestrian Data Generation

For our data generation, we considered a simplified model of a pedestrian with the following attributes: distance from the stop-walk, the cardinal direction they were facing, their current speed, whether or not they were wearing a red shirt. These were encoded as a float, one-hot vector, float, and a boolean respectively, resulting in a vector of length 7 for each sample generated. Our intention with this model was to capture enough of the characteristics of a pedestrian to make their behavior interesting without becoming too complex to learn. We then generated random samples of these characteristics within reasonable limits for a pedestrian, and used the function time_to_cross we wrote in data_generation.py to given the data a true label. We generated and saved 25000 samples to train on, 5000 to use as a validation set, and finally 1500 to test our models on and calculate statistics after training. Saving the same dataset allowed us to experiment with different architectures and reproduce our results more easily. For reproducibility, the seeds in our code are the ones used for all tests and training.

4.2 Neural Networks

Armed with the pedestrian data we had already generated, we investigated how to learn our "ground truth" function using a neural network. We wrote our code using the PyTorch deep learning library. The most important functionality is in the make_and_train and test_uncertainty functions, located in networks.py. In the first, we define and initialize our models and load our generated data, followed by training our networks one after another and then saving the models' parameters so that we can reuse them subsequently. After experimenting with several different architectures, including different numbers of layers and neurons in the hidden layers, we settled on the following architectures for our experiments. Here, Linear(in, out) indicates a fully connected linear layer with a bias term with in input size and out output size. ReLU indicates a Rectified Linear activation function, and Dropout(p) indicates a dropout layer that masks the output of a neuron from the previous layer to 0 with probability p. We trained and tested both the dropout and ensemble methods of uncertainty calculation. Our dropout model architecture was Linear(7, 50), ReLU, Linear(50, 1), Dropout(p=0.05), and we ran inference 40 times for the purposes of calculating the standard deviation of the predictions. Our ensemble architecture was similar, without the dropout layer and with an extra ReLU layer at the end: Linear(7,50), ReLU, Linear (50, 1), ReLU. We used standard NumPy library functions to calculate the mean and standard deviation of the models' predictions. Mean Squared Error was chosen as our loss function, and the Adam algorithm that is included in PyTorch as our optimizer.

5 Discussion

After we generated our datasets and trained our various neural networks, we wanted to know how reasonable our assumption was - that we could get a bound on the uncertainty in the predictions we intended to use in our model. Running our models on an unseen test dataset indicated that indeed, almost all test examples fell within a few standard deviations of our predicted value. However, this was only the case for the ensemble method. The Monte Carlo dropout method for estimating uncertainty did not yield reasonable results. For reasons we were unable to investigate fully, the predictions of the network would sometimes drop to 0, or else yield a standard deviation of 0 over all the simulations. We hypothesize that this could be due to a few neurons in the network having outsized importance in the model's output, perhaps corresponding to the most relevant important features of our input. This would also explain the better performance of the ensemble method, as the differences in the outputs of those models would be more due to initialization, instead of dropping weights that had learned potentially important information. See table 1 for a summary of the statistics of the models' predictions on the test data, and table 2 to see the distribution of answers compared to the calculated uncertainties.

Avg. Ensemble Std. Dev.	0.149
Ensemble MSE	0.030
Avg. Dropout Std. Dev.	2.442
Dropout MSE	0.253

Table 1: Statistics of running our models on a test set of size 1500

Std. Dev.	Ensemble	Dropout
1	93.20%	63.53%
2	99.53%	82.47%
3	99.93%	86.00%
4	99.93%	86.60%
5	100%	86.80%

Table 2: Percentage of predictions falling within x standard deviations of the true label

5.1 Deliverables

In our project proposal, we identified the following deliverables for our project:

- Experiment with more ML models and training schemes. The performance and training of the aforementioned Linear/ReLU combinations ended up being sufficient in practice.
- Create and reuse a significant number of training examples. This was accomplished, and we wrote a number of helper functions to facilitate reusability.
- Use both Monte Carlo Dropout and Ensemble methods to calculate uncertainty. This was also accomplished, though the dropout method did not end up performing well on our test dataset in practice.
- Adjust parameters of our dataset to be more realistic. We ended up slightly modifying our label function to remove the squaring factor, as well as adjusting the bounds on our data generation to reflect realistic values for distance to the crosswalk (in meters) and pedestrian speed (in meters per second).
- Incorporate "real world" constraints on our model assumptions. We ended up changing our KeYmaera X model to be a bit more realistic by modeling the car and pedestrian as squares instead of points, but we did not end up writing a version of our model that used specific constants.
- Weaken our assumptions on the uncertainty in the model. We were able to remove the assumption about the uncertainty value and it can now be arbitrarily large (even if that means our lower bound on the time to cross is negative).
- Change our model from point masses along a one-dimensional line to more realistic 2D motion. This was accomplished, and in our final model both the pedestrian and car are modeled as squares moving in a 2D plane.

In addition to these deliverables, we wrote a simple simulator that implemented our controller and used our ensemble network to make its predictions. This was done after we implemented the rest of the project, and would be a good target for further work.

5.2 Work Division

Equal work was performed by both team members.

5.3 Challenges Faced

Initially, our project targeted a slightly different domain, learning a verifiably safe controller with reinforcement learning. After spending a significant amount of time reading the literature and attempting to replicate the results of some papers with provided code, we ran into too many technical issues and had to go back and rework our idea. Following this brainstorming, we came up with our current project, deciding on a simplified model of a scenario that made its decision based on a prediction with some bounded uncertainty. We then further refined our project to concentrate on a car traveling down the street and predicting the time at which a pedestrian would cross into the street. Along the way, we faced issues stemming from generating our own data and trying to learn the function we defined. A significant challenge came from coming up with these ideas ourselves, and then writing the code to bring our ideas to life from scratch. However, we turned a corner after implementing more functions in data_generation.py so that we could re-use the same datasets to train multiple different networks.

6 Conclusion

Our work showed the viability of uncertainty estimation to verify the safety of prediction for autonomous vehicles under the assumption that the true value was within the uncertainty bounds. This is a fairly strong assumption and our model's dynamics and choices are also very simplified - the real world is much more unpredictable and noisy. Despite this, however, we hope that we have demonstrated that there is potential for very interesting future projects building off this work. In light of this hope, we made our codebase as modular and as easy to follow as possible to aid future development.

7 Appendix I: Full KeYmaera X Model

For the full proof, see the file in our github.

```
/* Exported from KeYmaera X v4.9.0 */

Theorem "Final Project Stopsign 2d with buffer"

Definitions

/*CONSTANTS*/
/*The time the ML model predicts the pedestrian will cross the road*/
    Real CrossTime;

/*The uncertainty in the CrossTime prediction (ie could cross anywhere in
 (CrossTime-EstimatedUncertainty,CrossTime-EstimatedUncertainty))*/
    Real EstimatedUncertainty;

/*maximum breaking force*/
    Real B;

/*Car and pedestrian only move along x, y axis respectivly
thus y and x positons are constant*/
    Real carPosY;
    Real pedPosX;

/*Car and pedestrian modeled as squares with
2*carSize = width of car = length of car and similar for pedestrian*/
    Real carSize;
    Real pedSize;

    Real pedVel;

/*FUNCTIONS*/
/*Car can safely stop before reaching a point where it could hit
the pedestrian if they enter the crosswalk*/
    Bool canStop(Real A, Real carVel) <->
    ((-1/2)*carVel^2*(1/A) < pedPosX - carSize - pedSize);

/*Car can safely travel past the crosswalk region before the first moment
the pedestrian could enter the crosswalk*/
    Bool canPassSafely(Real carVel) <->
    (carVel*(CrossTime - EstimatedUncertainty - ((pedSize + carSize)/pedVel))
    > pedPosX + carSize + pedSize);

/*Pedestrian velocity is such that it enters the crosswalk sometime in
the range [CrossTime - EstimatedUncertainty, CrossTime + EstimatedUncertainty]*/
```

```
    Bool pedVelInBounds(Real pedPosY) <->
    (pedVel >= -pedPosY/(CrossTime + EstimatedUncertainty) &
    (pedVel <= -pedPosY/(CrossTime - EstimatedUncertainty) |
    EstimatedUncertainty >= CrossTime));

/*Ensures positive time range*/
    Bool validEstimatedUncertainty() <->
    (EstimatedUncertainty >= 0);

/*Car and pedestrain not intersecting at start and pedestrain starts
fully lower and to the right of car*/
    Bool validStartingPosition(Real carPosX, Real pedPosY) <->
    (carPosX + carSize <= pedPosX - pedSize &
    carPosY - carSize >= pedPosY + pedSize);

/*Car and pedestrain have not crashed (ie do not intersect)*/
    Bool didntCrash(Real carPosX, Real pedPosY) <->
    (carPosX + carSize <= pedPosX - pedSize |
    carPosY + carSize <= pedPosY - pedSize |
    carPosX - carSize >= pedPosX + pedSize |
    carPosY - carSize >= pedPosY + pedSize);
End.

ProgramVariables
    Real carPosX;
    Real carVel;
    Real carAcc;

    Real pedPosY;

    Real t;
End.

Problem
    /*Preconditions*/
    (B > 0 & carVel > 0 & carPosX = 0 & carPosY = 0 &
    pedPosX > 0 & pedPosY < 0
    & t = 0 & carSize > 0 & pedSize > 0 &

    pedVelInBounds(pedPosY) &

    validEstimatedUncertainty() &

    (canPassSafely(carVel) | canStop(-B, carVel)) &
    validStartingPosition(carPosX, pedPosY))
    ->
    [
        /*Controller: assign a random accleration and check if it is in correct bounds
        and is safe*/
        {{carAcc := *; ?(canStop(carAcc, carVel) & carAcc < 0 & carAcc >= -B);}
        ++ {carAcc := 0; ?(canPassSafely(carVel));}}

        /*Dynamics*/
        {carPosX' = carVel, carVel' = carAcc,
        pedPosY' = pedVel, t' = 1 & carVel >= 0}
    ]
    /*Postcondition*/
    didntCrash(carPosX, pedPosY)

End.
End.
```

References

Charles Carson, Susan Helper, Erica Groshen, and John Paul Macduffie. America's workforce and the self driving future, Jun 2018. URL https://avworkforce.secureenergy.org/.

Jaime F. Fisac, Andrea Bajcsy, Sylvia L. Herbert, David Fridovich-Keil, Steven Wang, Claire J. Tomlin, and Anca D. Dragan. Probabilistically safe robot planning with confidence-based human predictions. *CoRR*, abs/1806.00109, 2018. URL http://arxiv.org/abs/1806.00109.

Shashank Ojha and Yufei Wang. Verified cruise control system on rc vehicle, Dec 2019. URL https://lfcps.org/course/lfcps19/projects/shashano_yufeiw2.pdf.

Ishan Pardesi and Dhruv Mahajan. Formal verification of v2i aided autonomous driving: A hybrid systems approach, Dec 2018. URL https://lfcps.org/course/lfcps18/projects/dmahajan-ipardesi.pdf.

André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018. ISBN 978-3-319-63587-3. doi: 10.1007/978-3-319-63588-0. URL http://www.springer.com/978-3-319-63587-3.

Mattia Segù, Antonio Loquercio, and Davide Scaramuzza. A general framework for uncertainty estimation in deep learning. *CoRR*, abs/1907.06890, 2019. URL http://arxiv.org/abs/1907.06890.