Perception Contracts for Safety of ML-Enabled Systems

ANGELLO ASTORGA*, University of Illinois at Urbana-Champaign, USA
CHIAO HSIEH*, University of Illinois at Urbana-Champaign, USA
P. MADHUSUDAN, University of Illinois at Urbana-Champaign, USA
SAYAN MITRA, University of Illinois at Urbana-Champaign, USA

We introduce a novel notion of perception contracts to reason about the safety of controllers that interact with an environment using neural perception. Perception contracts capture errors in ground-truth estimations that preserve invariants when systems act upon them. We develop a theory of perception contracts and design symbolic learning algorithms for synthesizing them from a finite set of images. We implement our algorithms and evaluate synthesized perception contracts for two realistic vision-based control systems, a lane tracking system for an electric vehicle and an agricultural robot that follows crop rows. Our evaluation shows that our approach is effective in synthesizing perception contracts and generalizes well when evaluated over test images obtained during runtime monitoring of the systems.

CCS Concepts: • **Theory of computation → Program specifications**; • **Software and its engineering → Software verification and validation**; • **Computer systems organization → Embedded and cyber-physical systems**; • **Computing methodologies** → *Machine learning*; *Computer vision.*

Additional Key Words and Phrases: perception contracts, safety, neural perception

ACM Reference Format:
Angello Astorga, Chiao Hsieh, P. Madhusudan, and Sayan Mitra. 2023. Perception Contracts for Safety of ML-Enabled Systems. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 299 (October 2023), 28 pages. https://doi.org/10.1145/3622875

1 INTRODUCTION

Deep learning has become the de facto method for solving perception tasks such as object detection, object tracking, and semantic segmentation. Deep Neural Network (DNN) functions are poised to be deployed en masse in safety-critical autonomous systems. Vision-based lane tracking in driver assist systems, vision-based landing protocols in urban aerial vehicles, and indoor robotic navigation with LIDAR are prime examples. To make progress towards reasoning about the correctness of the decision making programs in such systems, one has to gain some level of understanding of the correctness of the DNN-implemented perception. Fragilities of DNNs are well-known— they can fail to detect objects (e.g., pedestrians) and fail to recognize salient aspects of the environment (e.g., lanes from lane lines, and hence the position of ego vehicle in the environment). Since the precise set of raster images cameras can capture for each groundtruth setting is impossible to formally characterize, formal verification of such systems is hard.

*Equal contribution.

Authors' addresses: Angello Astorga, aastorg2@illinois.edu, University of Illinois at Urbana-Champaign, , USA; Chiao Hsieh, chsieh16@illinois.edu, University of Illinois at Urbana-Champaign, , USA; P. Madhusudan, madhu@illinois.edu, University of Illinois at Urbana-Champaign, , USA; Sayan Mitra, mitras@illinois.edu, University of Illinois at Urbana-Champaign, , USA.

2475-1421/2023/10-ART299
https://doi.org/10.1145/3622875

There have been, nevertheless, several novel attempts to characterize desirable characteristics of effective neural perception. The notion of *local robustness* (and *global robustness*) demand that the neural perception on training images (respectively, all images) evaluate to the same or similar perception when images are perturbed mildly with respect to a norm, such as the l_p norm. In recent years, several advances in automatic verification for local robustness for neural network have emerged [Bonaert et al. 2021; Gehr et al. 2018; Singh et al. 2019; Tjeng et al. 2019], including tool competitions [Bak et al. 2021b], and some approaches for global robustness have also emerged [Katz et al. 2017; Leino et al. 2021]. Robustness guarantees, to some extent, rule out adversarial examples to neural networks. Another approach that has emerged is that of using *image generators*— given an image generator that can generate images from groundtruth configurations of objects and probability distributions, we can test neural perception based controllers (such as autonomous vehicle controllers) against them, and use techniques such as fuzz testing to find scenarios where they fail [Dreossi et al. 2019; Fremont et al. 2022; Katz et al. 2021].

We posit that probing the perceptual role of DNNs in autonomous systems could unlock new opportunities for systematic analysis for safety of decision modules (called *reactive modules* throughout the rest of the paper). Neural perception pipelines are complex, but at the end of the day, they give *state estimates*. They provide estimates of certain *environmental variables* that are important for the reactive module. In the lane-keeping example, the vision pipeline provides an estimate of the relative position of the car with respect to the lanes and the heading angle. Thus, capturing how far the estimated state output from neural perception deviates from the true state (groundtruth state) serves as a natural specification or *contract* for the neural perception.

The primary contribution of this paper is the development of a technique based on *perception contracts*. Perception contracts are formal contracts for the perception module that describe the errors that the perception module can make regarding estimation of groundtruth while keeping the global system safe. Perception contracts hence negotiate the agreement between learned components for perception, and the programmed components of the system and the environment. This splits the analysis of the system to its modular components. In particular, in order to prove that a programmed system (which we call a reactive module in this paper) with a learned component NP for neural perception is safe under an environment, we need to ensure (a) that the ML component L for perception meets the perception contract, and (b) that assuming the perception contract is met by the perception module, the system stays safe under its environment. The second task can be performed using traditional techniques in formal verification of cyberphysical systems using automated logic engines. The first task cannot be formally proven as there is no formal specification of which images correspond to which groundtruth. However, we can use extensive testing techniques obtained from images gathered in the field as well as synthetically generated images that test for borderline/rare events to check if the perception module meets its contract.

The salient feature of perception contracts is that it is closely tied and dependent on the rest of the system and environment. The key idea is to measure neural perception errors, that is, the deviation of neural perception's estimation of groundtruth from the actual groundtruth. This deviation is not in absolute terms but *in terms of its effect on preserving invariants that prove the safety of the system*. More precisely, we deem a groundtruth estimate by neural perception as acceptable if the system's *action* based on this estimate, though it may differ from groundtruth, preserves the invariant designed for the system. A perception contract is hence a contract on groundtruth and groundtruth estimates that ensures that the system maintains its invariant.

The first contribution of this paper is the theory of perception contracts. We propose building safe systems that interact with neural perception using two distinct system-environment couplings that are *tied together* using perception contracts. First, we have the reactive module interacting with a *model of the environment* using *perfect perception* of groundtruth that is proven to be safe. Second,

we consider the same reactive module working with the real environment (which cannot even be modeled, of course), and where sensors compute an impression of the environment state (say camera images). A neural perception module processes this impression to estimates groundtruth, which is then used by the reactive module.

A perception contract relates the above two instantiations of the reactive module —one working with the environment model under perfect perception, and the other working with the real environment using sensors and neural perception (see Figure 2). The key property of a perception contract is that it *maintains* the invariant *Inv* that was used to prove the first system safe. More precisely, a perception contract is a symbolic contract expressed in logic that captures a set of pairs (gt, gte) of groundtruth and groundtruth estimates such that in any configuration satisfying the invariant *Inv*, the system acting on the estimate gte nevertheless preserves the invariant (even though the invariant itself typically will refer to the actual groundtruth gt).

We develop the theory of perception contracts, developing a framework and formally proving that the system working with the real environment through sensors and neural perception is guaranteed to be safe as long as the environment model can simulate the environment and the real environment with sensors meets the perception contract.

Since machine learned components are not designed by humans, their perception contracts, which specify the errors they make in various regions while maintaining the invariant, are complex to determine. Errors in perception that maintain an invariant also are dependent on the state of the system. Consider an invariant proving that a vehicle stays within the lane boundaries in a lane tracking system. This invariant naturally allows for more error in perception when the vehicle is in the middle of the lane and its steering angle is aligned with the road, while allowing less error when it is at the edge of the lane or its steering angle is misaligned with the road.

This leads us to the second main contribution of this paper: formulating and solving the *synthesis problem for perception contracts*. We are given a reactive module working with an environment model that has been proved to be safe using an invariant *Inv*, and we are given a neural perception module. We are also given a finite set of impressions of the sensors (like a set of images) along with the corresponding groundtruth values. We assume that the groundtruth estimates computed by neural perception on these impressions preserve the invariant (in practice, we can in fact verify this). The problem then is to find a perception contract *PC* in a given logic $\mathcal{L}$ (*PC* preserves the invariant of the system) that includes all the groundtruth-estimate pairs computed by the neural perception module. In other words, we want a contract in $\mathcal{L}$ that includes the groundtruth and estimation pairs exhibited by the neural perception on the given images, and maintains the invariant.

We develop a technique for synthesizing symbolic perception contracts using counterexample guided synthesis [Alur et al. 2015]. A decision-tree learner synthesizes quantifier-free logic formulae (allowing Boolean combinations) working with a verification engine that checks whether proposed contracts indeed preserve the invariant, and returns counterexamples otherwise.

The synthesis of perception contracts in specific regions of the groundtruth configuration space where the reactive module and the real environment seem to be working safely gives a conjecture of a formal specification of neural perception as to why the system stays safe even with neural perception. The adherence of the perception module to this contract cannot be formally proven, of course, as the real environment and sensors cannot be modeled mathematically effectively. However, extensive runtime monitoring in specialized settings where groundtruth can be determined (say environments with special sensors to detect position and heading angle of a vehicle) and checking such images against the contract gives greater confidence in the safety of the system. The perception contract also gives a formal specification for downstream attack techniques— for example, techniques that may try to generate an image using an image generator on which the neural perception violates the perception contract.

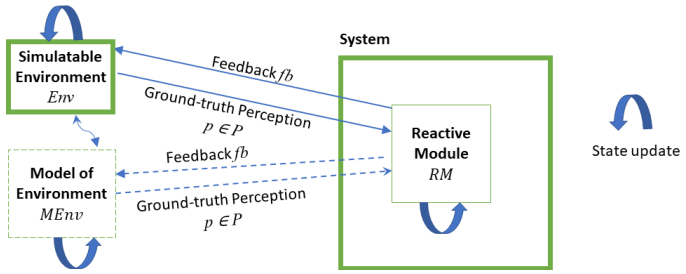


Fig. 1. Autonomous system interacting with environment/model of environment with perfect perception

One particular utility of perception contracts is in runtime monitoring of a system in an environment where groundtruth can be assessed. Rather than just monitoring the system for safety or even the invariant, we can monitor the perception module against the perception contract. Since a perception contract is a *local specification for the perception module*, monitoring adherence to it is stronger as the contract ensures safety of the system even for other valuations of variables in the system. For instance, consider an autonomous vehicle monitored when driving on a dry road; monitoring against a perception contract will check for the safety of the vehicle in other states as well, like at different speeds and wetness conditions of the road. We articulate this subtle issue in the paper, aided with examples in the evaluation section (Section 3.2 IV and Section 6.4).

The final contribution of this paper is an implementation and evaluation of our techniques for synthesizing perception contracts using two autonomous systems: a GEM vehicle for lane-tracking and an agricultural robot that navigates in aisles between crop lines, both using DNN-based perception (LaneNet [Neven et al. 2018] and CropFollow [Sivakumar et al. 2021]). Our evaluations show that our approach is extremely effective in synthesizing perception contracts. We also evaluate how well these synthesized contracts generalize by evaluating them against test sets obtained using runtime monitoring of these vehicles in closed-loop, and show that the contracts have high precision in capturing safe perception.

2 PRELIMINARIES: SAFETY OF SYSTEMS INTERACTING WITH ENVIRONMENTS USING PERFECT PERCEPTION

In this paper, we advocate the design approach that starts with designing a system that satisfies safety properties that interact with environments using *perfect perception*, before analyzing the system with sensors and neural-network based perception. In this setting, we assume that the system *perceives* aspects of the environment (called *percepts*) precisely— we do not yet model sensors, the output of sensors, and perception algorithms based on machine-learning and other computation (these are introduced in the next section).

We hence start by first modeling reactive modules interacting with environments with perfect perception. In such cyberphysical systems, traditionally, we consider the system interacting with a *model of the environment*, and prove that the system and model of the environment with perfect perception satisfy certain desirable safety properties. Such a proof ensures that the system is safe with respect to any environment that can be simulated by the model of the environment. We formalize these notions in this section (see Figure 1). The notations that we use in this section and the next section are depicted in Table 1.

Table 1. Notations used in Section 2 and Section 3

Notation for perfect perception	Extra notation for neural perception
Env: Environment	$Sensor$: Sensor reporting impression from environment state
$MEnv$: Model of Environment	NP: Neural perception module
RM: Reactive module (controller)	Imp: Impressions (output of sensor)
P: Percept variables	$Preserve_{Inv}$: Set of groundtruth estimates that preserve the invariant Inv at a configuration
FB: Feedback variables	PC: Perception contract
L: Latent variables of environment	
LM: Latent variables of environment model	
$Init_{Env}$: Initial states of environment	
$Init_{MEnv}$: Initial states of model of environment	
T_{Env}: Transition relation of environment	
T_{MEnv}: Transition relation of model of environment	
$Safe$: Safety property	
Inv: Invariant of system and model of environment interacting with perfect perception	

2.1 Systems Interacting with an Environment Using Perfect Perception

We formalize systems interacting with an environment (see Figure 1, in particular the solid path showing an environment model interacting with the system (called *reactive module* directly, without sensors and machine-learned perception components).

We have two modules— the environment Env and the Reactive Module RM modeling the system. In our driving example, RM is the control software of the car, and Env includes the car's hardware platform, the road, as well as the surrounding environment that influences the behavior of the car. Let us fix a set of *percept variables* P that captures some attributes of reality in the environment which are used by RM. In an autonomous driving setting, for example, this set of percepts would be variables that give the position of the car in some fixed coordinate system, e.g., the ego vehicle's coordinates, the lay of the road, the position and speed of pedestrians and vehicles nearby, etc.

Let us also fix a set of control or *feedback* FB variables that captures the feedback the RM gives to effect changes to the environment. Again, in the autonomous vehicle example, this feedback can be variables for control of brakes, acceleration, and steering angle of the vehicle.

The system and environment communicate with each other in *discrete time steps* or rounds. In each round the environment updates its state based on the current feedback. The percepts are fed to the system which in turn computes both an update to its state and feedback to the environment.

In the autonomous vehicle example, note that vehicle dynamics are part of the environment, and the feedback from the system is used to effect only certain variables, such as an update to the steering angle.

We model both the system and the environment using a set of variables over arbitrary domains, with state-space being the valuations of these variables, a set of initial states, and a transition relation describing (potentially nondeterministic) changes to these variables. For any set of variables X, let $Val(X)$ denote the set of all possible valuations of X (over the respective domains). Also, for any set of variables X, let X' denote a fresh set of primed variables corresponding to variables in X, i.e., $X' = \{x' \mid x \in X\}$.

Let us fix an environment Env with a set of variables $L \cup P$ that consist of a set of latent variables L and the percept variables P. Let us assume an initial state predicate $Init_{Env}(L, P)$, giving a set of initial valuations for latent and percept variables. And a transition relation $T_{Env}(L, P, FB, L', P')$.

Let $Q_{Env} = Val(L \cup P)$ denote the set of states of the environment and we will denote particular states as a pair (l, p). A transition $((l, p), fb, (l', p'))$ denotes that the environment, when in state (l, p) and reading feedback fb can transition to (l', p'), and give the system the percept p'.

Let the reactive module RM have a state-space defined by a set of variables S and an initial state predicate $Init_{RM}(S, fb)$ giving possible initial states and initial feedback. Let $Q_{RM} = Val(S)$ denote the set of states of the reactive module. The transition relation is a relation $T_{RM}(S, P, S', FB)$. A transition of the form (s, p, s', fb) means that the system, when in state s, reading a percept p, can transition to state s', and give the feedback fb to the environment.

The global behavior of the system with the environment and with perfect perception is defined over configurations $C = Loc \times Val(P) \times Val(FB) \times Val(L) \times Q_{RM}$, where $Loc = \{env, sys\}$ models whether it is the turn of the environment or system to move:

- There is a transition from (env, p, fb, l, s) to (sys, p', fb, l', s) if $((l, p), fb, (l', p')) \in T_{Env}$
- There is a transition from (sys, p, fb, l, s) to (env, p, fb', l, s') if $(s, p, s', fb') \in T_{RM}$

The initial set of configurations are those of the form $(\{env\}, p_0, fb_0, l_0, s_0)$, where $Init_{Env}(l_0, p_0)$ holds and $Init_{RM}(s_0, fb_0)$ holds.

2.2 Safety under Perfect Perception

Real world environments are incredibly complex to model and reason with. The classical approach of proving safety of systems that interact with the environment proceeds in two steps: (a) develop a simpler *model* of the environment $MEnv$ to simulate all possible behaviors of the real world environment (and potentially more), and (b) prove the safety of the system with respect to this model of the environment. The completion of this proof shows that the system interacting with *any* environment that can be simulated by the model of the environment will be safe.

We formally outline the invariant-based technique for proving systems safe against (a model of the) environment.

A model of the environment $MEnv$ has precisely the same structure as environments as described above. It has a set of latent variables LM (latent variables of the model), initial state predicate $Init_{MEnv}(LM, P)$, and transition relation $T_{MEnv}(LM, P, FB, LM', P')$, as described above.

Typically, the set of latent variables and transitions of the model are much simpler than real world environments, and can be seen as *assumptions* made of the real world. In the automated vehicle context, for example, vehicle dynamics may model nondeterministic skidding of a vehicle up to some degree (without modeling the wetness of the road precisely), making formal reasoning much easier.

Safety Properties. A *safety* property is a predicate $Safe(P, S)$ over the set of percepts and state of the system. Note that the safety property is independent of the latent variables of the environment, and consequently, we can determine safety of systems with respect to different environments (e.g., the real world environment as well as a model of it).

Turning to the autonomous vehicle example, a safety property may demand that the vehicle remains well within the left and right flanks of the road. Another desirable property (related to stability) is to demand that in any two consecutive configurations c and c', if c is reasonably away from the center of the lane, then c' will be closer than c is to the center of the lane. Though this property involves two consecutive configurations, it can be modeled as a safety property. We can have the system storing in its state the vehicle's previous configuration (say using variables added to the set P) and having the safety property demand the appropriate relation using the previous configuration and the current configuration.

Verification of Safety against Environment Models. Let us fix a system and an environment model $MEnv$. The set of *reachable configurations* of the system and environment is defined as usual—

the least set that includes the initial configurations and is closed with respect to the transitions over configurations (as defined above). Let us denote this set as *Reach*. The system and environment are said to satisfy the safety property $Safe(P, S)$ if for every configuration $(loc, p, fb, lm, s) \in Reach$, $Safe(p, s)$ holds.

An *invariant* is a predicate over global configurations $Inv(Loc, P, FB, LM, Q_{Sys})$ that defines a *superset* of *Reach*. Such an invariant is said to prove the safety property $Safe(P, S)$ if for every configuration $(loc, p, fb, lm, s) \in Inv$, $Safe(p, s)$ holds.

It is in general hard to check whether a given predicate Inv is an invariant of the system. The typical approach to ensure predicates are invariants is to have a stronger notion of invariants that is easier to verify. One approach is to use *inductive invariants*. An inductive invariant is a predicate Inv that satisfies the following properties: it includes all the initial configurations, and for every configuration c satisfying Inv and every configuration c' that c can transition to, $Inv(c')$ also holds. It is easy to see (by induction on length of executions) that such a predicate is always an invariant.

In this paper, we will not focus on methods to synthesize or prove safety for systems that work with perfect perception, as these are well studied problems in cyberphysical system verification (see, for example [Alur 2015; Astrom and Murray 2008; Mitra 2021]). However, we assume some *invariant* Inv that establishes safety (or stability) of the system working with an environment model— i.e., we assume that as long as a system and environment model work together in such a way that the reachable state space stays within the invariant, the safety/stability property is guaranteed to hold.

We assume that for desired safety properties of the system and a model of the environment, interacting with perfect perception, there is an invariant $Inv(Loc, P, FB, LM, Q_{Sys})$ that establishes safety of the system and environment.

Safety under Environments Simulated by an Environment Model. Now let us turn to proving the safety of systems interacting with an environment using perfect perception using the fact that the system is safe under an environment model using perfect perception.

Consider an environment Env with variables $L \cup P$ interacting with a system. And assume that the system interacting with a model of the environment $MEnv$ (over variables $LM \cup P$) satisfies a safety condition $Safe(P, S)$, established using an invariant $Inv(Loc, P, FB, LM, Q_{Sys})$. Now under conditions that relate the environment and the model of the environment, namely a simulation relation, we can argue that the system working with the environment Env will continue to be safe.

Formally, we say that the model of the environment $MEnv$ simulates the environment Env if there is a *simulation relation* $\sim$ between the states of the environment and the states of the environment model, i.e., between $Val(LM \cup P)$ and $Val(L \cup P)$, such that the following hold (below, $l, l' \in Val(L)$, $lm, lm' \in Val(LM)$, and $p, p' \in Val(P)$)

- Whenever $(l, p) \sim (lm, p')$, $p = p'$.
- For every initial state of the environment, there is an initial state of the model of the environment that it is related to, i.e., for every l, p, if $Init_{Env}((l, p))$ holds, there is some lm such that $Init_{MEnv}((lm, p))$ holds and $(l, p) \sim (lm, p)$.
- Let $(l, p) \sim (lm, p)$ and let $fb \in \mathcal{FB}$ be a feedback, and let $T_{Env}(l, p, fb, l', p')$ hold. Then there is some lm' such that $T_{MEnv}(lm, p, fb, lm', p')$ holds and $(l', p') \sim (lm', p')$.

The first condition says that states of the environment and environment model must share the same perception valuations. The second demands that every initial state of the environment is related to some initial state of the environment model. And the third demands that from any pair of states that are similar, the environment model should be able to simulate every move of the environment, and reach similar states.

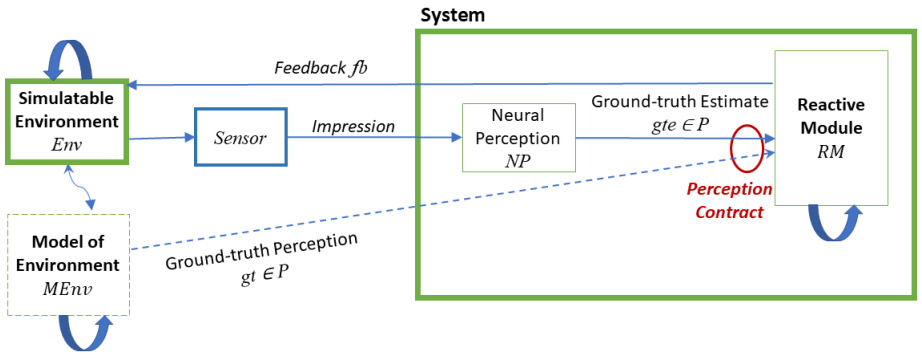


Fig. 2. Autonomous system interacting with environment/model of environment with neural perception; perception contracts.

Theorem 2.1. *Let a reactive module (system) RM with the environment model MEnv satisfy an invariant Inv, which in turn proves a property Safe, under perfect perception. Let Env be an environment module that MEnv simulates Env. Then the environment Env working with the system RM is guaranteed to preserve the property Safe.*

Proof. Let $\sim$ be a simulation relation between *Env* and *MEnv*. We can show by induction that on n that for every reachable configuration (loc, p, fb, l, s) reached by $Env \| Sys$ in n steps, there is a reachable configuration (loc, p, fb, lm, s) such that $(l, p) \sim (lm, p)$. The base case is easy, and the induction step involving a system move as the last move are trivial. For the induction step involving an environment move as the last move, the fact that the simulation relation guarantees a move of the environment model that simulates the environment ensures the property. □

3 SYSTEMS WITH NEURAL PERCEPTION AND PERCEPTION CONTRACTS

We now define systems that interact with environments with "neural perception"— perception is now not precise, but approximate perception is realized from impressions of observing the environment using machine-learned components and possibly other forms of computation.

3.1 Perception Contracts

Let us fix a system reactive module $RM = (Init_{RM}(S, FB), T_{RM}(S, P, S', FB))$ and an environment $Env = (Init_{Env}(L, P), T_{Env}(L, P, FB, L', P'))$ over a set of perception variables P and feedback variables FB.

We introduce two new components, a *sensor* and a *neural perception module* (see Figure 2). A sensor computes from environment states an *impression*. Impressions include many kinds of data produced by sensors— images taken by cameras, sound recordings of the environment, LIDAR readings, sensor readings of vehicles, etc. Formally, we fix a set of impression variables Imp, and sensors are modeled as a function $Sensor : Val(L \cup P) \rightarrow Val(Imp)$, i.e., functions from environment states to impressions.

A neural perception (NP) module is a module that processes impressions and outputs *perceptions*. Neural perception modules typically consist of machine-learned models (e.g., for vision, processing sound, etc.) as well as programmed components (e.g., geometric algorithms that complete line segments detected to lines, calculate middle of lanes from flanking lines, etc.). Formally, neural perception modules are modeled as a function $NP : Val(Imp) \rightarrow Val(P)$

The global behavior of the system with the environment and with neural perception is defined over the same set of configurations as in perfect perception, $C = Loc \times Val(P) \times Val(FB) \times Val(L) \times$

Q_{RM}. The transitions are the following two kinds, where the first kind for environment moves is precisely as earlier for perfect perception, while the second one utilizes the sensor and neural perception:

- There is a transition from (env, p, fb, l, s) to (sys, p', fb, l', s) if $((l, p), fb, (l', p')) \in T_{Env}$
- There is a transition from (sys, p, fb, l, s) to (env, p, fb', l, s') if $(s, NP(Sensor(l \cup p)), s', fb') \in T_{RM}$

Note that the system moves on the perceived perception it gets, delivered through the sensor and neural perception module.

Safety with respect to a predicate $Safe(P, S)$ is defined as usual.

Imperfect Perception but Action That Maintains Invariants. Let us fix a proposed invariant of the system $Inv(Loc, P, FB, LM, Q_{Sys})$.

The key idea of the contracts we define in this paper relies on the following idea of maintaining inductively the invariant with imperfect perception:

> In any configuration satisfying the invariant, the reactive module, though it acts on the *estimated* perception, gives feedback that results in a configuration satisfying the invariant.

The formalization of the above introduces certain subtleties. First, consider a predicate $Inv(Loc, P, FB, LM, Q_{Sys})$ that has been proved to be an inductive invariant when the system interacts with an environment model $MEnv$.

Let $gte \in Val(P)$ be a *ground-truth estimate*— this is any valuation of the perception variables, not necessarily the one given by the composition of sensors and neural perception. Let c be any configuration of the system working with the environment model satisfying Inv, where it is the system's turn to move, i.e., c is of the form (sys, p, fb, lm, s), and let c satisfy the invariant Inv.

Then we say the *ground-truth estimate gte preserves the invariant at c* if for any configuration $c' = (env, p, fb', l, s')$ where $(s, gte, s', fb') \in T_{RM}$, it is the case that c' satisfies Inv.

The above definition declares gte to preserve the invariant at the configuration c if the system module, reacting to the ground-truth estimate (rather than the groundtruth p) results in configurations c' that respect the invariant. Intuitively, gte, despite being not the precise groundtruth, is tolerable by the system as it keeps the invariant.

It is easy to see that the system will preserve the invariant as long as the neural perception gives, at every reachable configuration c, a groundtruth estimate that preserves the invariant at that configuration.

Let $Preserve_{Inv}(c)$ be the set of all ground-truth estimates at the configuration c that preserve the invariant. Our goal now is to find simple perception contracts that *imply* that groundtruth estimates preserve the invariant.

Perception Contracts. We are now ready to define the primary contribution of this paper—perception contracts.

Fix a system interacting with a model of the environment, and an invariant Inv for the system interacting with this model under perfect perception. A *perception contract* is a predicate $PC(P, P)$ with respect to the invariant Inv, expressed in a given logic $\mathcal{L}$, such that for every configuration $c = (sys, p, fb, lm, s)$ and $gte \in Val(P)$, if $PC(p, gte)$ holds, then gte preserves the invariant at configuration c, i.e., $gte \in Preserve_{Inv}(c)$.

Intuitively, a perception contract is a condition between groundtruths (p) and ground-truth estimates (gte) that ensures the estimate is safe with respect to maintaining the invariant.

We can now prove the main theorem regarding perception contracts. Consider an environment Env, with a sensor $Sensor$ and neural perception NP. We say that $(Sys, Env, Sensor)$ *satisfies a*

perception contract PC if in any reachable configuration, the neural perception working on the output of the sensor reading an environment state with groundtruth p computes a ground-truth estimate gte such that $PC(p, gte)$ holds.

We can now prove the main theorem associated with perception contracts.

Theorem 3.1. *Let a system interact with an environment Env through a sensor Sensor and a neural perception module NP. Let MEnv be a model that simulates the environment Env. Let Inv be an invariant of the system interacting with the model of the environment MEnv with perfect perception that ensures safety with respect to a predicate Safe. Let PC be a perception contract with respect to Inv and assume that the environment, sensor, and neural perception satisfy PC. Then the system working with the environment Env, under the sensor and neural perception, ensures safety with respect to Safe.*

Proof. (gist): The proof is by induction on the number of steps of taken by the environment, that for any reachable environment state q_{env} with perception groundtruth p, there is a similar state in the environment model ms with the same perception p that satisfies the invariant Inv. Notice that this proves that the system working with the environment through neural perception is safe.

The base case is trivial. The induction step is as follows. Given a state q_{env} with similar state in the environment model ms, both with the same perception valuation v, where ms with the system state satisfies the invariant Inv. Now, since the environment satisfies the perception contract, the groundtruth estimate that the neural perception computes from the signal generated from the environment state, must produce a feedback and system state update that satisfies the invariant Inv. Furthermore, the next state reached by the environment from q_{env} will be similar to a successor of ms (since the model simulates the environment). Consequently, the invariant will be preserved in the next state as well, completing the proof. □

3.2 Perception Contracts: Salient Features, Extensions, and the Synthesis Problem

There are several salient features of the notion of perception contracts that we want to highlight.

I. Perception Contracts Relate Different Systems. Perception contracts relate groundtruth perception values and groundtruth estimates given by the neural perception. Intuitively, we expect groundtruth estimates to be close to groundtruth. This error in perception is captured by the perception contract, symbolically. The error that is allowed is not *a priori* (say using some bounded norm), but rather it is allowed to be any error that the system can tolerate and maintain its invariant. Note that perception contracts are not like usual contracts which often relate only pre- and post-states of modules. Rather, a perception contract *ties together* two distinct systems—(a) the system that interacts with a model of the environment under perfect perception, and (b) the system that works with an environment (that can be simulated by the model) under sensors and neural perception. Note that the neural perception module takes as input the impression from the signal and generates a groundtruth estimate, and has no access to the groundtruth perception that the contract mentions. One can think of the groundtruth perception variables as a variant of *ghost variables* in verification literature that helps capture the specification for the module.

II. Maximal Set of Groundtruth Estimates That Preserve Invariants. Let Γ be the set of pairs (p, gte) such that for *every* configuration c satisfying the invariant Inv with groundtruth perception p, gte preserves the invariant at c. Then any perception contract PC defines a set that is a subset of Γ. However, this maximal set of pairs Γ, being defined using universal quantification over an unbounded set of configurations, and with intricate non-linear semantics of how the system and environment module behave, is a complex set that may not even be expressible in a reasonable symbolic logic. In this paper, we seek perception contracts that assure the invariant but also are expressible in a simple logic $\mathcal{L}$ (for example, using Boolean combinations of linear constraints over

the groundtruth and groundtruth estimates). See section 4.2 for a formal description of the logic used in our approach.

III. Perception Contract Synthesis Problem. The above discussion motivates the central algorithmic problem that we tackle in this paper— *synthesizing perception contracts*.

Let us fix a *finite* set of sensor outputs $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ (these are, for example, a set of training images obtained by cameras observing a real environment). For each $\sigma_i \in \Sigma$, assume that we know the groundtruth perception, gt_i. We can execute the neural perception module on each element of Σ_i to get a set of groundtruth estimates, gte_i. Let us assume that these estimates indeed preserve the invariant of the system (note that we can verify this). We hence have a set of pairs (gt_i, gte_i) of groundtruth and groundtruth estimates that are exhibited by the environment and neural perception, and that preserve the invariant. It is natural to expect to *include* these points in any perception contract for the network. Let us also fix a logic $\mathcal{L}$ to express perception contracts.

The **perception contract synthesis problem** is the following:

> Given a set of sensor outputs Σ associated with groundtruths, i.e., the set $\{(gt_i, gte_i)\}$, as above, find a perception contract expressed in $\mathcal{L}$ (which by definition maintains the invariant *Inv*) and that includes these pairs.

The perception contract synthesis problem asks to find a perception contract *PC* expressible in the logic $\mathcal{L}$ that includes the given pairs $\Sigma = \{(gt_i, gte_i)\}$ (and *provably* maintains the invariant *Inv*). We propose to solve the perception contract synthesis problem computationally.

Can Humans Write Perception Contracts? Given that a machine learning model is trained using tons of data (like ResNet), we *believe* that it would be hard for humans to predict the errors the ML perception module makes in each region in order to write a perception contract for it. Even for the given set of images (hundreds/thousands), we do not believe that humans can look at the errors made by ML perception to write a perception contract. The contracts synthesized by our tool also do not suggest that they can be written by humans easily (even though they are interpretable by humans). For example, a contract synthesized by our tool for a particular region is the decision tree: $\mathit{If}\ (y + \theta + d + \psi >= 0.163)\ \mathit{then}\ (y + \theta - \psi >= 0.859 \wedge y + \theta + d + \psi < 0.415)\ \mathit{else}\ \theta + \psi < 0.132$ where (y, θ) represents gt (groundtruth) variables—y is the distance to the center line and θ is the heading angle, and (d, ψ) represents gte (groundtruth estimate) variables—d is the estimated relative distance and ψ is the estimated relative heading angle.

Another possibility is for the user to write or mechanically derive the most general perception contract, independent of the ML module. Note that we need a perception contract expressed in a symbolic logic that maintains a complex invariant that the definition of perception contract demands. The most general perception contract may not even be expressible in the logic, let alone be simple enough for humans to construct.

We hence posit that we require automatic computational methods to solve the perception contract synthesis problem.

IV. Utility of Perception Contracts. Assume we have a set of outputs Σ of training data along with groundtruth, and we synthesize a perception contract *PC* from it that provably maintains the invariant (and hence keeps the system safe). Note that, of course, this does not mean the system is safe under the environment (that kind of safety is after all impossible to prove, given that we cannot even formalize the environment, and have only sampled some states). However, the contract *PC* not only gives a formal symbolic contract conjectured for the neural perception but also keeps the system satisfying its safety property whenever the contract is obeyed. More precisely, if all reachable states of the environment satisfy the perception contract *PC*, then we are guaranteed that the system will be safe. Though formal checking of whether the environment will satisfy the

contract *PC* is never possible, given that the training data adheres to the contract, and by more runtime monitoring to gather data and checking against the contract, gives confidence that the contract is satisfied, and the system is safe. Notice though that runtime monitoring needs to be in special environments where groundtruth can be measured or inferred (for example, cars in environments with enough position sensors that observe the actual groundtruth).

In fact, one utility of perception contracts is that they can be used for runtime monitoring. As opposed to naive runtime monitoring that monitors whether the system is within the invariant, runtime monitoring perception against perception contracts give a *stronger guarantee*. These checks ensure that the system is safe not only from the current state, but in *any other state*, where latent variables can be different. For example, in the setting of autonomous vehicles, a perception that verifies against a perception contract ensures that the perception is the same at *different* speeds and *different* road conditions as well. We refer the reader to the end of the Evaluation Section 6.4 for a more detailed discussion.

V. Relaxing Invariants. In the above development, we assumed that the inductive invariant *Inv* is any invariant that proves the safety of the system working with an environment model. However, extremely restrictive invariants may *disallow* perception contracts despite the system being safe. Intuitively, we want the invariant to be a bit more liberal than the strictest invariants that can prove the system under perfect perception correct, in order to allow errors in perception.

A concrete example occurs in our evaluation. Consider a lane-keeping vehicle where safety is proved using an invariant that demands that the vehicle always move towards the center line. More precisely, we record the position of the vehicle in the previous time step, and require that at the next time step, the vehicle is closer or at the center line. In operating with perfect perception, this is easy to achieve, as the system knows where the center line is, and can navigate the vehicle towards it. However, in any system with neural perception, this invariant will not hold. In particular, when the vehicle is very close to the center line, even an extremely accurate neural perception may detect it to the other side of the line, causing the system to violate the invariant. However, notice that for the safety property of being within the lane, this stringent invariant isn't required. We can relax this by asking for the invariant property to hold only if the vehicle is far enough away from the center of the lane. This leads to the existence of perception contracts that can prove the system safe, and we show in our evaluation that our mining algorithms are able to synthesize one.

4 LEARNING ARCHITECTURE FOR SYNTHESIZING PERCEPTION CONTRACTS

In this section we explain our general learning architecture (Figure 3) for synthesizing perception contracts. As defined in the previous section, the synthesis problem for perception contracts is to find a formula in a logic $\mathcal{L}$ that captures properties of groundtruth estimates (in relationship to groundtruth perception) that (a) include the groundtruth estimates of the neural perception module working on a finite set of impressions (images), and (b) maintain an invariant *Inv*.

Let us fix a system, environment model *MEnv*, sensor, neural perception (using notation developed in the previous section). Let us also fix a finite set of impressions (images) Σ. Let *GTE* be the finite set of groundtruth estimates computed by the neural perception on Σ.

4.1 Overview

To synthesize a perception contract, we use a *symbolic learning algorithm* that learns concepts in the given logic $\mathcal{L}$.

Our learning architecture (depicted in Figure 3) has the learner interact with a teacher, which provides counterexamples to adherence to the invariant.

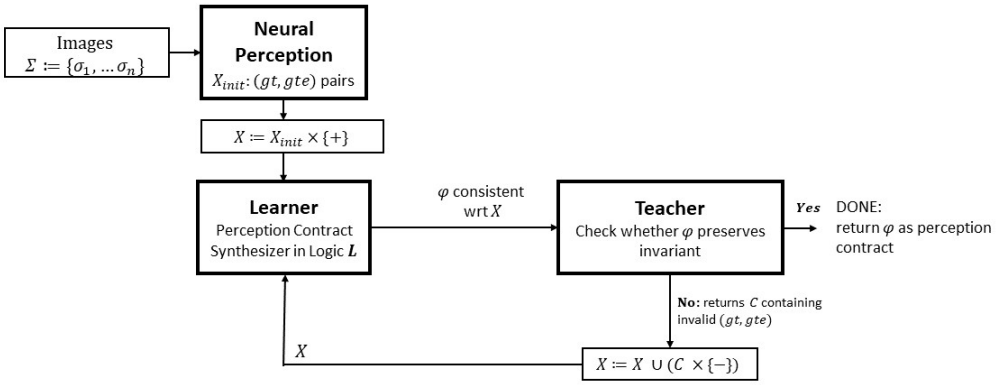


Fig. 3. Architecture for synthesizing Perception Contracts (PCs)

In each round, the learner takes a set X of (gt, gte) pairs (of groundtruth and groundtruth estimate), each pair labeled positive and negative. It then synthesizes a classifier in the logic $\mathcal{L}$ that is precisely classifies the samples in X, and forwards it to the teacher.

The teacher, receiving a proposed perception contract PC checks whether the pairs of groundtruth and groundtruth estimates defined by the contract maintain the invariant Inv. This requires checking whether for every pair (gt, gte) satisfied by PC, and for every configuration c with perception component gt that adheres to the invariant, gte preserves the invariant at c. We achieve this using a constraint solver (Gurobi in our evaluation). If the perception contract does not maintain the invariant, we extract a counterexample pair of the form (gt, gte), label it *negative*, add it to X, and recurse calling the learner.

The learner, in each round receiving a set of positive and negative sample X, computes a feature vector which has additional features depending on the logic, and uses decision tree learning to learn concepts expressed as Boolean combinations of these features. The decision-tree learning algorithm is an *exact* learning algorithm (classified the training set perfectly), and is an adaptation of Quinlan's ID3 algorithm [Quinlan 1986], modified slightly as described below to increase margins.

Note that when the teacher verifies a perception contract to maintain the invariant the algorithm can return this contract as it is a valid perception contract that includes X_{init}.

4.2 Realizing the Learning Architecture

We now describe how to realize the learner and teacher in the learning architecture for perception contracts.

Logics for Expressing Invariants. We assume that the logics for invariants are expressed as quantifier-free formulae over the domains of variables appearing in the environment model and the system, the feedback variables, and the perception variables. While the logics can be arbitrary, in order to apply decidable verification techniques such as Gurobi, we restrict these to use a class of nonlinear functions, trigonometric functions, etc. that lead to instances of constraint solving problems that can be decided by Gurobi.

Logics for Perception Contracts. In fixing a logic to realize our learning, one must strike a balance between expressivity and simplicity. The logic for perception contracts should be powerful enough to capture regions of groundtruth and groundtruth estimate pairs that preserve an invariant. Note that invariant expressions can be complex (expressed using nonlinear arithmetic and trigonometric

functions). We need the formulas in the logic describing the perception contracts to be simple in order for verification of invariant preservation to be decidable using tools such as Gurobi.

Let us fix the variables P and P' (two copies of percept variables). Let Π be a finite set of predicates over $P \cup P'$ and let Γ be a set of real-valued functions over $P \cup P'$. Let us introduce new variables $\{\pi_i\}$ and $\{\gamma_i\}$ that stand for the Boolean and real values corresponding to the valuation of predicates Π and Γ, respectively. The logics for learning are parameterized over Π and Γ, and are essentially quantifier-free Boolean combinations of predicates in Π and upper/lower bounds on the functions in Γ:

$$f ::- \texttt{ite}(bp, f, f) \mid bp \mid \texttt{true} \mid \texttt{false}$$
$$bp ::- \pi \mid \gamma < c \mid \gamma \leq c$$

where π ranges over π_i, and γ ranges over γ_i. The *ite* expression stands for *if-then-else* terms: $ite(b, f, f')$ evaluates to f if b is true and to f' otherwise.

In our evaluation, the predicates Γ are typically octagonal expressions ($+ - x + -y$) over the real-valued variables. Consequently, perception contracts are Boolean combinations of linear expressions over reals, and hence amenable to verification by tools like Gurobi.

Symbolic Learner. We can learn formulas over the logic for perception contracts from samples using decision-tree learning. For every positive/negative sample (gt, gte), we expand it to a larger feature vector where new features evaluate to every π_i and γ_i. The problem then reduces to learning a Boolean formula over predicates (π) and upper/lower bounds of numerical variables by constants ($\gamma < c, \gamma \leq c$), and we can employ a standard decision-tree learning algorithm for this purpose.

In our evaluation, we utilize the ID3 algorithm for decision tree learning, modified appropriately to find perfect classifiers [Garg et al. 2014; Mitchell 1997]. The ID3 algorithm builds the tree top down in one pass, choosing predicates to apply at each node. The best attribute at any node of the tree is chosen based on the information gain statistical measure based on entropy [Mitchell 1997]. While typical decision tree algorithms can stop building the tree when the leaf nodes are mostly pure, we need to continue building the tree till leaves are entirely pure (i.e., all samples that flow to any leaf must all be positive or all be negative).

Teacher. Given a proposed perception contract φ, the teachers need to check if all (gt, gte) pairs admitted by φ preserve the invariant.

Formally, we need to check the validity of the following formula:

$$(PC(P, P') \wedge Inv(sys, P, FB, LM, S) \wedge T_{RM}(S, P', S', FB')) \Rightarrow Inv(env, P, FB', LM, S') \quad (1)$$

The above formula captures the crucial property of when a perception contract preserves an invariant. It says that if $PC(gt, gte)$ holds and there is a configuration c satisfying the invariant with percept variables evaluating to gt, and the system reading the groundtruth estimate gte can move to a new state s' giving feedback fb', then the invariant must hold where the perception variables are evaluated according to the groundtruth gt.

The teacher needs to check the validity of the above formula, and if not valid, return a valuation of variables where the formula does not hold. Projecting this valuation to (P, P') gives the counterexample pair that is returned to the learner. In our evaluation, we implement this teacher using the constraint-solver Gurobi [Gurobi Optimization 2020].

5 CASE STUDIES

We will study how perception contracts can be used to analyze the safety of two realistic vision-based control system, namely, a Lane Tracking System (LTS) on a Polaris GEM E2 electric vehicle [Du et al. 2020] and an agricultural robot that follows crop rows [Sivakumar et al. 2021]. The controllers

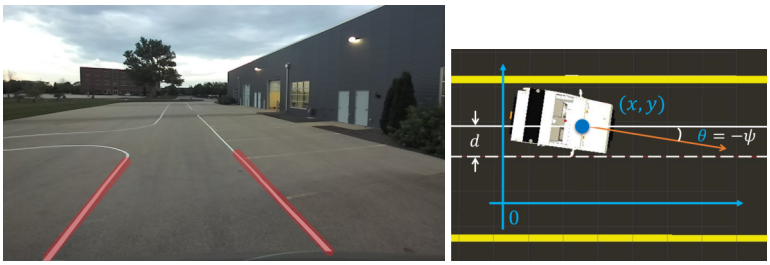


Fig. 4. Vision-based lane tracking system (LTS) on AV platform (*Left*). Latent variables (x, y, θ) for vehicle positions and percept variables (d, ψ) for the lane tracking system (*Right*).

(reactive modules) used in these systems do provably satisfy certain safety properties *when they work with perfect perception.* However, those same properties may be violated under some conditions when the controllers work with neural perception. The perception contracts constructed here capture the positive examples, and as we shall see, they can help discover the conditions under which the overall system can be proven to be safe.

5.1 Case Study: Lane Tracking System

For the experiments on the GEM vehicle in this paper, we use the high-fidelity Gazebo simulation of the vehicle model together with the actual controller code. The neural perception module uses LaneNet [Neven et al. 2018] for lane detection.[1] We use the well-studied Stanley controller [Hoffmann et al. 2007] for lane tracking as the reactive module.

Table 2. Constants in Lane Tracking System.

	Value	Description
W	4.0	Lane width (meter).
ΔT	0.05	Time discretization parameter (second).
v_f	2.5~3.0	Possible forward velocity (meter/second).
L	1.75	Length of the wheelbase (meter).
K	0.45	Proportional gain for Stanley controller.
δ_{max}	0.61	Maximum steering angle limit (radian).

Environment Model. Given a global reference coordinate system, we define latent variables for the environment model $LM = \{x, y, \theta\}$ where x and y represent the position of the vehicle in the 2D plane, and θ represents the orientation of the vehicle with respect to the x-axis (see Figure 4). The percept variables are $P = \{d, \psi\}$, where d is called the cross-track error and heading error ψ is the difference between the direction of the lane and the heading of the vehicle. When the lane center line is aligned with the x-axis of the global reference, the groundtruth cross-track error $d = -y$ and the heading error $\psi = -\theta$. Thus, the valuation $Val(LM \cup P) \subseteq \mathbb{R}^5$. Feedback variable $FB = \{\delta\}$ is the steering input to the vehicle, and the valuation $Val(FB) \subseteq \mathbb{R}$. The transition of the environment model is derived from a relatively simple textbook bicycle model [Hsieh et al. 2022b]:

$$((x, y, \theta), (d, \psi), \delta, (x', y', \theta'), (d', \psi')) \in T_{MEnv} \text{ iff}$$

[1]We use https://github.com/MaybeShewill-CV/lanenet-lane-detection, a popular open source implementation of LaneNet on GitHub.

$$
\begin{aligned}
x' &= x + \mathrm{v_f}\cos(\theta+\delta)\Delta\mathrm{T} \\
y' &= y + \mathrm{v_f}\sin(\theta+\delta)\Delta\mathrm{T} \\
\theta' &= \theta + \mathrm{v_f}\frac{\sin\delta}{\mathrm{L}}\Delta\mathrm{T} \\
(d',\psi') &= (-y',-\theta')
\end{aligned}
$$

where $\mathrm{v_f}$, $\Delta\mathrm{T}$, and L are constants as in Table 2.

Reactive Module. The Stanley controller g in [Hoffmann et al. 2007] is given as follows:

$$
\delta = g(d,\psi) = \begin{cases} \psi + \arctan\left(\frac{\mathrm{K}\cdot d}{\mathrm{v_f}}\right), & \text{if } \left|\psi + \arctan\left(\frac{\mathrm{K}\cdot d}{\mathrm{v_f}}\right)\right| < \delta_{\max} \\ \delta_{\max}, & \text{if } \psi + \arctan\left(\frac{\mathrm{K}\cdot d}{\mathrm{v_f}}\right) \geq \delta_{\max} \\ -\delta_{\max}, & \text{if } \psi + \arctan\left(\frac{\mathrm{K}\cdot d}{\mathrm{v_f}}\right) \leq -\delta_{\max} \end{cases}
$$

where $\mathrm{v_f}$, K, and $\delta_{\max}$ are constants in Table 2. The transition of the reactive module is defined as follows:

$$((d,\psi),\delta) \in T_{RM} \text{ iff } \delta = g(d,\psi)$$

Initial Configurations, Safety, and Invariant. Safety is to guarantee the vehicle never leaves lane boundary.

$$Safe((d,\psi)) = |d| \leq 0.5\mathrm{W}$$

where W is the constant value of lane width in Table 2.

Recall that configuration is defined by $C = Loc \times \mathcal{P} \times \mathcal{FB} \times \mathcal{LM} \times Q_{RM}$. Since the reactive module has no state, we can ignore Q_{RM}. We consider the following initial configurations:

$$
\begin{aligned}
Init &= \{(\mathrm{env},(d_0,\psi_0),\delta_0,(x_0,y_0,\theta_0)) \mid ((x_0,y_0,\theta_0),(d_0,\psi_0)) \in Init_{MEnv} \wedge \delta_0 \in Init_{RM}\} \\
Init_{MEnv} &= \{((x_0,y_0,\theta_0),(d_0,\psi_0)) \mid |d_0| \leq 0.3\mathrm{W} \wedge |\psi_0| \leq \frac{\pi}{12} \wedge (d_0,\psi_0) = (-y_0,-\theta_0)\} \\
Init_{RM} &= \{\delta_0 \mid |\delta_0| \leq \delta_{\max}\}
\end{aligned}
$$

To show the safety, we consider that the system should preserve the Lyapunov stability [Gibson et al. 1961] as the invariant and relax the Lyapunov stability based on input-to-state stability [Sontag 2008]. That is, we are given a Lyapunov function V, which takes the groundtruth values of d and ψ and outputs a non-negative real value representing the (generalized notion of) distance to a desired state. The invariant then is to require that, when the system is evolving, this distance is always nonincreasing so that the system stays safe since it always stays within a distance to the desired state. In addition, the nonincreasing requirement is relaxed if the distance has decreased to less than a small threshold parameter ρ. In this case study, the desired state is when the vehicle is aligned with the center line of the lane, i.e., $d = 0$ and $\psi = 0$; hence the Euclidean norm $V(d,\psi) := \sqrt{d^2+\psi^2}$ is given as the Lyapunov function over the domain $\mathcal{P}$. The relaxed invariant Inv that includes initial configurations $Init$ is as follows:

$$Inv = \{(loc,(d,\psi),\delta,(x,y,\theta)) \mid V(d',\psi') \leq \max(V(d,\psi),\rho)\} \tag{2}$$

where ρ is the parameter for relaxing the invariant. We will further evaluate over different ρ values in Section 6.

5.2 Case Study: Agricultural Robots

The second case study is a visual navigation system, named CropFollow, for under-canopy agricultural robots (AgBot) developed in [Sivakumar et al. 2021]. The system is responsible for avoiding collisions to the row boundaries when the vehicle traverses the space between two rows of crops. Similar to Lane Tracking System, the latent variables $LM = \{x, y, \theta\}$ consist of the 2D position

x and y and the heading θ. The sensor captures the image in front of the vehicle with a camera (Figure 5). Likewise, the percept variables $P = \{d, \psi\}$ are composed of the heading difference ψ and cross track distance d to an imaginary center line of two rows. The valuation of environment model states is thus $Val(LM \cup P) \subseteq \mathbb{R}^5$.

Fig. 5. Real and simulated camera images for corn row following for agricultural robots.

However, all components in CropFollow are very different from the Lane Tracking System in GEM. The neural perception module uses ResNet-18, as used in [Sivakumar et al. 2021], to detect the row boundaries. In contrast to the bicycle model, the vehicle dynamics is a kinematic differential model of a skid-steering mobile robot [Sivakumar et al. 2021] as follows:

$$((x, y, \theta), (d, \psi), \delta, (x', y', \theta'), (d', \psi')) \in T_{MEnv} \text{ iff}$$

$$\begin{aligned} x' &= x + \mathrm{v_f} \cos\theta \Delta\mathrm{T} \\ y' &= y + \mathrm{v_f} \sin\theta \Delta\mathrm{T} \\ \theta' &= \theta + \omega \Delta\mathrm{T} \\ (d', \psi') &= (-y', -\theta') \end{aligned}$$

where $\mathrm{v_f}$ and $\Delta\mathrm{T}$ are constants in Table 3.

Table 3. Constants in Corn Row Following System for Agricultural Robots

	Value	Description
W	0.76	Corn row width (meter).
$\Delta\mathrm{T}$	0.05	Time discretization parameter (second).
$\mathrm{v_f}$	0.2	Forward velocity (meter/second).
K	0.1	Proportional gain for the modified Stanley controller.
$\omega_{\max}$	0.5	Maximum steering angular velocity (radian/second).

Reactive Module. The modified Stanley controller uses the feedback variable $FB = \{\omega\}$ where ω is the angular velocity instead of the steering angle. Formally, the reactive module using the modified Stanley controller g is as follows:

$$\omega = g(d, \psi) = \begin{cases} \left(\psi + \arctan\left(\frac{\mathrm{K} \cdot d}{\mathrm{v_f}}\right)\right) / \Delta\mathrm{T}, & \text{if } \left|\psi + \arctan\left(\frac{\mathrm{K} \cdot d}{\mathrm{v_f}}\right)\right| < \omega_{\max}\Delta\mathrm{T} \\ \omega_{\max}, & \text{if } \psi + \arctan\left(\frac{\mathrm{K} \cdot d}{\mathrm{v_f}}\right) \geq \omega_{\max}\Delta\mathrm{T} \\ -\omega_{\max}, & \text{if } \psi + \arctan\left(\frac{\mathrm{K} \cdot d}{\mathrm{v_f}}\right) \leq -\omega_{\max}\Delta\mathrm{T} \end{cases}$$

where v_f, K, and ω_{max} are constants shown in Table 3. The transition of the reactive module is defined as follows:

$$((d, \psi), \omega) \in T_{RM} \text{ iff } \omega = g(d, \psi)$$

Initial Configurations, Safety, and Invariant. For the agricultural robots, we wish to avoid two undesirable outcomes: (1) if $|d| > 0.5\text{W} = 0.38$ meters, the vehicle will hit the corn, and (2) if $|\psi| > \frac{\pi}{6}$, the camera view will face crops and neural perception will fail. Formally,

$$Safe((d, \psi)) = |d| \leq 0.5\text{W} \wedge |\psi| \leq \frac{\pi}{6}$$

We consider the following initial configurations:

$$Init = \{(\text{env}, (d_0, \psi_0), \omega_0, (x_0, y_0, \theta_0)) \mid ((x_0, y_0, \theta_0), (d_0, \psi_0)) \in Init_{MEnv} \wedge \omega_0 \in Init_{RM}\}$$

$$Init_{MEnv} = \{((x_0, y_0, \theta_0), (d_0, \psi_0)) \mid |d_0| \leq 0.3\text{W} \wedge |\psi_0| \leq \frac{\pi}{12} \wedge (d_0, \psi_0) = (-y_0, -\theta_0)\}$$

$$Init_{RM} = \{\omega_0 \mid |\omega_0| \leq \omega_{max}\}$$

The invariant *Inv* that includes initial configurations *Init* and prove the safety *Safe* is:

$$Inv = \{(loc, (d, \psi), \delta, (x, y, \theta)) \mid V(d', \psi') \leq \max(V(d, \psi), \rho)\}$$

where $V(d, \psi) := \sqrt{(3d + 0.75\psi)^2 + (2\psi)^2}$ is a norm function for proving the safety and stability.

6 EVALUATION

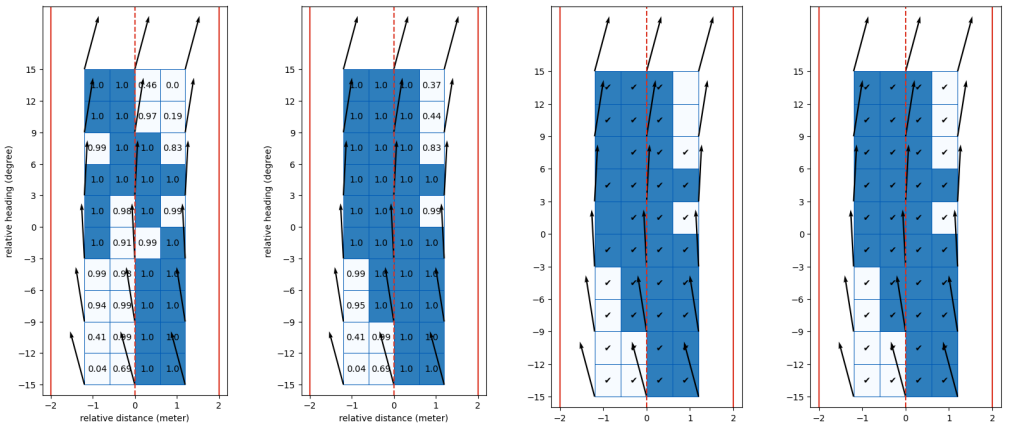
We implemented our technique for synthesizing perception contracts for autonomous systems with ML-based perception in a framework called Perceptor.

Our system is given (a) a system (a reactive module) interacting with a model of the environment, (b) a safety property and an invariant that proves the safety property under perfect perception, and (c) a set of ground truths and their estimates (perception pairs) i.e., $\{(gt_i, gte_i)\}$, Perceptor synthesizes a perception contract that is guaranteed to maintain the system invariant and includes the given set of perception pairs. The ground truths, *gt*, are sampled from a simulated environment. The estimates, *gte*, are derived from images and outputs of ML perception working over them. The resulting Perception contracts synthesized are parameterized over a logic $\mathcal{L}$, which is the one described in Section 4.2.

To assess the efficacy of our approach, we investigate *(1) how effectively can Perceptor synthesize perception contracts that include all positive samples* $((gt, gte), +)$ *while preserving the invariant* and also *(2) how well can Perceptor synthesize perception contracts that generalize.*

6.1 Implementation

Perceptor is implemented in Python and contains approximately 2536 lines of code. We use Quinlan's C 5.0 decision tree algorithm [Mitchell 1997; Quinlan 1986], with some modifications, to build our learner. We ensure our learner can only find exact classifiers for a given set of samples. Additionally, for every individual sample our learner receives, it produces a feature vector that has additional features depending on the logic. Therefore, perception contracts are decision trees (i.e., Boolean combinations) over these new features. We use the optimization solver Gurobi [Gurobi Optimization 2020] as the teacher. Given a candidate contract, Gurobi checks if the perception contract maintains the invariant, i.e., whether Formula (1) is valid, if not, Gurobi returns a set of counterexamples (negatively labeled $((gt, gte), -)$ perception pairs) indicating valuations where the formula does not hold. Due to approximations of trigonometric functions, limitations of floating point arithmetic, etc., Gurobi can return spurious counterexamples, i.e., (gt, gte) pairs that are already excluded by the contract. In these cases, we use Z3 [De Moura and Bjørner 2008] to double-check



(a) Safe and unsafe regions with $\rho = 0.0$ (b) Safe and unsafe regions with $\rho = 1.0$ (c) Regions with learned PCs (checkmarks) for 2D features with $\rho = 1.0$ (d) Regions with learned PCs (checkmarks) for 4D features with $\rho = 1.0$

Fig. 6. Safe and unsafe regions with learned perception contracts with respect to the original and relaxed invariants in combination with two choices of features.

that the returned pairs are true counterexamples. When Gurobi returns spurious counterexamples, if it cannot also return at least one true counterexample (i.e., a pair allowed by the contract but not the invariant after a transition), we deem the contract safe (in these cases, technically, we have not proven the contract to be safe, and using more powerful verification techniques is a potential future direction).

Case Studies, Relaxing Invariants, and Safe Regions. We evaluate our prototype on two case studies of vision-based control systems, a Lane Tracking System (LTS) for an electric vehicle and a navigation system (CropFollow) for a crop row-following agricultural robot, as described in Section 5. There are several aspects of the systems that need to be articulated. First, these systems are proven to be formally safe *under perfect perception*. Second, these systems, with their current neural network based perception, are evidently *not* safe. For example, if the vehicle is at the edge of a road and the heading angle and camera are facing *away* from the center of the lane, then NN perception fails many times, and the vehicle is led off the road by the controller. In fact, there are concrete images in our training set that already show this unsafe behavior.

We refer the reader to Figure 6a that shows the ratio of safe images in the training data. We divide the groundtruth values into regions based on intervals of heading angle (on the y-axis) and the distance of the ego vehicle to the center of the lane (on the x-axis), and represent the fraction of safe images in the training set in each region.

Notice that there are unsafe regions labeled in white (safe regions are those colored blue). In fact, some of these unsafe regions are where the ego vehicle is further away from the center of the lane *and* where the vehicle is facing away from it.

Another set of unsafe regions is when the vehicle is close to the center of the lane. However, in these cases, the vehicle is not truly unsafe. The reason the invariant is not maintained is that even a slight error in perception can make the controller veer the vehicle slightly away from the center of the lane. For example, when the vehicle is slightly to the right of the center of the lane, a mild perception error can place it to the left of the center, making the controller steer it to the right and violate the invariant. As described in Section 3.2, the invariant needs to be typically relaxed to

accommodate some error in perception. Recall from Page 16, Equation 2, that the invariant demands that the V function always decreases, where the V function captures both how close the vehicle is to the center of the lane and how well it is aligned to it. We relax this invariant $V(d', \psi') \leq V(d, \psi)$ to $V(d', \psi') \leq max(V(d, \psi), \rho)$. This relation says the vehicle need not decrease the V-function under a threshold value ρ (as it is already quite close to the center and aligned). Note that this relaxes the invariant only when the vehicle is close to the center of the lane, and hence still assures that the vehicle will not leave the lane.

We now refer the reader to Figure 6b, where the training images are now evaluated with respect to the relaxed invariant above. Notice that now there are no unsafe regions when the distance and heading angle are both small.

The goal of our experiments is to synthesize perception contracts for every region using both the original and relaxed invariant. For each safe region, we want to synthesize a perception contract that includes the perception of all training images in that region. And for unsafe regions, we want to synthesize perception contracts that include only the safe training images in the region.

6.2 Experiment Setup

Our experiments for the Lane Tracking System (LTS) case study span six experiments where the training data is the same across the experiments. In each experiment, the data is divided into 40 partitions (as in Figure 6a). However, experiments are configured differently to study the impact of the feature space and choice of invariant on the effectiveness of our learning architecture. The feature space for configurations are either 2D and 4D values. The 2D feature space consists of two base features $(d_{gt} - d_{gte})$ and $(\psi_{gt} - \psi_{gte})$. That is, the two base features capture the difference between groundtruth and perceived values for distance and heading angle. The derived octagonal constraint features are formed using the two base features (to get 8 features). The 4D space has base features $d_{gt}, \psi_{gt}, d_{gte}$, and ψ_{gte}.

The invariant for the LTS is defined in Equation 2 and it is parameterized by ρ. We use two values for ρ, 0.0 and 1.0. When ρ := 1.0, the invariant is relaxed while when ρ := 0.0, the invariant is strict. We perform four experiments taking two choices for feature space and two choices for values of ρ, and a speed of $2.8m/s$. And an additional two experiments where the speed of the car is assumed to be in the range $[2.5m/s, 3.0m/s]$, for 2D and 4D feature spaces, and with ρ := 1.0.

The setup for the Agricultural Robots case study is the same except that we do two experiments, both for the relaxed invariant, one for the 2D space and the other for the 4D space.

Training Data Preparation. To prepare the training data for the Lane Tracking System (see Section 5.1), we use the Gazebo model for the GEM vehicle [Du et al. 2020] and generate camera images with their groundtruth percepts $gt = (d_{gt}, \psi_{gt})$. Each image is sampled from a uniform distribution over (i) the states of the environment model $Q_{MEnv} = \left\{(x, y, \theta) \mid |y| \leq 0.3\text{W} \wedge |\theta| \leq \frac{\pi}{12}\right\}$ (ii) three types of roads with two, four, and six lanes, and (iii) two lighting conditions, day and dawn. We process each image using LaneNet [Neven et al. 2018] and obtain $gte = (d_{gte}, \psi_{gte})$. In total, we collect 24144 pairs of (gt, gte) as training data.

For each of these images and outputs gte, we check using Gurobi whether the system maintains the invariant from any state satisfying the invariant when the controller is fed gte (note that this labeling is different for the constant speed and variable speed cases). This gives us a positive/negative label for each sample.

The training data for the agricultural robot (AgBot) case study is prepared using the Gazebo model for AgBot in a similar manner with different environments. Images are sampled from a uniform distribution over (i) the states of the environment model $Q_{MEnv} = \left\{(x, y, \theta) \mid |y| \leq 0.3\text{W} \wedge |\theta| \leq \frac{\pi}{12}\right\}$ (ii) five different plant fields, including three stages of corn (baby, small, and adult). We process

Table 4. Learned Perception Contracts (PCs) for LTS

Invariant Relaxation	Feature Space	Safe/Unsafe Regions	# Regions learned PCs	Avg. time per region (s)	Rounds		#Paths in DT	
					Avg.	Max.	Avg.	Max.
Constant speed $v_f = 2.8$								
$\rho = 0.0$	2D	Safe	18/22	577	15.83	35	4.4	11
	2D	Unsafe	13/18	378	11.2	19	3.0	7
	4D	Safe	19/22	1035	30.95	64	2.52	6
	4D	Unsafe	12/18	962	29.0	61	2.5	5
$\rho = 1.0$	2D	Safe	**30/30**	72	5.67	24	1.83	6
	2D	Unsafe	8/10	175	10.5	21	3.25	7
	4D	Safe	26/30	97	8.27	79	1.38	5
	4D	Unsafe	7/10	100	11.43	25	2.29	4
Range of speed $v_f \in [2.5, 3.0]$								
$\rho = 1.0$	2D	Safe	28/30	252	9.5	92	1.68	6
	2D	Unsafe	8/10	664	27.5	58	3.75	10
	4D	Safe	**30/30**	321	14.88	141	1.83	6
	4D	Unsafe	**10/10**	872	29.5	139	3.6	8

each image and obtain groundtruth estimates from the pretrained ResNet-18 model used in CropFollow [Sivakumar et al. 2021]. In total, we collect 7733 pairs of (gt, gte) as the set of samples and labelled them positive and negative, as described above.

6.3 Results: Effectiveness in Learning Perception Contracts

We study the effectiveness of our architecture in learning perception contracts. In this paper, we say an approach is effective if it can learn a perception contract that includes all positive samples and preserves the invariant. We present our results in Table 4 for the Lane Tracking System (LTS) case study and Table 5 for the CropFollow case study. We especially focus on finding perception contracts in safe regions (i.e., when all training data, in a region, maintain the invariant). Nonetheless, the tables show our results for safe as well as unsafe regions. For each configuration, we report whether the 2D/4D feature space was used, the number of regions where our technique was successful in generating a perception contract (for safe as well as unsafe regions), the average time required for synthesis, the average and maximum number of rounds our tool takes to find a correct contract, and the average and maximum number of paths (leaves) in the decision tree representing the synthesized perception contract.

The results for the LTS case study in Table 4 show that our technique is in general able to synthesize perception contracts ($\sim$ 85% of the time). In particular, when the invariant is relaxed ($\rho = 1.0$) and in the 4D case, our tool successfully found perception contracts in all the 30 safe regions (30/30, 100%), for both the constant speed case as well as the variable speed case. Furthermore, for the variable speed case, our tool was also successful in synthesizing contracts in all the unsafe regions as well (where these contracts include all safe images in the region).

For the CropFollow case study in Table 5, our findings show that the tool is effective in finding perception contracts in most cases as well, and in the 4D feature space, succeeds in finding perception contracts in all the 14 safe regions and in all 11 unsafe regions.

6.4 Generalizability of Perception Contracts

In this subsection, we want to study how well our synthesized perception contracts generalize. To do this, we want to evaluate on new images processed by ML perception. We check whether images

Table 5. Learned Perception Contracts (PCs) for AgBot

Invariant Relaxation	Feature Space	Safe/Unsafe Region	# Regions learned PCs	Avg. time per region (s)	Rounds		#Paths in DT	
					Avg.	Max.	Avg.	Max.
$\rho = 1.0$	2D	Safe	14/ 14	4.62	4.21	46	1.07	2
	2D	Unsafe	4 / 11	257	61.5	79	8.0	14
	4D	Safe	**14 / 14**	3.35	3.21	32	1.0	1
	4D	Unsafe	**11 / 11**	1110	45.82	114	5.36	8

that maintain the invariant satisfy the contract and images that do not maintain the invariant violate the contract. We use runtime monitoring of the two case studies to obtain these new images.

Setup for Obtaining Test Set. We validate the learned perception contracts within the context of the closed-loop system. As test data, we use simulation traces of the system and use conformance to the perception contracts for runtime monitoring.

For the lane tracking system, we use the simulator to collect 800 traces as the testing set for the synthesized contracts. Each trace simulates the lane tracking system for 20 seconds and consists of at least 400 pairs of groundtruth values *gt* and estimated values *gte*. We then prepare our test dataset by labeling each pair of *gt* and *gte* positive (negative) with respect to preserving the invariant $Preserve_{Inv}$.

Similarly, we collect 400 traces as the testing set for AgBot. Each trace simulates the AgBot system for 10 seconds and consists of over 200 pairs of images.

Results on Generalizability of Perception Contracts. To calculate precision of a contract over a set of images, we compute the percentage of correctly classified ground truth and estimate pairs, (gt, gte), over the total number of pairs. Note that the pairs are extracted from images evaluated on ML perception.

Table 6 and Table 7 show results on precision of perception contracts synthesized by our tool, for the various configurations involving relaxation of the invariant, feature space, and safe and unsafe regions. Our results on synthesized contracts show high precision ($> 79\%$) for both case studies when the invariant is relaxed.

Using Perception Contracts as Runtime Monitors. Our results show that our learning algorithm has generalized from training data well, and hence can be used for runtime monitoring. On average, it only takes 0.003 seconds to evaluate perception values over the perception contract in our setting, and hence can be used in an online runtime monitoring setting.

The reader may wonder why we cannot do runtime monitoring simply by checking whether the system is within the invariant. Note that the runtime monitoring using perception contracts performs a much better job than simply checking invariants at runtime. When perception on an image during runtime monitoring passes the perception contract, we are guaranteed that in *all states of the system satisfying the invariant* (not just the current state), the current perception will maintain the invariant. For example, consider a car moving at a particular speed on a dry road, and consider perception on an image that keeps the car within the invariant. While naive runtime monitoring will declare this safe, we require a lot more— we want the perception on the image to be safe in all states, where latent variables can be different, like when the car is moving at a different speed or the road is wet. Monitoring using perception contracts gives this assurance.

For example, in Figure 7a, the left image shows an ego vehicle in a three-lane road positioned on the rightmost lane but close to the middle lane, with heading angle towards the middle lane. However, the vision component only perceives (seen in Figure 7b) a two-lane road represented by the three red lines in the image on the right. As a result, the ego vehicle considers itself positioned in the *perceived left lane* instead of the rightmost lane, and maintains its current heading instead

of steering to the right. It turns out that when the vehicle is at a slower speed, the invariant is not violated, but the invariant is violated at higher speeds (the invariant is a complex function). Consequently, if we were doing naive runtime monitoring when the car is going slow, we may declare the perception is safe, though it is not, while our perception contract will declare the perception unsafe.

Table 6. Precision represents the percentage of correctly classified testing data in the LTS case study.

Invariant	Feature Space	Safe/Unsafe Region	Precision
$\rho = 0.0$	2D	Safe	90.44%
	2D	Unsafe	90.72%
	4D	Safe	91.58%
	4D	Unsafe	53.58%
$\rho = 1.0$	2D	Safe	99.91%
	2D	Unsafe	79.59%
	4D	Safe	99.89%
	4D	Unsafe	88.57%

Table 7. Precision represents the percentage of correctly classified testing data in the AgBot case study.

Relaxation	Feature Space	Safe/Unsafe Region	Precision (%)
$\rho = 1.0$	2D	Safe	100.0%
	2D	Unsafe	83.33%
	4D	Safe	100.0%
	4D	Unsafe	99.49%

6.5 Comparison with Approximate Abstractions of Perception

The work in [Hsieh et al. 2022b] creates approximate abstractions of perception (AAPs) that are similar to perception contracts in that it captures errors in perception that preserve the invariant. However, there are many differences; AAPs do not intend to capture *all* safe images in regions unlike perception contracts. AAPs capture errors using simple shapes like spheres (and hence convex regions) while perception contracts use a logic that represents various shapes (which is why they can include all safe images). In short, AAPs are not perception contracts as defined in this paper.

We however compare experimentally our learned PCs with AAPs generated using the tool in [Hsieh et al. 2022b]. In order to run the tool for AAPs following [Hsieh et al. 2022b], we collect the training data by uniformly sampling different positions and orientations of the LTS on the one-road scenario and collect the testing data using the three-road scenario. Further, we need to simplify the system model with constant speed $v_f = 2.8$ so that we can obtain AAPs.

Table 8 shows the results comparing the effectiveness of the two approaches. It includes the time for synthesis, the percentage of included pairs in training data, and precision with respect to testing data. We first observe that, unsurprisingly, inferring AAPs is much faster than learning PCs. However, AAPs do not provide the strong guarantee by PCs; they do not include all positive pairs in training data. Especially, for unsafe regions of states, the inferred AAPs include only half of the pairs on average and none of the pairs in the worst case. On the other hand, our approaches using either 2D or 4D feature space can consistently find PCs that include almost all positive pairs

Table 8. Comparison between Approximate Abstractions of Perception (AAPs) from [Hsieh et al. 2022b] and Perception Contracts (PCs) for the LTS case study with invariant relaxation $\rho = 1.0$ and constant speed $v_f = 2.8$ m/s.

	PCs by Perceptor		AAPs
	2D	4D	
# Regions learned	38/40	33/40	**40/40**
Avg. time for synthesis per region (s)	93.68	97.6	**5.3**
Percentage of included positive pairs in training data (%)			
Worst region in safe regions	**100.0%**	99.62%	64.16%
Avg. over safe regions	**100.0%**	99.98%	94.68%
Worst region in unsafe regions	**100.0%**	**100.0%**	0.0%
Avg. over unsafe regions	**100.0%**	**100.0%**	50.94%
Precision w.r.t testing data from uniform distribution (%)			
Worst region in safe regions	**98.99%**	98.79%	71.41%
Avg. over safe regions	**99.90%**	99.89%	96.44%
Worst region in unsafe regions	72.34%	**80.85%**	0.0%
Avg. over unsafe regions	96.40%	**96.61%**	56.83%

for all regions (the entries that are close to but not 100% are due to numerical errors in Gurobi to determine true safety of perception on images). Similarly, when the testing data is gathered from a uniform distribution over positions and orientations, PCs achieve higher precision scores than AAPs regardless of the texture and the color of the roads in the three-road scenario.

6.6 Example Perception Contract

In this section, we show a perception contract synthesized by Perceptor. We further illustrate how to interpret the contract to better understand the constraint it poses on estimated percept values. An example contract synthesized in the LTS case study, is the decision tree:

$$\begin{aligned} &\mathit{If}\ (y + \theta + d + \psi \geq 0.163) \\ &\quad \mathit{then}\ (y + \theta - \psi \geq 0.859 \wedge y + \theta + d + \psi < 0.415) \\ &\quad \mathit{else}\ \theta + \psi < 0.132 \end{aligned}$$

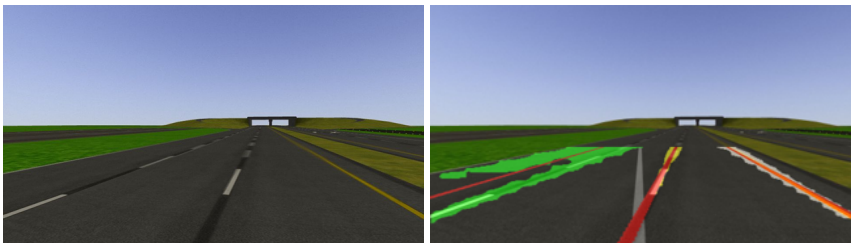
corresponding to region $y \in [0.6, 1.2]$ meters and $\theta \in [-0.16, -0.10]$ radians. The region associated with the perception contract indicates that the car would be positioned to the left of the center line within the range of 0.6 to 1.2 meters away from the line. The range of values for the heading angle indicate that the vehicle is oriented toward the center line as depicted in Figure 4.

The variables d and ψ are estimates of the *negative* of y and θ, respectively. Hence when there is perfect perception for estimating relative distance and relative heading angles, $d = -y$ and $\psi = -\theta$ as defined in Section 5.1.

7 RELATED WORK

Related work spans the areas of program synthesis, specification mining and quality assurance techniques for autonomous systems.

Analysis of Closed-Loop Systems with NN Perceptions. The closest related work is the recent paper [Hsieh et al. 2022b] by Hsieh et al. In that work, a notion called *approximate abstraction of perception (AAP)* was developed to create abstractions of neural perception modules, which were then used for system-level safety analysis. Similar to the perception contracts proposed here,



(a) The image shows a three-lane road from the perspective of a car positioned in the rightmost lane.

(b) Image from LaneNet misperceiving lane boundaries.

Fig. 7. Images depicting actual view of road vs vision estimate

AAPs relate groundtruth values with the groundtruth estimates produced by neural perception. The relation there is defined in terms of piece-wise error bounds that are derived from sampled data as well as the constraints imposed by the system invariant. In [Hsieh et al. 2022a,b], it has been shown how AAPs can be used to establish system-level safety invariant properties for a realistic lane tracking system, a formation control system for drones, and a row-following system for an agricultural robot. The industry paper [Abraham et al. 2022] discusses the relevance of these approaches in the context of engineering autonomous systems. There are several salient differences between these prior works and this paper. (1) This work gives a much more general formalization of perception contracts (PCs), including stateful environments and reactive modules. (2) We formulate the general problem of synthesizing perception contracts (which is independent of the logic used for the contracts). Finally, (3) we propose an iterative learning approach that synthesizes PCs that includes all positive samples from neural perception module and maintains a system invariant.

Other closely related works are VerifAI [Dreossi et al. 2019; Ghosh et al. 2021] by Dreossi and Ghosh et al., [Katz et al. 2021] by Katz et al., and NNLander-VeriF [Santa Cruz and Shoukry 2022]. VerifAI [Dreossi et al. 2019] and related publications [Fremont et al. 2020, 2022] provide a comprehensive framework to *falsify* a closed loop system with ML-based perception. Further, the Counter Example Guided Inductive Synthesis (CEGIS) based approach in [Ghosh et al. 2021] uses VerifAI to find counterexamples of the closed-loop system, synthesizes a controller, and learns a surrogate model. Their techniques focus on the falsification of the system specification. [Katz et al. 2021] trains generative adversarial networks (GANs) to produce a simpler network for DNN-based perception. NNLander-VeriF [Santa Cruz and Shoukry 2022] verifies NN perception along with NN controllers for an autonomous landing system.

Isolated Neural Network Verification. Recently, there are many works on verifying an isolated neural network such as ReLuplex [Katz et al. 2017], NNV [Tran et al. 2020], Verisig [Ivanov et al. 2019], etc. We refer readers to a summary of the VNN competition [Bak et al. 2021a] for a complete list. Our notion and learning approach of *PC* can decompose the system level specification and search for the specification for the neural network perception, and the NN verification tool can be applied to check neural network perception with respect to *PC*.

Program Synthesis. Program synthesis deals with the problem of synthesizing expressions that satisfy a specification. Counterexample-guided inductive synthesis (CEGIS) [Alur et al. 2015] is one of the most promising approaches in synthesis and resembles online learning. In this setting, the target expression is learned in multiple rounds of interaction between a learner and a verifier. In each round, the learner proposes a candidate expression and the verifier checks whether the expression meets the specification. Our work can be seen as instance of CEGIS. where along with a verifier, we also rely on a vision component for positive examples via simulation.

Specification Mining. Our work can also be seen as mining specifications [Alur et al. 2005; Ammons et al. 2002; Astorga et al. 2019, 2021; Ernst et al. 1999; Henzinger et al. 2005; Whaley et al. 2002; Xie et al. 2006] for ML-based vision components. In this setting, mining approaches observe program executions to build abstractions of the specification by generalizing from observed runs. Ammons et. al [Ammons et al. 2002] are the first to propose this line of work. It learns Probabilistic Finite State Automatons (PFA) to represent valid method call sequences. Various approaches have been proposed since then on automata learning [Alur et al. 2005; Henzinger et al. 2005; Whaley et al. 2002; Xie et al. 2006]. Ernst et al. [Ernst et al. 1999] proposed Daikon for dynamically inferring *conjunctive* Boolean formulas as likely invariants from black-box executions by collecting program state at method entry and exit. The work by [Jahangirova et al. 2016] also uses mutation testing to infer assertions, and iterates over rounds *with the programmer* to infer specs.

8 CONCLUSION

We have introduced perception contracts that formalize groundtruth estimation deviations by neural perception that preserve invariants of systems. We additionally argue that they can form contracts for neural perception modules for ensuring safety. We have also studied the problem of synthesizing perception contracts given a set of images whose groundtruth estimate by neural perception maintains the invariant. We have evaluated our synthesis algorithm on two realistic vision-based control systems, for efficacy and precision. We also demonstrate the effectiveness of perception contracts as runtime monitors over simulation traces of these vision-based control systems. Several future directions are interesting. First, perception modules in autonomous vehicles, drones, robots, etc. glean groundtruth not just from a single image, but a sequence of frames. Extending the notion of perception contracts to sequences of frames augmented with reasoning mechanisms for perception is an interesting future direction. Second, it would be interesting to perform larger experiments that subject vision-based control systems to continuous runtime monitoring, checking synthesized perception contracts to validate them. Third, we believe that our techniques are orthogonal to other approaches for evaluating perception correctness such as robustness and testing against image generators; exploring combinations of these techniques would be interesting, especially using perception contracts as specifications for these ML components. Finally, it would be interesting to utilize synthesized perception contracts for both runtime monitoring as well as falsification, exploiting the fact that these are local contracts for the perception module. Evaluating the efficacy of perception contract guided monitoring and falsification in comparison with traditional monitoring and falsification techniques, would be interesting. We are also intrigued as to how perception contracts, monitoring, and falsification extend to more complex properties than safety, such as temporal properties of systems [Lukina et al. 2021; Mamouras et al. 2021].

ACKNOWLEDGMENTS

This work is supported in part by research grants from Amazon, Discovery Partners Institute (DPI) science team seed grants, USDA National Institute of Food and Agriculture (USDA/NIFA #2021-67021-33449), and the Boeing company.

REFERENCES

Michael Abraham, Aaron Mayne, Tristan Perez, Italo Romani De Oliveira, Huafeng Yu, Chiao Hsieh, Yangge Li, Dawei Sun, and Sayan Mitra. 2022. Industry-track: Challenges in Rebooting Autonomy with Deep Learned Perception. In *2022 International Conference on Embedded Software (EMSOFT)*. 17–20. https://doi.org/10.1109/EMSOFT55006.2022.00016

Rajeev Alur. 2015. *Principles of Cyber-Physical Systems*. MIT Press, Cambridge, MA.

Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama,

Emina Torlak, and Abhishek Udupa. 2015. Syntax-guided synthesis. In *Dependable Software Systems Engineering*. NATO Science for Peace and Security Series, D: Information and Communication Security, Vol. 40.

Rajeev Alur, Pavol Černý, P. Madhusudan, and Wonhong Nam. 2005. Synthesis of Interface Specifications for Java Classes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2005)*. 98–109. https://doi.org/10.1145/1047659.1040314

Glenn Ammons, Rastislav Bodík, and James R. Larus. 2002. Mining Specifications. In *POPL 2002*.

Angello Astorga, P. Madhusudan, Shambwaditya Saha, Shiyu Wang, and Tao Xie. 2019. Learning Stateful Preconditions modulo a Test Generator. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. 775–787. https://doi.org/10.1145/3314221.3314641

Angello Astorga, Shambwaditya Saha, Ahmad Dinkins, Felicia Wang, P. Madhusudan, and Tao Xie. 2021. Synthesizing Contracts Correct modulo a Test Generator. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 104 (oct 2021), 27 pages. https://doi.org/10.1145/3485481

Karl Johan Astrom and Richard M. Murray. 2008. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, USA.

Stanley Bak, Changliu Liu, and Taylor Johnson. 2021a. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. arXiv:2109.00498 https://arxiv.org/abs/2109.00498

Stanley Bak, Changliu Liu, and Taylor T. Johnson. 2021b. The Second International Verification of Neural Networks Competition (VNN-COMP 2021): Summary and Results. *CoRR* abs/2109.00498 (2021). arXiv:2109.00498 https://arxiv.org/abs/2109.00498

Gregory Bonaert, Dimitar I. Dimitrov, Maximilian Baader, and Martin Vechev. 2021. Fast and Precise Certification of Transformers. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 466–481. https://doi.org/10.1145/3453483.3454056

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS'08/ETAPS'08* (Budapest, Hungary). Springer-Verlag, Berlin, Heidelberg, 337–340.

Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2019. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In *Computer Aided Verification (Lecture Notes in Computer Science)*. Springer International Publishing, Cham, 432–442. https://doi.org/10.1007/978-3-030-25540-4_25

Peter Du, Zhe Huang, Tianqi Liu, Tianchen Ji, Ke Xu, Qichao Gao, Hussein Sibai, Katherine Driggs-Campbell, and Sayan Mitra. 2020. Online Monitoring for Safe Pedestrian-Vehicle Interactions. In *2020 IEEE 23rd Int. Conf. Intell. Transportation Systems (ITSC)*. 1–8. https://doi.org/10.1109/ITSC45102.2020.9294366

Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. 1999. Dynamically Discovering Likely Program Invariants to Support Program Evolution. In *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*. 213–224. https://doi.org/10.1145/302405.302467

Daniel J. Fremont, Johnathan Chiu, Dragos D. Margineantu, Denis Osipychev, and Sanjit A. Seshia. 2020. Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI. In *Computer Aided Verification (Lecture Notes in Computer Science)*. Springer International Publishing, Cham, 122–134. https://doi.org/10.1007/978-3-030-53288-8_6

Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Scenic: A Language for Scenario Specification and Data Generation. *Mach Learn* (Feb. 2022). https://doi.org/10.1007/s10994-021-06120-5

Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. 2014. ICE: A Robust Framework for Learning Invariants. In *Computer Aided Verification*, Armin Biere and Roderick Bloem (Eds.). Springer International Publishing, Cham, 69–87.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 3–18. https://doi.org/10.1109/SP.2018.00058

Shromona Ghosh, Yash Vardhan Pant, Hadi Ravanbakhsh, and Sanjit A. Seshia. 2021. Counterexample-Guided Synthesis of Perception Models and Control. In *2021 American Control Conference (ACC)*. 3447–3454. https://doi.org/10.23919/ACC50511.2021.9482896

John E Gibson, Eccles S McVey, Clive Douglas Leedham, et al. 1961. Stability of nonlinear control systems by the second method of Liapunov. (1961).

LLC Gurobi Optimization. 2020. Gurobi Optimizer Reference Manual. http://www.gurobi.com

Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. 2005. Permissive Interfaces. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*. 31–40. https://doi.org/10.1145/1081706.1081713

Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun. 2007. Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing. In *2007 American Control Conference*. 2296–2301. https://doi.org/10.1109/ACC.2007.4282788

Chiao Hsieh, Yangge Li, Yubin Koh, and Sayan Mitra. 2022a. Assuring safety of vision-based swarm formation control. https://doi.org/10.48550/ARXIV.2210.00982

Chiao Hsieh, Yangge Li, Dawei Sun, Keyur Joshi, Sasa Misailovic, and Sayan Mitra. 2022b. Verifying Controllers With Vision-Based Perception Using Safe Approximate Abstractions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4205–4216. https://doi.org/10.1109/TCAD.2022.3197508

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers. In *Proc. 22nd ACM Int. Conf. HSCC*. 169–178.

Gunel Jahangirova, David Clark, Mark Harman, and Paolo Tonella. 2016. Test Oracle Assessment and Improvement. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. 247–258. https://doi.org/10.1145/2931037.2931062

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proc. 29th Int. Conf. CAV*. 97–117.

Sydney M. Katz, Anthony L. Corso, Christopher A. Strong, and Mykel J. Kochenderfer. 2021. Verification of Image-based Neural Network Controllers Using Generative Models. In *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. 1–10. https://doi.org/10.1109/DASC52595.2021.9594360

Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-Robust Neural Networks. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 6212–6222. http://proceedings.mlr.press/v139/leino21a.html

Anna Lukina, Christian Schilling, and Thomas A. Henzinger. 2021. Into the Unknown: Active Monitoring of Neural Networks. In *Runtime Verification*, Lu Feng and Dana Fisman (Eds.). Springer International Publishing, Cham, 42–61.

Konstantinos Mamouras, Agnishom Chattopadhyay, and Zhifu Wang. 2021. A Compositional Framework for Quantitative Online Monitoring over Continuous-Time Signals. In *Runtime Verification*, Lu Feng and Dana Fisman (Eds.). Springer International Publishing, Cham, 142–163.

Thomas M. Mitchell. 1997. *Machine Learning* (1 ed.).

Sayan Mitra. 2021. *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. MIT Press.

Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. 2018. Towards End-to-End Lane Detection: an Instance Segmentation Approach. In *2018 IEEE Intell. Veh. Symp.* 286–291. https://doi.org/10.1109/IVS.2018.8500547

J. R. Quinlan. 1986. Induction of Decision Trees. *Mach. Learn.* 1, 1 (1986).

Ulices Santa Cruz and Yasser Shoukry. 2022. NNLander-VeriF: A Neural Network Formal Verification Framework for Vision-Based Autonomous Aircraft Landing. In *NASA Formal Methods*, Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez (Eds.). Springer International Publishing, Cham, 213–230.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *Proc. ACM Program. Lang.* 3, POPL, Article 41 (jan 2019), 30 pages. https://doi.org/10.1145/3290354

Arun Narenthiran Sivakumar, Sahil Modi, Mateus Valverde Gasparino, Che Ellis, Andres Eduardo Baquero Velasquez, Girish Chowdhary, and Saurabh Gupta. 2021. Learned Visual Navigation for Under-Canopy Agricultural Robots. In *Proc. Robotics: Science and Systems*. Virtual. https://doi.org/10.15607/RSS.2021.XVII.019

Eduardo D. Sontag. 2008. *Input to State Stability: Basic Concepts and Results*. Springer Berlin Heidelberg, Berlin, Heidelberg, 163–220. https://doi.org/10.1007/978-3-540-77653-6_3

Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=HyGIdiRqtm

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In *Proc. 32nd Int. Conf. CAV*. 3–17.

John Whaley, Michael C. Martin, and Monica S. Lam. 2002. Automatic Extraction of Object-Oriented Component Interfaces. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2002)*. 218–228. https://doi.org/10.1145/566172.566212

Tao Xie, Evan Martin, and Hai Yuan. 2006. Automatic Extraction of Abstract-Object-State Machines from Unit-Test Executions. In *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*. 835–838. https://doi.org/10.1145/1134285.1134427

Received 2023-04-14; accepted 2023-08-27