Risk-Driven Design of Perception Systems

Anthony L. Corso *
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA
acorso@stanford.edu

Sydney M. Katz *
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA
smkatz@stanford.edu

Craig Innes
School of Informatics
University of Edinburgh
Edinburgh, UK
craig.innes@ed.ac.uk

Xin Du
School of Informatics
University of Edinburgh
Edinburgh, UK
x.du@ed.ac.uk

Subramanian Ramamoorthy
School of Informatics
University of Edinburgh
Edinburgh, UK
s.ramamoorthy@ed.ac.uk

Mykel J. Kochenderfer
Department of Aeronautics and Astronautics
Stanford University
Stanford, CA
mykel@stanford.edu

Abstract

Modern autonomous systems rely on perception modules to process complex sensor measurements into state estimates. These estimates are then passed to a controller, which uses them to make safety-critical decisions. It is therefore important that we design perception systems to minimize errors that reduce the overall safety of the system. We develop a risk-driven approach to designing perception systems that accounts for the effect of perceptual errors on the performance of the fully-integrated, closed-loop system. We formulate a risk function to quantify the effect of a given perceptual error on overall safety, and show how we can use it to design safer perception systems by including a risk-dependent term in the loss function and generating training data in risk-sensitive regions. We evaluate our techniques on a realistic vision-based aircraft detect and avoid application and show that risk-driven design reduces collision risk by 37 % over a baseline system.

1 Introduction

The design of reliable perception systems is a key challenge in the development of safety-critical autonomous systems [1], [2]. Modern perception systems are often required to predict state information from complex, high-dimensional inputs such as images or LiDAR data [3]–[5]. This information is then passed to a controller, which uses the state estimate to make safety-critical decisions. For example, vision-based perception systems have been proposed for detect and avoid applications in aviation [5]. It is important that perception systems produce accurate estimates, and these systems are typically trained to minimize overall perceptual error using a regression loss on a set of training data. This training method, however, fails to account for the effect of perceptual errors on the performance of the fully-integrated, closed-loop control system. In particular, not all errors will have an equal

*equal contribution

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

effect on overall performance. While some errors pose minimal risk, others may lead to catastrophic outcomes [6], [7].

Contribution In this work, we develop a technique to design perception systems that are sensitive to the overall risk of the closed-loop system in which they operate. We formulate a risk function that quantifies the downstream effect on safety of making a given perceptual error in a given state. We then show how the risk function can be used to design safer perception systems by incorporating it into the loss function during training and using it to focus data collection efforts on the most risk-sensitive regions of the state space. Finally, we analyze the impact of our risk-driven design approach on the safety of a realistic vision-based aircraft detect and avoid system. We show that our approach is able to reduce collision risk by up to 37 % over a baseline perception system.

2 Background

This work uses a notion of conditional value at risk (CVaR) and a Markov decision process (MDP) formulation to estimate the effect of perceptual errors on the safety of a closed-loop system. This section details the necessary background on these topics.

Conditional Value at Risk (CVaR) Let X be a bounded random variable and $F(x) = P(X \leq x)$ be its cumulative distribution function. The value at risk (VaR) is the highest value that X is guaranteed not to exceed with probability α written as

$$\text{VaR}_\alpha(X) = \min\{x \mid F(x) \geq \alpha\} \tag{1}$$

Using the VaR, we can derive the CVaR for a given $\alpha \in (0, 1)$, which represents the expected value of the top $1 - \alpha$ quantile of the probability distribution over X. More formally,

$$\text{CVaR}_\alpha(X) = \mathbb{E}[X \mid X \geq \text{VaR}_\alpha(X)] \tag{2}$$

If X is random variable that represents cost, $\text{CVaR}_\alpha(X)$ will correspond to the expected value of the worst α-fraction of costs. As α approaches 1, $\text{CVaR}_\alpha(X)$ approaches the cost of the worst-case outcome. As α approaches 0, $\text{CVaR}_\alpha(X)$ approaches the expected value of X. CVaR has some desirable mathematical properties [8] that have often made it the preferred risk metric in risk-sensitive domains such as finance [9]–[11] and robotics [8].

Markov Decision Process (MDP) An MDP is a way of encoding a sequential decision making problem in which an agent's action at each time step depends only on its current state [12]. An MDP is defined by the tuple (S, A, T, C, γ), where S is the state space, A is the action space, $T(s' \mid s, a)$ is the probability of transitioning to state s' given that we are in state s and take action a, $C(s, a)$ is the cost of taking action a in state s, and γ is the discount factor. A stochastic policy for an MDP, written as $\pi(a \mid s)$, denotes the likelihood of taking action a in state s and induces a distribution over future costs. We define a cost distribution function $Z^\pi(s, a)$ that maps a given state s and action a to a random variable representing the sum of discounted future costs obtained by taking action a from state s and subsequently following policy π as follows

$$Z^\pi(s_t, a_t) = C(s_t, a_t) + \sum_{t'=t+1}^{\infty} \gamma^{t'} C(s_{t'}, a_{t'}) \tag{3}$$

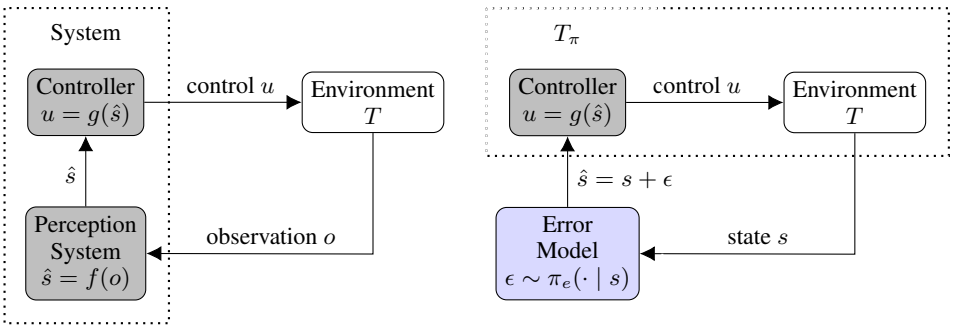
where $s_{t'+1} \sim T(\cdot \mid s_{t'}, a_{t'})$ and $a_{t'} \sim \pi(\cdot \mid s_{t'})$. To evaluate a given policy π, we calculate a state-action value function, denoted $Q^\pi(s, a)$, as the expected sum of discounted future costs

$$Q^\pi(s, a) = \mathbb{E}\left[Z^\pi(s, a)\right] \tag{4}$$

In traditional reinforcement learning, Q^π is the optimization objective [12]; however, in safety-critical domains we may wish evaluate a policy based on an upper quantile of the worst-case outcomes. In this case, rather than calculating the expectation of future costs, we define a new state-action value function $Q^\pi_\alpha(s, a)$ that represents the CVaR of the distribution of future costs as

$$Q^\pi_\alpha(s, a) = \text{CVaR}_\alpha[Z^\pi(s, a)] \tag{5}$$

Intuitively, $Q^\pi_\alpha(s, a)$ is the expected value of the α-percentile of worst-case outcomes. We note that when $\alpha = 0$, eq. (5) reduces to eq. (4). One way to solve for $Q^\pi_\alpha(s, a)$ is to first solve for the cost distribution function $Z^\pi(s, a)$, which can be done using techniques from distributional reinforcement learning [13]. The CVaR can then be computed using this distribution. Other techniques that do not require explicitly solving for the distribution over costs may also be applied [10], [14].



(a) System Architecture (b) Abstracted Perception MDP

Figure 1: Problem formulation.

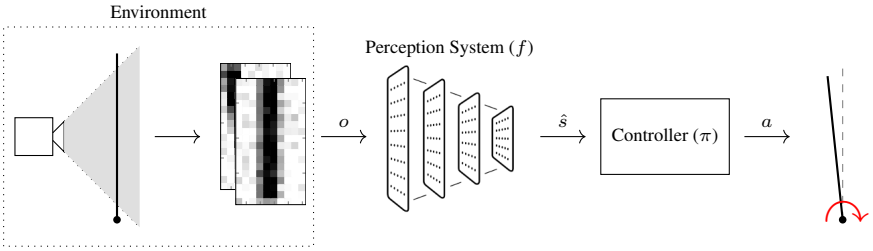


Figure 2: Overview of the inverted pendulum example system.

3 Approach

We consider the setting shown in fig. 1a, in which a system is composed of a perception module f and a controller g. Given an observation $o \in \mathcal{O}$ of the true state $s \in \mathcal{S}$, the perception system produces an estimate of the state $\hat{s} = f(o)$. The controller produces a control input $u = g(\hat{s})$ and the system transitions to state s' with probability $T(s' \mid s, u)$. We assume that the controller g is given, and we wish to design a perception module f such that the overall system satisfies a safety property.

Pendulum example We will use a vision-based inverted pendulum problem as a running example to describe our approach (see Figure 2). We assume that a camera produces noisy image observations of the pendulum's current state $s = [\theta, \omega]$ where θ is its angle from the vertical and ω is its angular velocity. To estimate both position and velocity information from the images, we use two consecutive image frames for each observation. The perception system is a multi-layer perceptron (MLP) that takes in the image observations and produces an estimate of the state. Using the state estimate, the controller selects a torque to keep the pendulum upright. We specify a safety property that requires that $|\theta| < \pi/4$ at all time steps. We use a rule-based controller that satisfies the safety property under perfect perception; however, image noise, image downsampling, and the finite capacity of the MLP make perception errors inevitable, and the goal is to design a perception system that limits the occurrence of high risk errors. Appendix A.1 summarizes the inverted pendulum model.

3.1 Overview

The key insight in our approach is to abstract the perception system using a notional model of perception errors and analyze the effect of specific perception errors on the performance of the closed-loop system. As shown in fig. 1b, we model the errors $\epsilon \in \mathcal{E}$ of the abstracted perception system as a stochastic policy $\pi_e(\epsilon \mid s)$ in an MDP. We note that this representation implicitly takes into account the possible observations for state s by directly outputting the errors the perception system could make when processing each observation. The cost function of the MDP is chosen to reflect the risk of a being in a state with respect to a safety property. By formulating the problem in this way, we can use the policy evaluation techniques outlined in section 2 to evaluate the risk of making perception error ϵ in state s. Given that π_e is an abstraction of the perception system, it will not exactly reflect the true distribution of perception errors. To account for this mismatch, we are

conservative in our risk estimate by evaluating the policy according to eq. (5). Once computed, the risk function represents the downstream impact of making a given perception error in the current state. This function can therefore be used in a supervised learning setting to encourage a model to avoid making high-risk errors. We also show that the risk function can be used to identify error-sensitive regions of the state space, from which additional training data can be collected.

3.2 Risk Function

Our goal is to define a risk function $\rho(s, \epsilon)$ that quantifies the risk of making perception error ϵ in state s when using a controller g. We model the perception error as a stochastic policy $\pi_e(\epsilon \mid s)$ that reflects a notional distribution over errors in state s. The controller produces an action based on the perceived state $\hat{s}$, which is computed from s and ϵ. For additive noise, the resulting transition function for the abstracted perception MDP is

$$T_\pi(s' \mid s, \epsilon) = T(s' \mid s, g(s + \epsilon)) \tag{6}$$

The error model π_e can be designed to match the performance of a previously trained baseline perception system or using domain knowledge. If we are confident that our choice closely matches the errors we will see when designing the perception system, we can set our risk function to match the state action-value function in eq. (4). However, if we have not yet designed the perception system, there will be a mismatch between our estimate of π_e and the true distribution of perception errors.

In these scenarios, we may want a more conservative risk function in which we consider the CVaR of the distribution over future costs when making perception error ϵ in state s given that all future perception errors are distributed according to π_e. Therefore, we select our risk function as follows

$$\rho_\alpha^{\pi_e}(s, \epsilon) = Q_\alpha^{\pi_e}(s, \epsilon) \tag{7}$$

where $Q_\alpha^{\pi_e}(s, \epsilon)$ refers to the CVaR state-action value function defined in eq. (5) for the abstracted perception MDP. The parameter α smoothly controls the degree of conservatism. When $\alpha = 0$, the risk function is equivalent to the expected cost when following the notional perception model. When $\alpha = 1$, we consider only the worst-case sequence of possible perception errors.

To solve for the CVaR state-action value function we apply distributional dynamic programming to compute the distribution over costs. Distributional dynamic programming (alg. 5.3 in Bellemare *et al.* [13]) resembles value iteration where the Bellman operator is replaced with a distributional version to update a parameterized distribution. In this work, we approximate the distribution over costs using a categorical distribution with a fixed number of discrete costs. Additionally, we discretize the states and actions and use a linear weighting function for local approximation. To extend our method to problems with larger state and action spaces, we could apply function approximation techniques developed in prior work [15]–[17], but we leave this as future work. Once we solve for the cost distributions, we can calculate $\rho_\alpha^{\pi_e}(s, \epsilon)$ for any value of α using eq. (2) with minimal additional computation.

Pendulum example For the inverted pendulum system, we model the perception errors $\epsilon = [\epsilon_\theta, \epsilon_\omega]$ as additive noise on the true state ($\hat{s} = s + \epsilon$) distributed according to a multivariate Gaussian distribution with zero mean and diagonal covariance. The cost function penalizes the absolute deviation from the vertical $|\theta|$. See appendix A.2 for more detail. Figure 3 shows the values of the pendulum risk function at the state $s = [0.2, 0.0]$ over the set of errors for multiple values of α. A higher value of α will put more weight on worst-case perception errors in the future, so as α increases, the overall risk values increase. The risk function also provides a notion of *which* perception errors are most risky in a given state. For example, the optimal action when $s = [0.2, 0.0]$ is to apply a negative torque to move the pendulum closer to upright, but if perception errors are negative, the resulting $\hat{s}$ may indicate that the pendulum is at a smaller angle and already moving in the negative direction as shown in the leftmost image on fig. 3. This estimate may cause the controller to produce a smaller torque or even a torque in the positive direction, leading to high risk of falling over.

3.3 Risk-Sensitive Loss Function

Perceptual errors are inevitable whenever there is sensor noise, occlusions, limited training data, or limited model capacity. Our goal is to ensure that the errors that are made by the perception system are in the direction of lower, rather than higher, risk. We incorporate this notion into our design by

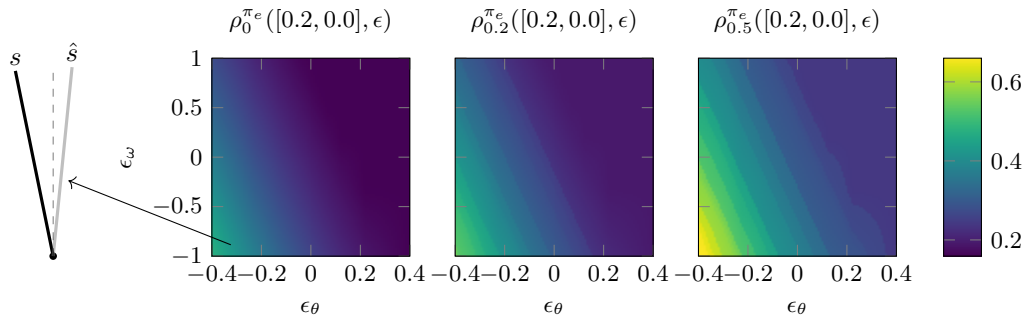


Figure 3: Values of $\rho_\alpha^{\pi_e}([0.2, 0.0], \epsilon)$ for different values of α. The state and an example error are depicted on the left.

formulating a risk-sensitive loss function. Let $\mathcal{L}(s, \hat{s})$ be the standard loss function for the baseline perception system such as mean squared error for regression tasks or cross entropy for classification tasks. We define a risk-sensitive loss function for a single datapoint with additive noise as follows

$$\mathcal{L}_R(s, \hat{s}) = \mathcal{L}(s, \hat{s}) + \lambda \rho_\alpha^{\pi_e}(s, \hat{s} - s) \tag{8}$$

where λ is a hyperparameter that controls the relative weighting of each term. The first term in the loss function pushes all perception errors toward zero, while the second term seeks to minimize the risk of the current errors that the model makes. We note that this is different from simply minimizing perception errors in states with high risk values. If the risk function outputs the same value for a given state regardless of perception error, the gradient of the risk term in the loss function will be zero.

Pendulum example For the inverted pendulum problem, we use a mean squared error baseline loss function, which penalizes positive and negative errors equally. However, a positive and negative error of the same magnitude do not necessarily have an equal effect on the overall risk of the closed-loop system. For example, consider the results shown in fig. 3 where the pendulum is positioned at an angle of 0.2 from the vertical. The risk function shows that a perception error of $\epsilon_\theta = 0.2$ ($\hat{\theta} = 0.4$) is less risky than a perception error of $\epsilon_\theta = -0.2$ ($\hat{\theta} = 0.0$). Intuitively, predicting that the pendulum is tipped further in the positive direction will still result in a negative torque to move the pendulum closer to upright. In contrast, predicting that the pendulum is perfectly upright will cause the controller to output zero torque. The second term in eq. (8) accounts for this effect.

3.4 Risk-Driven Data Generation

In addition to considering the directionality of the errors, we can use the risk function we formulated in section 3.2 to identify states that are especially sensitive to perception errors and generate more training data in those regions. In effect, this process gives higher weight to risky states during training. Since training with a regression loss will push all errors toward zero, we define risky states as states in which making a nonzero perception error will result in a high risk value compared to the risk of zero error. We use a weighting function $w_\alpha(s)$ to capture this notion as follows

$$w_\alpha(s) = \max_{\epsilon \in \mathcal{E}} \rho_\alpha^{\pi_e}(s, \epsilon) - \rho_\alpha^{\pi_e}(s, 0) \tag{9}$$

We note that $w_\alpha(s)$ is not the same as (and not necessarily correlated with) the risk of being in state s. Instead, $w_\alpha(s)$ represents the risk of making a nonzero perception error in state s. For instance, states in which failure is inevitable regardless of perception error will have high risk values but low weight assigned to them according to $w_\alpha(s)$. To sample data, we use rejection sampling with $w_\alpha(s)$ as the likelihood function.

Pendulum example Figure 4 illustrates each term in eq. (9) for the inverted pendulum example problem. The lower left and upper right regions of the state space represent states in which the pendulum is at a high angle and moving in a direction away from the vertical. Due to underactuation of the pendulum system, the states at these extremes are guaranteed to lead to failure regardless of perception error. States in the central band of each plot, on the other hand, constitute states in which the pendulum is likely to remain upright as long as proper control is applied. This band widens when we consider the effect of having zero perception error compared to the worst-case perception

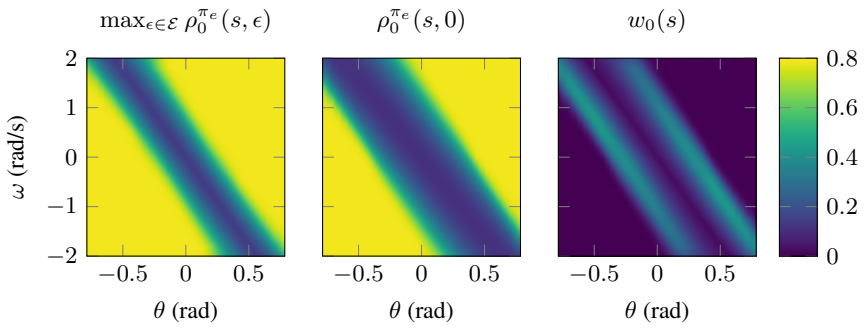


Figure 4: Illustration of eq. (9).

error. By subtracting these quantities, we can understand in which states having zero perception error is most critical. Given a limited data budget, we can focus our data collection on these states by sampling data according to $w_\alpha(s)$.

3.5 Hypotheses

We design the experiments in this work to test the following two hypotheses:

- **H1**: A perception module trained using the risk-sensitive loss function proposed in section 3.3 will result in a safer closed-loop system than one trained using the baseline loss on the same dataset.
- **H2**: A perception system trained on data sampled according to the risk-driven data generation technique proposed in section 3.4 will result in a safer closed-loop system than one trained on data uniformly sampled throughout the state space.

Pendulum example We evaluate the inverted pendulum by measuring the mean time to failure over 100 trajectories each with 500 time steps from random initial states. We report the mean and standard error of this metric over 5 trials with 500 being the maximum attainable value. To test **H1**, we train a baseline perception system using a mean squared error loss function and a risk-driven perception system using the loss function described in section 3.3 on 10,000 uniformly sampled data points. To make the perception problem more challenging, we add noise to the images. The risk sensitive loss function improved the mean time to failure from the baseline of 234 ± 62 to 492 ± 6. To test **H2** on the inverted pendulum system, we limit the perception system to a data budget of 50 training examples. We train a baseline perception system on data sampled uniformly throughout the state space and a risk-driven perception system on data sampled according to the weighting function shown in fig. 4. The mean time to failure for the baseline system is 152 ± 89, while the time to failure for the system trained of the risk-driven data was 500 for all trials. See appendix A.3 for full results.

4 Vision-Based Detect and Avoid Application

Aircraft collision avoidance systems use information gathered from sensors to detect intruding aircraft and issue advisories for safe collision avoidance maneuvers [18]. Traditional sensors used for surveillance and tracking include ADS-B, onboard radar, and transponders [19]; however, autonomous aircraft require additional sensors both for redundancy and to replace the visual acquisition typically performed by the pilot. For this reason, vision-based traffic detection systems have been proposed, in which intruding aircraft are detected from images taken by a camera sensor mounted on the aircraft [5], [20]. In this section, we will apply our risk-driven design techniques to improve the safety of a vision-based detect and avoid (DAA) system. The code for this work can be found at https://github.com/sisl/RiskDrivenPerception.

4.1 Problem Setup

Figure 5 outlines the components of a vision-based DAA system. A camera mounted on the aircraft (which we refer to as the ownship) produces image observations of its surroundings. These images

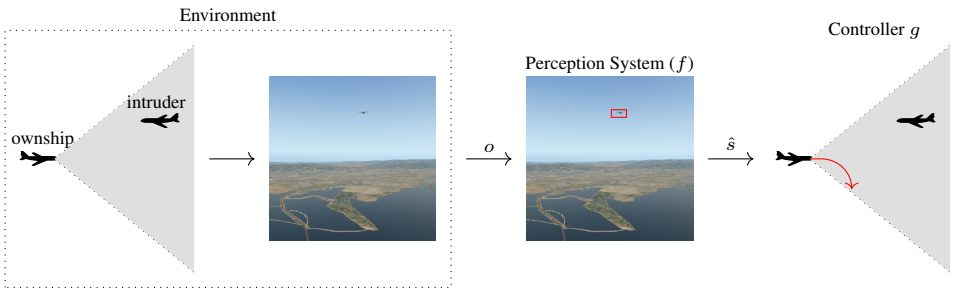


Figure 5: Overview of the vision-based detect and avoid system application.

are then passed through a perception system to produce an estimate of the state for the controller. The collision avoidance controller g, which is loosely based on the ACAS X family of collision avoidance systems [18], commands vertical maneuvers based on an estimate of the relative position and velocity of an intruder aircraft. Specifically, the state is defined as $s = [p, h, \dot{h}, a_{\text{prev}}, \tau]$ where p is a binary variable describing whether an intruder is present, h is the relative altitude of the intruder, $\dot{h}$ is the relative vertical rate of the intruder, a_{prev} is the previous action, and τ is the time to loss of horizontal separation. The controller selects an action $u \in \{\text{COC}, \text{CLIMB}, \text{DESCEND}\}$ where COC represents clear of conflict. If $p = 0$, the action is always COC.

We assume that the state estimate for the controller is produced using two steps. First, the image is passed through a perception network that is responsible for detecting other aircraft (which we refer to as intruders) and producing a bounding box. Once an intruder has been detected, the ownship interrogates it to determine the rest of the state. We focus our risk-driven design efforts on the detection component of the perception system. We train a baseline network using the YOLOv5 algorithm [21], [22] on 10,000 simulated images in which the intruder location is sampled uniformly within the ownship field of view. For additional details on the controller, network architecture, computational resources, and training, see appendix B.

4.2 Risk-Driven Design

An object detector can make two types of errors: false positives and false negatives. While false positives have an effect on the efficiency of the system, they do not affect the risk of a collision. We therefore focus on quantifying the risk of of false negatives in the presence of an intruder. We define the perception error ϵ as zero when an intruder is properly detected and one when it is missed, leading to a perceived state

$$\hat{s} = [1 - \epsilon, h, \dot{h}, a_{\text{prev}}, \tau] \tag{10}$$

The notional error model $\pi_e(\epsilon \mid s)$ is a Bernoulli distribution with a success probability based on the detection performance of the baseline model (see appendix B for more detail). To solve for $Q_\alpha^{\pi_e}(s, \epsilon)$, we discretize the state space and apply approximate dynamic programming.

Risk-sensitive loss function In addition to outputting the bounding box coordinates, the detection network outputs an objectness score $\hat{p}$ representing the model's confidence that an aircraft is present in the frame. The model detects an aircraft after checking if $\hat{p}$ exceeds a threshold; however, this check poses a numerical challenge when differentiating the risk sensitive loss since the gradient is undefined at the decision threshold and is zero everywhere else. To avoid this problem we evaluate the risk of $\hat{p}$ directly by interpolating between the risk of not detecting and detecting the intruder

$$\rho_\alpha^{\pi_e}(s, \hat{p}) = (\hat{p}) Q_\alpha^{\pi_e}(s, 0) + (1 - \hat{p}) Q_\alpha^{\pi_e}(s, 1) \tag{11}$$

The gradient of risk function in eq. (11) with respect to ϵ is equal to the risk weight, so the model will be incentivized to be more confident in states where a detection has a significant effect on overall risk.

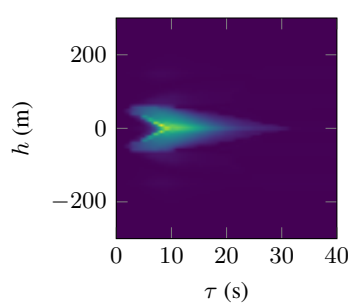


Figure 6: DAA weighting function.

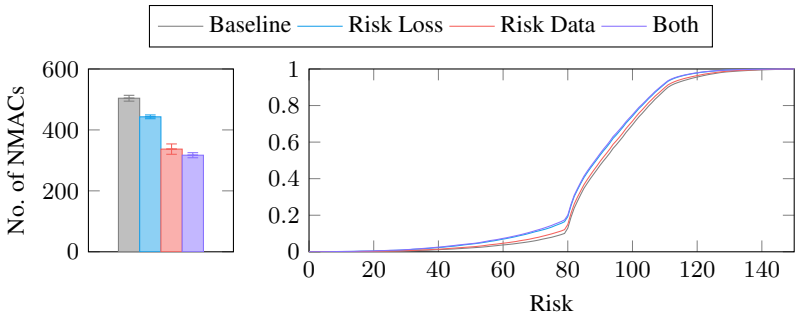


Figure 7: Number of NMACs and cumulative risk for each perception system.

Risk-driven data generation Figure 6 shows the weighting function when $\alpha = 0$ and an intruder is present. States with high weights correspond to states where not detecting the intruder is significantly more risky than detecting the intruder. While a collision is imminent in states with low values of τ and $|h|$, these states have low weights since a collision will occur regardless of whether the intruder is detected. The states with the highest risk weights are states in which there is just enough time to resolve an impending collision as long as the intruder is detected.

4.3 Evaluation

The safety of aircraft collision avoidance systems is often assessed using Monte Carlo analysis on airspace encounter models [23], [24]. Encounter models are probabilistic representations of typical aircraft behavior during a close encounter with another aircraft. To analyze the safety of a particular collision avoidance system, we can simulate the system on a set of encounters and count the number of times a near mid-air collision (NMAC) occurs. We define an NMAC as a simultaneous loss of aircraft separation to within 50 m vertically and 100 m horizontally. For the evaluation in this work, we use model consisting of 1000 pairwise encounters in which the ownship and intruder follow straight line trajectories with various relative geometries. All encounters result in an NMAC if no collision avoidance actions are taken (see appendix B.5).

We trained perception networks using four perception design techniques: no risk awareness (baseline), risk-sensitive loss function, risk-driven data generation, and both a risk-sensitive loss function and risk-driven data generation. We performed three trials for each perception design method. The first section of fig. 7 shows the number of encounters that result in an NMAC for each technique. In support of **H1**, training with the risk-sensitive loss function (blue) results in a slight increase in safety. In support of **H2**, training on data generated using the weighting function (red) results in a 33 % decrease in the number of NMACs over the baseline. With a 37 % decrease in the number of NMACs compared to the baseline, the perception systems that were developed using both risk-driven techniques (purple) are slightly safer than using either technique individually. Figure 7 also shows the cumulative distribution over the risks of the errors made by each perception system on the encounter set. This result implies that even small shifts in overall risk can have significant impacts on safety.

Figure 8 shows an example encounter that resulted in NMAC for the baseline system but was resolved by the risk-sensitive systems. The risk-sensitive system (purple) detects the intruder aircraft sooner and more often than the baseline. Moreover, the plot at the bottom of the figure shows that the risks of the perceptual errors made by the baseline system are much higher than the risks of the errors made by the risk-sensitive system.

5 Related Work

We discuss prior work in perception system evaluation, risk-aware control, and task-aware prediction.

Perception system evaluation The high-dimensional, complex nature of perception inputs makes closed-loop safety evaluation of perception systems especially challenging. This challenge was first noted in early work on computer vision, in which systems were evaluated by propagating uncertainty in a set of input images through each computer vision component [25], [26]. Similarly, a number of recent works have focused on finding adversarial perturbations that will cause misclassification

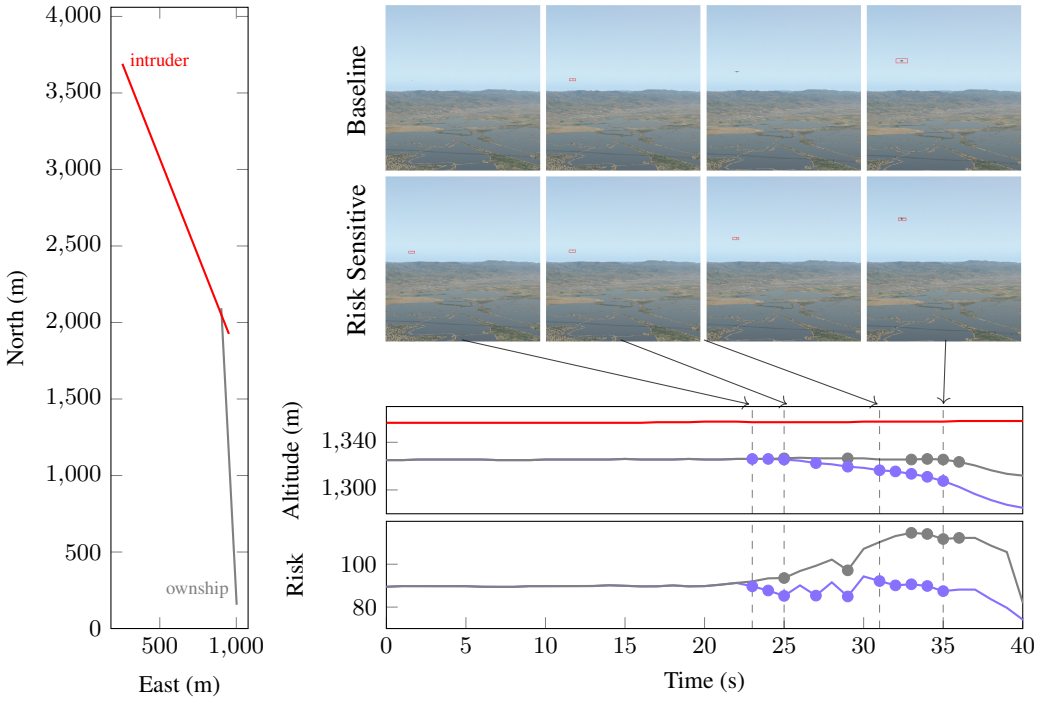


Figure 8: Example encounter that was resolved by the risk-sensitive systems but resulted in an NMAC for the baseline system. The red line denotes the intruder trajectory, while the gray and purple lines denote the ownship trajectory when using the baseline and risk-sensitive perception systems respectively. Left: overhead view of the encounter. Bottom right: altitude of each aircraft over time along with the risk of each perception error. The dotted markings indicate time steps in which the intruder was detected. Top right: perception outputs of each system at four snapshots in time.

of single input images [27]. Julian *et al.* [6], in contrast, use adaptive stress testing (AST) [28] to find sequences of adversarial perturbations that will lead a vision-based aircraft taxi system to failure rather than focusing on single images in isolation. However, this approach is only able to find failures and cannot guarantee safety if no failures are found. Katz *et al.* [29] provide approximate formal guarantees of closed-loop safety properties by using a generative model to approximate the set of plausible inputs to a perception system. Dreossi *et al.* [30] address the complexity of analyzing the closed-loop system by analyzing the controller and perception system separately before combining the results to find failures in the overall system. This work draws inspiration from this decoupled approach.

Risk-aware control In safety-critical domains, it is desirable to design controllers that perform well even in worst-case conditions. This problem has been addressed through safety-constrained MDPs [31] and shielding techniques in reinforcement learning [32]. Other risk-aware control approaches use a notion of CVaR to design controllers. CVaR MDPs, for instance, have been used to produce control policies that are robust to modeling errors and worst-case outcomes [10], [14]. Distributional reinforcement learning approaches estimate the distribution of returns [15]. Using the estimate of the return distribution, it is possible to compute risk-related metrics such as CVaR and produce risk-aware reinforcement learning policies [11], [16], [17], [33]. In this work, rather than using a notion of risk to design control policies, we use it to design safe perception systems.

Task-aware prediction Previous work has explored the benefit of using task-aware metrics to inform the design of predictive models. For example, Lambert *et al.* [34] note that more accurate dynamics models in reinforcement learning do not necessarily correlate with higher rewards, and Bansal *et al.* [35] show improved performance when learning a dynamics model that achieves the best control rather than learning the most accurate model. Multiple techniques have been proposed to augment traditional training loss functions with task-specific metrics to design models that are not only accurate

but also satisfy domain constraints [36]–[38]. In the context of trajectory prediction, McAllister *et al.* [39] propose control-aware prediction objectives, which take into account the downstream effects of the predictions on controller performance. These works motivate the need for task-aware objectives but are not used with image-based perception. In the context of perception, Greiffenhagen *et al.* [40] introduce the idea of connecting perception errors to overall system performance for the design and evaluation of a system that uses traditional computer vision [40]–[42]. Furthermore, Philion *et al.* [43] develop planner-centric metrics for an machine-learning based object detection system and show their benefit over traditional metrics used to evaluate object detectors. However, they focus only on the evaluation of perception systems, while this work focuses on their design.

6 Conclusion

In this work, we presented a methodology for risk-driven design of safety-critical perception systems. We formulated a risk function that measures the effect of perception errors on the closed-loop performance of the fully-integrated perception and control system. We then showed how to use that risk function during the design process by incorporating it into the loss function and developing a risk-driven data generation technique. We demonstrated our approach on a realistic vision-based aircraft detect and avoid system and showed that our techniques could increase safety by 37 % over a baseline system. We note that while the methods presented here focus on designing safe perception systems, they do not represent safety guarantees. Therefore, perception systems designed in this manner should still be put through additional testing and safety validation before deployment. A limitation of this work is that we assume that the perception system is Markov, which limits the applicability of our technique to perception systems that use filtering to produce state estimates. Future work will address this limitation.

Acknowledgments and Disclosure of Funding

The NASA University Leadership Initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research. This research was also supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE–1656518. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any NASA entity or the National Science Foundation. This work was also supported by a grant from the UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020-2024).

References

[1] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, p. 648, 2019.

[2] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 384–406, 2018.

[3] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[4] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020.

[5] R. Opromolla and G. Fasano, "Visual-based obstacle detection and tracking, and conflict detection for small UAS sense and avoid," *Aerospace Science and Technology*, vol. 119, p. 107 167, 2021.

[6] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[7] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[8] A. Majumdar and M. Pavone, "How should a robot assess risk? Towards an axiomatic theory of risk in robotics," in *Robotics Research*, Springer, 2020, pp. 75–84.

[9] G. C. Pflug, "Some remarks on the value-at-risk and the conditional value-at-risk," in *Probabilistic constrained optimization*, Springer, 2000, pp. 272–281.

[10] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, "Risk-sensitive and robust decision-making: A CVaR optimization approach," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[11] R. Keramati, C. Dann, A. Tamkin, and E. Brunskill, "Being optimistic to be conservative: Quickly learning a CVaR policy," in *AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 4436–4443.

[12] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[13] M. G. Bellemare, W. Dabney, and M. Rowland, *Distributional Reinforcement Learning*. MIT Press, 2022, `http://www.distributional-rl.org`.

[14] W. B. Haskell and R. Jain, "A convex analytic approach to risk-aware Markov decision processes," *SIAM Journal on Control and Optimization*, vol. 53, no. 3, pp. 1569–1598, 2015.

[15] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International Conference on Machine Learning*, 2017, pp. 449–458.

[16] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[17] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International Conference on Machine Learning*, 2018, pp. 1096–1105.

[18] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next-generation airborne collision avoidance system," Massachusetts Institute of Technology-Lincoln Laboratory, Tech. Rep., 2012.

[19] M. P. Owen, A. Panken, R. Moss, L. Alvarez, and C. Leeper, "ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS," in *Digital Avionics Systems Conference (DASC)*, IEEE, 2019, pp. 1–10.

[20] J. James, J. J. Ford, and T. L. Molloy, "Learning to detect aircraft for long-range vision-based sense-and-avoid systems," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4383–4390, 2018.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[22] G. Jocher, A. Chaurasia, A. Stoken, J. Borovec, NanoCode012, Y. Kwon, TaoXie, J. Fang, imyhxy, K. Michael, Lorna, A. V, D. Montes, J. Nadar, Laughing, tkianai, yxNONG, P. Skalski, Z. Wang, A. Hogan, C. Fati, L. Mammana, AlexWang1900, D. Patel, D. Yiwei, F. You, J. Hajek, L. Diaconu, and M. T. Minh, *ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference*, version v6.1, Feb. 2022.

[23] B. J. Chludzinski, "Evaluation of TCAS II version 7.1 using the FAA fast-time encounter generator model," Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-346, 2009.

[24] J. E. Holland, M. J. Kochenderfer, and W. A. Olson, "Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance," *Air Traffic Control Quarterly*, vol. 21, no. 3, pp. 275–297, 2013.

[25] X. Liu, T. Kanungo, and R. M. Haralick, "On the use of error propagation for statistical validation of computer vision software," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1603–1614, 2005.

[26] R. M. Haralick, "Performance characterization in computer vision," in *British Machine Vision Conference*, Springer, 1992, pp. 1–8.

[27] G. R. Machado, E. Silva, and R. R. Goldschmidt, "Adversarial machine learning in image classification: A survey toward the defender's perspective," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–38, 2021.

[28] A. Corso, R. J. Moss, M. Koren, R. Lee, and M. J. Kochenderfer, "A survey of algorithms for black-box safety validation of cyber-physical systems," *Journal of Artificial Intelligence Research*, vol. 72, no. 2005.02979, pp. 377–428, 2021.

[29] S. M. Katz, A. L. Corso, C. A. Strong, and M. J. Kochenderfer, "Verification of image-based neural network controllers using generative models," in *Digital Avionics Systems Conference (DASC)*, 2021.

[30] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[31] A. Wachi, Y. Sui, Y. Yue, and M. Ono, "Safe exploration and optimization of constrained MDPs using Gaussian processes," in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[32] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[33] Y. Chow and M. Ghavamzadeh, "Algorithms for CVaR optimization in MDPs," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014.

[34] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective mismatch in model-based reinforcement learning," in *Learning for Dynamics and Control*, PMLR, 2020, pp. 761–770.

[35] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via Bayesian optimization," in *Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 5168–5173.

[36] C. Innes and R. Ramamoorthy, "Elaborating on learned demonstrations with temporal logic specifications," in *Robotics: Science and Systems*, 2020.

[37] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. Vechev, "DL2: Training and querying neural networks with logic," in *International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Sep. 2019, pp. 1931–1941.

[38] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," in *International Workshop on the Algorithmic Foundations of Robotics*, Springer, 2020, pp. 432–449.

[39] R. McAllister, B. Wulfe, J. Mercat, L. Ellis, S. Levine, and A. Gaidon, "Control-aware prediction objectives for autonomous driving," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[40] M. Greiffenhagen, D. Comaniciu, H. Niemann, and V. Ramesh, "Design, analysis, and engineering of video monitoring systems: An approach and a case study," *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1498–1517, 2001.

[41] M. Greiffenhagen, V. Ramesh, D. Comaniciu, and H. Niemann, "Statistical modeling and performance characterization of a real-time dual camera surveillance system," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000, pp. 335–342.

[42] M. Greiffenhagen, V. Ramesh, and H. Niemann, "The systematic design and analysis cycle of a vision system: A case study in video surveillance," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.

[43] J. Philion, A. Kar, and S. Fidler, "Learning to evaluate perception models using planner-centric metrics," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 055–14 064.

[44] S. M. Katz, A. Corso, S. Chinchali, A. Elhafsi, A. Sharma, M. Pavone, and M. J. Kochenderfer, "NASA ULI aircraft taxi dataset," *Stanford Digital Repository*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] see section 6
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We note in section 6 that the approach does not provide any safety guarantees.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] All code is attached in the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix A and appendix B
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We include standard error for both experimental domains across multiple trials
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix A and appendix B
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use and cite YOLOv5 [22]

(b) Did you mention the license of the assets? [Yes] As we note in appendix B, YOLOv5 is under the GNU general public license

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include trained aircraft detection networks in the supplementary material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We did not collect any data from people. Only flight simulator data from Xplane 11.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] see previous

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No human subjects

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Inverted Pendulum Details

A.1 Inverted Pendulum Model and Controller

The inverted pendulum system is modeled using state $s = [\theta, \omega]$ where θ is the angle of the pendulum from the vertical, and ω is its angular velocity. The discrete-time dynamics for the inverted pendulum are represented as follows

$$\begin{aligned} \theta_{t+1} &= \theta_t + \omega_t \Delta t \\ \omega_{t+1} &= \omega_t - \frac{3g}{2\ell}\sin(\theta_t + \pi) + \frac{3a}{m\ell^2}\Delta t \end{aligned} \tag{12}$$

where g is the acceleration due to gravity, ℓ is the length of the pendulum, m is the mass of the pendulum, Δt is the time step, and a is the input torque from the controller. In this work, we use $g = 10\,\mathrm{m/s}$, $\ell = 1\,\mathrm{m}$, $m = 1\,\mathrm{kg}$, and $\Delta t = 0.05\,\mathrm{s}$. We clip ω such that the magnitude of the angular velocity does not exceed 8 rad/s, and we clip the control inputs so that the maximum torque magnitude does not exceed 2 N·m.

We derive a simple rule-based policy to balance the pendulum according to the following two equations

$$\begin{aligned} \omega_{\text{target}} &= \text{sign}(\theta)\sqrt{60(1-\cos(\theta)} \\ a &= -2\omega + (\omega - \omega_{\text{target}}) \end{aligned} \tag{13}$$

where the first equation determines the angular velocity required to move the pendulum from its current angle to an angle of zero. The second equation performs proportional control using this quantity and the current angular velocity. This controller is able to keep the pendulum upright under perfect perception.

A.2 Inverted Pendulum Risk Function

Error model We model the perception errors $\epsilon = [\epsilon_\theta, \epsilon_\omega]$ as additive noise distributed according to a multivariate Gaussian distribution with zero mean and diagonal covariance. Noting that predicting ω from successive frames tends to be more difficult for a perception network than prediction θ according to our baseline model, we set the covariance as

$$\Sigma = \begin{bmatrix} 0.2 & 0.0 \\ 0.0 & 0.5 \end{bmatrix} \tag{14}$$

To solve for the risk function using distributional dynamics programming, we discretize the error model and approximate it as a categorical distribution. We select 11 discrete points for ϵ_θ on a logarithmic scale between –0.4 and 0.4 (two standard deviations) such that the points are more highly concentrated around zero error. We determine their corresponding weights by computing their corresponding probability densities from the continuous Gaussian model and normalizing.

State discretization We discretize $\theta \in [-\pi/4, \pi/4]$ and $\omega \in [-2, 2]$ using 41 points with a higher density of points around zero. To solve for the risk function, we assume an episodic setting in which each episode has 20 time steps. To reflect the episodic nature in the solving process, we add a time component to the state such that $s = [t, \theta, \omega]$ that specifies the number of steps left in the episode. For the next state in the dynamics, t is decremented by one.

Cost function The cost function for the inverted pendulum problem is defined as the absolute angle of the pendulum at the last time step in each episode:

$$c(s, \epsilon) = \begin{cases} |\theta|, & \text{if } t = 0 \\ 0, & \text{if } t > 0 \end{cases} \tag{15}$$

Since we specify that the pendulum fails if $|\theta| > \pi/4$, the cost is bounded between 0 and $\pi/4$. We select 50 discrete cost points on a logarithmic scale within this range for the distributional dynamic programming.

Solving for the cost distributions using dynamic programming takes under a minute on a single Intel Core i7 processor operating at 4.20 GHz. We save the distributions and use linear interpolation and

eq. (5) to evaluate the risk at an arbitrary ϵ. Since the autodiff package we use for risk-sensitive training does not have a straightforward way to differentiate through an interpolation of tabular data, we train a neural network surrogate model of the risk function to use in the risk-sensitive loss. We train a small MLP network with one hidden layer containing 64 hidden units for each value of α.

Perception system training All perception systems trained for the pendulum problem were neural networks with two hidden layers with ReLU activations and 64 units each. The final layer uses a hyperbolic tangent activation (the labels are scaled to range between −1 and 1). We train using the ADAM optimizer with a learning rate of 1×10^{-3}. We train for 200 epochs for the risk data experiment and 400 epochs for the risk loss experiment.

A.3 Inverted Pendulum Results

Table 1 shows the full results for the inverted pendulum experiments. We ran five trials for each experiment, and we report the mean and standard error of the time to failure for each experiment. Since we simulate 100 episodes of length 500 for each evaluation the highest possible mean time to failure is 500, which would indicate that the pendulum stayed upright for all time steps. The risk-sensitive design indicates a benefit for most values of α; however, we note that performance seems to degrade when α becomes too large, which is likely due to being overly pessimistic. If we always assume the worst-case outcomes, there may not be any states in which a failure can be prevented. In this case, the pendulum system will not be incentivized to produce accurate estimates at any state.

Table 1: Mean time to failure for pendulum perception systems.

	Risk Loss	Risk Data
Baseline	234 ± 62	152 ± 89
Risk Sensitive ($\alpha = 0.00$)	$\mathbf{492 \pm 6}$	334 ± 102
Risk Sensitive ($\alpha = 0.20$)	473 ± 18	$\mathbf{500 \pm 0}$
Risk Sensitive ($\alpha = 0.50$)	486 ± 12	404 ± 86
Risk Sensitive ($\alpha = 0.80$)	489 ± 8	438 ± 59
Risk Sensitive ($\alpha = 0.99$)	391 ± 59	273 ± 65

B Vision-Based Detect and Avoid Details

B.1 Collision Avoidance Controller

The collision avoidance controller takes in a state $s = [p, h, \dot{h}, a_{\text{prev}}, \tau]$ where p is a binary variable describing whether an intruder is present, h is the relative altitude of the intruder, $\dot{h}$ is the relative vertical rate of the intruder, a_{prev} is the previous action, and τ is the time to loss of horizontal separation. The relative states, h and $\dot{h}$, defined to be the ownship quantity minus the intruder quantity. For example, if the intruder is below the ownship, h will be positive. The previous action is contained within the state to penalize undesirable operational characteristics such as reversals in collision avoidance advisory. Finally, the variable τ is used to summarize the horizontal evolution of the encounter in a single variable. The goal is to have sufficient vertical separation h as τ gets close to zero to avoid a near mid-air collision (NMAC).

The controller outputs an action $u \in \{\text{COC}, \text{CLIMB}, \text{DESCEND}\}$ where COC represents clear of conflict and the CLIMB and DESCEND actions commands vertical rates. Specifically, the CLIMB action commands a vertical rate of 8 m/s, while the DESCEND action commands a vertical rate of −8 m/s. For COC, we let $u = 0$. Given a state s_t and action u_t, we calculate the next state as follows

$$\begin{aligned} p_{t+1} &= p_t \\ h_{t+1} &= h_t + \dot{h}_t \\ \dot{h}_{t+1} &= \dot{h}_t + u_t + w \\ a_{\text{prev},t+1} &= u_t \\ \tau_{t+1} &= \tau_t - 1 \end{aligned} \tag{16}$$

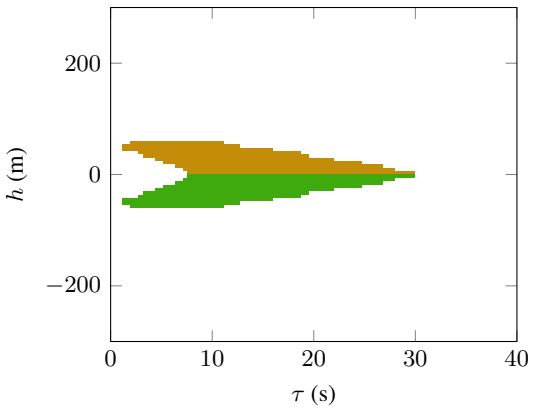


Figure 9: Policy for the collision avoidance controller for a slice of the state space where $\dot{h} = 0$ and $a_{\text{prev}} = \text{COC}$. Orange regions indicate regions of the state space where a CLIMB advisory is issued, green regions indicate DESCEND, and white areas are COC.

where w represents potential noise in vertical rate due to factors such as pilot response delays. We set this noise to a categorical distribution in which $w = 0$ with 0.8 probability and $w = \pm 0.5$ with probability 0.1 each. We also provide a maximum acceleration to the model and clip u_t based on the current vertical rate to comply with this maximum acceleration.

We develop a controller that is loosely inspired by the ACAS X family of collision avoidance problems by formulating the control problem as an MDP and solving for the optimal policy using dynamic programming [18]. The solving process takes under a minute on a single Intel Core i7 processor operating at 4.20 GHz. Figure 9 shows the resulting policy over a slice of the state space. As expected, the alerting region shrinks with increasing τ since there is more time to resolve collisions. No action is taken in the region near $h = 0$ and $\tau = 0$. This region represents states in which a collision is imminent and taking any action to avoid collision would be futile.

B.2 Baseline DAA Perception Network

To train a baseline detection network, we gathered images of intruder aircraft labeled with their corresponding bounding boxes using the X-Plane 11 flight simulator, which has been used in previous work to gather data for a vision-based aircraft taxi scenario [44]. For each data point, we first sample a random ownship position and orientation in the airspace around the Palo Alto airport (PAO). We then sample an intruder position and orientation uniformly within the field of view and record the resulting image. Finally, we compute the corresponding bounding box label based on the relative position of the intruder aircraft. We train the detection networks using the YOLOv5 algorithm [21], [22] based on the open source (GNU general public license) PyTorch implementation at https://github.com/ultralytics/yolov5. For each experiment, we use the YOLOv5s model architecture, which has 7.2 million trainable parameters and train on a dataset of 10,000 images for 200 epochs with a batch size of 16 using the default hyperparameters. A single training run takes ≈ 11 hours on a single Intel Core i7 processor operating at 4.20 GHz and adding the risk-sensitive loss component did not significantly increase the required training time.

B.3 DAA Risk Function

Error model We model the perception error for the DAA model as $\epsilon = 0$ when the intruder aircraft is detected and $\epsilon = 1$ when it is not detected. The notional error model $\pi_e(\epsilon \mid s)$ is a Bernoulli distribution with a success probability corresponding to the probability of detecting an intruder when in state s. We determined the parameters of the error distributions by sampling 10,000 images at states uniformly distributed throughout the state space and feeding them through the baseline (no risk awareness) perception system to determine whether the intruder is detected.

Using the results as training data, we train a small neural network with a single hidden layer of 10 hidden and a sigmoid output to represent $\pi_e(\epsilon \mid s)$. We only input part of the state to the neural

network. In particular, since $\dot{h}$, and a_{prev} have no influence on the image observation and we only consider cases where $p = 1$, we omit those variables at inputs to the error model. Furthermore, noting that the error model should be roughly symmetric about $h = 0$, we input $|h|$ rather h. In summary, the network takes as input values for $|h|$ and τ and outputs a probability of detecting the intruder. Figure 10 shows the resulting model outputs over the state space. The cone shape is due to some states lying outside the ownship field of view.

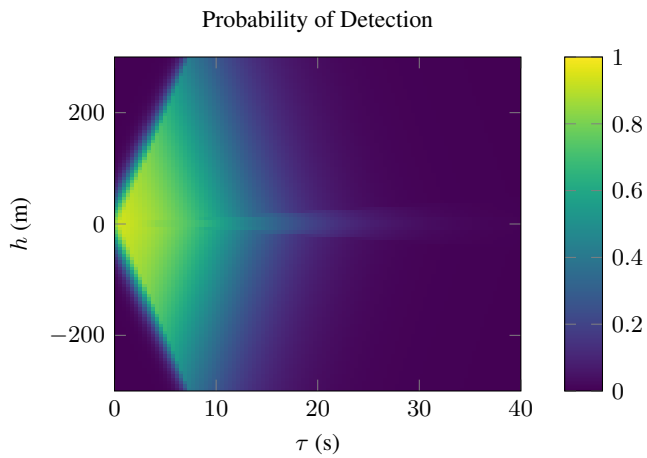


Figure 10: Perception error model for the vision-based DAA application based on the baseline model.

State discretization For the distributional dynamic programming, we discretize $h \in [-300, 300]$ and $\dot{h} \in [-10, 10]$ using 41 and 21 points respectively with a higher density of points around zero. We discretize $\tau \in [0, 41]$ with a uniform step size of 1. The remaining variables are already discrete.

Cost function We set the cost function to reflect the separation of the aircraft when $\tau = 0$. Since we want high value of cost to correspond to low separation, we set the cost function as

$$c(s, \epsilon) = \begin{cases} 150 - |h|, & \text{if } \tau = 0 \\ 0, & \text{if } \tau > 0 \end{cases} \tag{17}$$

We discretize the cost uniformly between 0 and 150 using 50 points for the distributional dynamic programming.

With these modeling assumptions, we use distributional dynamic programming to solve for $Z^{\pi}(s, \epsilon)$. Solving for the cost distributions takes under 10 seconds on a single Intel Core i7 processor operating at 4.20 GHz. As noted previously, $\dot{h}$, and a_{prev} have no effect on the observed image at a given state, so we marginalize over these variables before calculating the risk function. We compute weights for each discrete value of $\dot{h}$ and a_{prev} from simulations of the controller with perfect perception. We then take the weighted sum of the $Z^{\pi}(s, a)$ using these weights to obtain a cost distribution function that is only a function of h, τ, and ϵ. Using this result, we can compute the risk function.

B.4 DAA Risk-Driven Data Generation

To generate data according the samples from the weighting function shown in fig. 6, we must map a particular h and τ into a corresponding image observation. We first sample an ownship position and orientation uniformly in the airspace around the Palo Alto airport (PAO). Next, we select the intruder altitude based on the current altitude of the ownship and the current value for h. The remaining position variables are selected based on τ; however, there is not a 1-to-1 mapping due to varying aircraft speeds and relative headings. Therefore, we sample values for the ownship speed, intruder speed, and relative heading and use these values to calculate τ. Specifically, we sample speeds uniformly between 45 m/s and 55 m/s and relative headings between 120° and 240°.

B.5 Encounter Model

Sampling encounters For simplicity, the encounters in this work are modeled as straight-line trajectories in which the ownship and intruder follow constant horizontal speeds. We sample an encounter by first sampling a set of encounter features according to uniform distributions over the ranges shown in table 2 and then using these features to generate trajectories for the ownship and intruder aircraft. The horizontal and vertical miss distance parameters indicate the horizontal range

Table 2: Encounter model parameters.

Parameter	**Min**	**Max**	**Unit**
Ownship Horizontal Speed	45	55	m/s
Intruder Horizontal Speed	45	55	m/s
Horizontal Miss Distance	0	100	meters
Vertical Miss Distance	−50	50	meters
Relative Heading	120	240	degrees

and relative altitude of the ownship and intruder aircraft at closest point of approach. We select these distances such that all encounters will results in an NMAC if no collision avoidance action is taken. We simulate encounters that are 50 s long with the closest point of approach occurring 40 s into the encounter. The range of relative headings is set such that the encounters are close to head-on, and the intruder should almost always be within the ownship field of view (although at the beginning of the encounter it may be too small to be detected by the camera). The features in table 2 fully determine the relative trajectories of the ownship and intruder. Once the relative trajectories are generated, we rotate and shift both trajectories to put them in random positions around the Palo Alto airport.

Simulating encounters When we simulate the encounters, we add the perception system and ownship controller into the loop. At each time step, we read the horizontal components of the next state from the encounter. For the vertical components, we assume noisy vertical rate centered around zero until the ownship receives an alert, at which point it accelerates to the vertical rate commanded by the collision avoidance advisory.

B.6 Additional DAA Results

In addition to simulating each vision-based DAA perception system on the encounter set, we also evaluated the precision and recall of each system on a validation set of 1000 images with uniformly sampled intruder positions within the ownship field of view. Table 3 shows the results. We note that

Table 3: Additional DAA results.

	Precision	Recall
Baseline	0.78 ± 0.04	0.38 ± 0.03
Risk Loss	0.82 ± 0.03	0.37 ± 0.02
Risk Data	0.77 ± 0.01	0.39 ± 0.01
Both	0.79 ± 0.01	0.39 ± 0.01

one could improve safety of the overall system by simply biasing the perception system towards outputting a detection more often; however, this result would likely increase false positives and result in worse operational efficiency. To ensure that this effect is not causing the safety benefits we show in the risk-driven systems in fig. 7, we can analyze the effect of risk-driven training on the precision of the overall system. From these results, we determine that the risk-driven design techniques did not harm overall precision and focused their efforts only on risky states.