Reducing Conservativeness in Safety Guarantees by Learning Disturbances Online: Iterated Guaranteed Safe Online Learning

Jeremy H. Gillula
Computer Science Dept., Stanford University
jgillula@cs.stanford.edu

Claire J. Tomlin
Electrical Engineering and Computer Sciences Dept., UC Berkeley
tomlin@eecs.berkeley.edu

***Abstract*—Reinforcement learning has proven itself to be a powerful technique in robotics, however it has not often been employed to learn a controller in a hardware-in-the-loop environment due to the fact that spurious training data could cause a robot to take an unsafe (and potentially catastrophic) action. One approach to overcoming this limitation is known as Guaranteed Safe Online Learning via Reachability (GSOLR), in which the controller being learned is wrapped inside another controller based on reachability analysis that seeks to guarantee safety against worst-case disturbances. This paper proposes a novel improvement to GSOLR which we call Iterated Guaranteed Safe Online Learning via Reachability (IGSOLR), in which the worst-case disturbances are modeled in a state-dependent manner (either parametrically or nonparametrically), this model is learned online, and the safe sets are periodically recomputed (in parallel with whatever machine learning is being run online to learn how to control the system). As a result the safety of the system automatically becomes neither too liberal nor too conservative, depending only on the actual system behavior. This allows the machine learning algorithm running in parallel the widest possible latitude in performing its task while still guaranteeing system safety. In addition to explaining IGSOLR, we show how it was used in a real-world example, namely that of safely learning an altitude controller for a quadrotor helicopter. The resulting controller, which was learned via hardware-in-the-loop reinforcement learning, out-performs our original hand-tuned controller while still maintaining safety. To our knowledge, this is the first example in the robotics literature of an algorithm in which worst-case disturbances are learned online in order to guarantee system safety.**

I. Introduction

Reinforcement Learning (RL) is a branch of machine learning in which an agent attempts to learn which actions to take in order to maximize a reward. RL has proven itself to be a powerful technique in control applications for robotics, with successes ranging from flying complex maneuvers on an autonomous helicopter [2] to teaching a quadruped robot to jump up steps [8], to helping a robot car navigate at high speeds in an unstructured environment [9]. Despite this, RL has not often been employed to learn a controller in a hardware-in-the-loop environment, especially in situations where an unstable or poorly tuned controller could cause injury to humans or damage expensive hardware [13]. In fact, most applications of RL in robotics have been situations in which reference trajectories or cost functions were learned offline (either based on recorded data or a model) and were then used with standard control techniques to control a robot. One reason for this, as described by Roberts et al. [13], is that as they are being learned, natural parameterizations of controllers can behave poorly on systems near the edge of stability. As a result spurious training data could cause a robot to take an unsafe (or even catastrophic) action. Thus despite their successes online RL algorithms are limited to being used in scenarios where safety is not critical, or where a large number of trials can be conducted beforehand in a controlled environment to guarantee system safety.

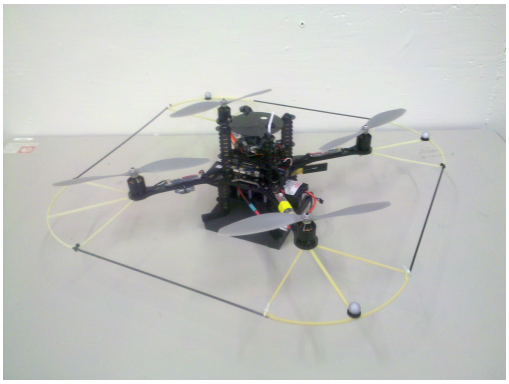

Fig. 1. The quadrotor helicopter on which IGSOLR was demonstrated.

One approach to overcoming this limitation is to wrap the controller being learned inside another controller that seeks to guarantee safety; this is the approach taken by Gillula and Tomlin [5], in which we proposed a framework known as Guaranteed Safe Online Learning via Reachability (GSOLR). This framework combines machine learning techniques with Hamilton-Jacobi-Isaacs (HJI) reachability analysis in a way that prevents the system being controlled from taking an unsafe action, while avoiding any changes to the machine learning algorithm being used [5]. In essence, GSOLR is a least-restrictive safe controller: it allows the machine learning algorithm to control the system (e.g. via RL) except when the state is detected as being near a safety threshold, at which time a safety-guaranteeing control (dictated by HJI reachability analysis) is used instead.

One of the limitations of GSOLR (as well as most HJI

reachability analysis) is that the computations for guaranteeing safety are assumed to have been done offline, prior to the online learning phase. As a result the system is unable to incorporate any new information it may learn about the disturbances while it is running; essentially its notion of safety cannot change based on what it experiences. Additionally, although not explicitly stated in [5], it is assumed that the worst-case disturbance is fixed over the entire state space, preventing the system from taking advantage of the fact that the range of disturbance values may be state-dependent and thus potentially causing the safety guarantees to be too conservative.

This paper overcomes these limitations by proposing an adapted form of GSOLR known as Iterated Guaranteed Safe Online Learning via Reachability (IGSOLR). In IGSOLR the worst-case disturbances are modeled in a state-dependent manner (either parametrically or nonparametrically) and these models are learned online, causing the resulting safe sets to be periodically recomputed (in parallel with whatever machine learning is being run online to learn how to control the system). As a result the safety of the system automatically becomes neither too liberal nor too conservative, depending only on the actual system behavior. This allows the machine learning algorithm running in parallel the widest possible latitude in performing its task while still guaranteeing system safety. The proposed algorithm is demonstrated on the example of quadrotor altitude control. The resulting controller, which was learned via hardware-in-the-loop RL on the quadrotor pictured in Figure 1, out-performs the initial hand-tuned controller while still maintaining safety.

Of course many other methods for guaranteeing system safety while improving performance online exist in the robotics literature. The work of Aswani et al. [3] makes use of a Model Predictive Control (MPC) framework in which an a priori system model is used to verify that the constraints in the MPC optimization problem are met (safety) while a learned model is used to evaluate the MPC cost function (performance). The safety guarantees generated by this method are limited to systems with linear dynamics, however. Expanding beyond linear systems, Steinhardt and Tedrake [14] have developed a technique for bounding the probability of failure over a finite time for stochastic, nonlinear systems using exponential barrier functions. In a related manner, Perkins and Barto [12] have successfully used Lyapunov design methods to guarantee safety for high-level controllers which learn how to switch between a number of base-level controllers. However the work that is most related to GSOLR (and thus IGSOLR) is that of Ng and Kim [11], in which they propose an algorithm for linear dynamical systems which "monitors" controllers suggested by a learning algorithm and rejects those that would lead to instability. Although this is just a brief overview of related work, to our knowledge IGSOLR is the first example in the robotics literature of an algorithm in which disturbances are learned online in order to automatically tune system safety.

The rest of this paper is organized as follows. Section II briefly reviews GSOLR and then elaborates on the limitations described above before describing IGSOLR in detail. Section III describes how IGSOLR was applied to the problem of learning an altitude controller for a quadrotor helicopter online. Section IV describes the experimental platform and illustrates the results obtained from running IGSOLR on the quadrotor. Finally Section V concludes the paper and elaborates on open questions and future work.

II. Iterated Guaranteed Safe Online Learning via Reachability

Before explaining IGSOLR, we begin with a review of GSOLR. Due to space constraints, the review of GSOLR here (and HJI reachability analysis in particular) is very brief and omits a great deal of detail. For a more in-depth explanation of GSOLR, as well as a discussion of its strengths and weaknesses, we refer the reader to Gillula and Tomlin [6]. For an easily understandable overview of HJI reachability, we refer the reader to Ding et al. [4].

A. GSOLR

Guaranteed Safe Online Learning via Reachability seeks to ensure system safety using HJI reachability analysis while simultaneously improving system performance online using machine learning.

1) Safety via HJI Reachability Analysis: HJI reachability analysis [10] assumes a system model of the form $\dot{x} = f(x, u, d)$, where $x \in \mathbb{R}^n$ is the system state, u is the control input to the system, and d is the disturbance to the system, with u and d bounded to be in some sets $\mathcal{U}$ and $\mathcal{D}$, respectively. d can be used to represent a wide variety of disturbances, and typically is used to account for noise in the system (both in estimation and control), unmodeled system dynamics, and the potentially stochastic nature of the underlying system, as long all of these quantities are bounded.

HJI reachability analysis assumes safety can be encoded by a keep-out set $\mathcal{K} \subset \mathbb{R}^n$, i.e. the system is safe if $x \notin \mathcal{K}$. By solving the modified HJI partial differential equation

$$\frac{\partial J(x,t)}{\partial t} = \min\{0, \max_{u \in \mathcal{U}} \min_{d \in \mathcal{D}} \frac{\partial J(x,t)}{\partial x} f(x,u,d)\} \quad (1)$$

one can calculate the unsafe backwards reachable set $Pre_\tau(\mathcal{K})$ (given by the zero sub-level set of $J(x,\tau)$), which is the set of all states x such that for all possible control strategies $u(\cdot)$, there exists a disturbance strategy $d(\cdot)$ such that $x \in \mathcal{K}$ at some point within τ units of time. Thus as long as we guarantee that x always remains outside $Pre_\tau(\mathcal{K})$, we can guarantee that there will exist control actions u that will allow the system to stay safe for the next τ units of time. In particular, the maximization in Equation 1 provides the optimal control u^* which the system must take when the state is on the border of $Pre_\tau(\mathcal{K})$ in order to maintain safety.

2) Combining HJI Reachability with Machine Learning: Given this unsafe set $Pre_\tau(\mathcal{K})$ and the optimal control inputs u^* to take to maintain safety, GSOLR is simply a least restrictive safe controller [5] of the form:

1) When the state x is on the border of $Pre_\tau(\mathcal{K})$, execute the optimal control action u^*.

2) Otherwise, perform whatever control action the given machine learning algorithm dictates.

The resulting controller is guaranteed to be safe and is also able to accommodate a wide variety of machine learning algorithms, including online RL algorithms.

B. Limitations of GSOLR

As mentioned in the introduction, GSOLR suffers from two weaknesses which were not described in Gillula and Tomlin [5]. The first limitation (which actually applies to most HJI reachability analysis in the literature) is that GSOLR assumes the set $\mathcal{D}$ describing the disturbance does not vary over the state space. By not accounting for the fact that the disturbance might be smaller in some areas of the state space than others, and instead using a single worst-case range over the entire state space, the generated unsafe sets can be overly conservative. There are a variety of applications in which it makes sense to model the disturbance in a state-dependent manner, including autonomous aircraft flight, where different wind disturbances could be expected based on weather; ground vehicle control, in which different terrain types (e.g. pavement, mud, etc.) could lead to different types of disturbances; robotic arm control, where motor noise may be different depending on the load on the arm or the angle of its joints; and of course quadrotor altitude control, which will be explained in more detail in Section III-A.

The second weakness of GSOLR is that it does not learn new information online to update its models of the system and disturbances. Instead, GSOLR uses a set of worst-case values for the disturbances which are chosen a priori by the system designer. If these values do not match reality, however, then the resulting safety guarantees could be either too conservative (if they are too large) in which case the system would never explore parts of its state space that are in fact safe, or too liberal (if they are too small) in which case the system may venture into parts of the state space that are in fact unsafe. Both cases can have negative consequences, either in terms of limiting the operating envelope of the system, or in risking system safety.

Of course these two limitations are intertwined. When doing HJI reachability analysis it is sometimes improper to try to model the disturbance as varying with the system state. For example, in the canonical collision avoidance scenario [10] it would not make sense to vary the pursuer's possible range of inputs based on the state of the system. Fortunately the sorts of situations in which it does make sense to model the disturbance as varying over the state space lend themselves to the idea of learning the disturbance model, which is precisely what IGSOLR seeks to do in a simple and straightforward manner.

C. IGSOLR

The first major difference between GSOLR and IGSOLR is that IGSOLR employs a model of the worst-case disturbances that varies over the state space. More specifically, we replace $d \in \mathcal{D}$ in the Hamiltonian with $d \in \mathcal{D}(x)$; the rest of the machinery for calculating the unsafe set $Pre_\tau(\mathcal{K})$ remains unchanged. We purposely do not specify what form $\mathcal{D}(x)$ should take (or even whether it should be parametric or nonparametric) in order to allow IGSOLR to apply to as broad a range of systems as possible.

The second major difference between GSOLR and IGSOLR is that a model of the system $\hat{f}(x, u, d)$ as well as $\mathcal{D}(x)$ is learned as the system runs. This model is initialized to be conservative (in order to guarantee safety) and is then updated based on data measured by the system (once a sufficiently large number of measurements have been taken to ensure that the probability of any spurious data points adversely biasing the model is sufficiently low). Periodically the most recently learned model is then used to recalculate the unsafe set $Pre_\tau(\mathcal{K})$, which is then used for future safety checks as described in Section II-A2. As mentioned in Gillula and Tomlin [6], one of the weaknesses of GSOLR is that it is limited to systems with low state dimension due to the curse of dimensionality present in the HJI reachability calculations. Unfortunately IGSOLR suffers even more from this limitation since the HJI reachable set calculations must be done in a online manner. Thus until faster methods for computing reachable sets are developed, IGSOLR is limited to systems of state dimension two or three.

It is important to note that because IGSOLR learns the disturbances in an online manner the safety guarantees that are generated will only be as good as the data on which they are based. As a result IGSOLR cannot be thought of as giving a "true" worst-case safety guarantee, since there is always the possibility that a low-probability worst-case event could occur. However, it is worth noting that even with "true" worst-case safety guarantees (i.e. those generated by a method like GSOLR) the guarantee is only as good as the a priori choice of values for the worst-case disturbances. In particular, even a "true" worst-case method is susceptible to very low probability disturbances that the system designer did not take into account (i.e. an earthquake occurring while your autonomous vehicle is driving, or a stray cosmic ray flipping a bit in your quadrotor's RAM [16], or other various "acts of God"). In this sense, no safety guarantee can ever truly be worst-case. Instead, the point of IGSOLR is to generate an approximate worst-case safety guarantee that is neither too liberal nor too conservative, based on a best-effort automatic analysis of the data. In this light the reduction in conservativeness of the safety guarantees as IGSOLR learns the disturbances should not be viewed as diminishing the legitimacy of its safety guarantees, but instead strengthening them by basing them on the data gathered so as to bring them more in line with the actual system.

III. Application to Quadrotor Altitude Control

A. Why Use IGSOLR?

Because IGSOLR is such a broad framework, it is helpful to illustrate how it works with an example. To that end we will show how it was used to learn an altitude controller for a quadrotor vehicle. Quadrotor altitude control is ideally suited to IGSOLR for a multitude of reasons, including:

1) As described by Waslander et al. [15] classical linear control techniques are unable to provide a high degree of performance for altitude control due to the complex airflow interactions present in a quadrotor system. Thus RL is a strong candidate for learning a controller.
2) However, online RL would be unsafe to try since learning with hardware-in-the-loop could be dangerous given that the hardware could be damaged by an unstable controller. Thus GSOLR is a strong candidate for ensuring safety while an online RL algorithm runs.
3) We have good reason to believe that the range of disturbances experienced by a quadrotor indoors is likely to be dependent on its state: near the ground the vehicle experiences ground effect, in which air forced through the propellers interacts with the ground, and at certain velocity ranges the vehicle may enter vortex ring state or windmill braking state [7], causing unmodeled changes in the dynamics. Since modeling the disturbance as a function of the state makes sense, we should use IGSOLR.[1]

To demonstrate IGSOLR, we implemented the algorithms described below on board an Ascending Technologies Pelican quadrotor (pictured in Figure 1) [1]. The vehicle was flown indoors in a lab environment equipped with a VICON MX motion-capture system, which provided sub-centimeter accuracy position measurements at a rate of 120 Hz. To add additional disturbances (since ground effect proved to be too easy for the IGSOLR algorithm to compensate for) an industrial-strength fan was positioned to blow upwards at the quadrotor between approximately 0.2 m and 0.8 m above the ground, with average wind gust speeds of around 50 kph.

We now describe the implementation details of IGSOLR for the quadrotor system. Since IGSOLR treats safety and performance in a parallel manner we will present these two aspects of the problem setup separately.

B. Safety via Reachability

We model the quadrotor's altitude dynamics as

$$\ddot{x} = g + \gamma u + d \tag{2}$$

where x is the quadrotor's altitude and $\mathbf{x} = [\ x \quad \dot{x}\ ]^\top$ is its complete state; g is some constant term assumed to be related to gravity and the vehicle's mass; $u \in \mathcal{U}$ is the control input to the vehicle's rotors, measured in counts (the abstract unit used by the low-level interface to our quadrotor); γ is a thrust multiplier that converts from counts to m/s^2; and $d \in \mathcal{D}(\mathbf{x})$ is a disturbance, due to any or all of the effects mentioned in the previous section. In our algorithm, we will attempt to learn g, γ, and $\mathcal{D}(\mathbf{x})$. (We assume that $\mathcal{U}$ is known, since the quadrotor's low-level interface automatically clips the input to a certain range of counts.) Because the system executes its control and measures its state at discrete intervals according to a fixed frequency, however, we must first re-write the dynamics as

$$(\dot{x}_{i+1} - \dot{x}_i)/\Delta t = g + \gamma u_i + d \tag{3}$$

where Δt is the discrete time step.

In order to learn the model parameters we begin with an initial estimate for g based on gravity and the quadrotor's measured mass, γ based off of the specifications of the quadrotor, and $\mathcal{D}(\mathbf{x})$ is initialized to a conservative constant value over the entire state space. Then, as the quadrotor is flown under the IGSOLR controller described below, traces of state data, $\mathbf{x}_0, \ldots, \mathbf{x}_{r+1}$, and input data, $u_o, \ldots, u_r$ are recorded (where r is the horizon over which we periodically recompute the unsafe set). We then use linear regression based on this data to compute estimates $\hat{g}$ and $\hat{\gamma}$ by writing $\hat{\beta} = \left(X^\top X\right)^{-1} X^\top Y$, where

$$Y = \begin{bmatrix} \dot{x}_1 - \dot{x}_0 \\ \vdots \\ \dot{x}_{r+1} - \dot{x}_r \end{bmatrix},\ X = \begin{bmatrix} 1 & u_0 \\ \vdots & \vdots \\ 1 & u_r \end{bmatrix},\ \hat{\beta} = \Delta t \begin{bmatrix} \hat{g} \\ \hat{\gamma} \end{bmatrix}. \tag{4}$$

To determine $\mathcal{D}(\mathbf{x})$ we first compute the residuals $[\ \hat{\varepsilon}_0 \quad \ldots \quad \hat{\varepsilon}_r\ ]^\top = Y - X\hat{\beta}$. We then divide the state space into a regularly spaced grid based on x and $\dot{x}$. For each grid cell $c(i,j) = [x_i, x_{i+1}) \times [\dot{x}_j, \dot{x}_{j+1})$, we calculate the mean $\mu_{i,j}$ and standard deviation $\sigma_{i,j}$ of the residuals $\hat{\varepsilon}_i$ that were generated by state data from that cell. Then we model

$$\mathcal{D}(\mathbf{x} \in c(i,j)) = [-3\sigma_{i,j} + \mu_{i,j}, \mu_{i,j} + 3\sigma_{i,j}] \tag{5}$$

for grid cells for which state data was measured.

In essence this approximation ignores the 0.3% of disturbances which will be larger than three standard deviations from the mean. While such an approximation is unfortunately necessary (as described in Section II-C), we feel it is a valid one. In particular, it is important to note that this does not mean that our safety guarantees are only valid 99.7% of the time: for the safety guarantee to be invalid it would be necessary to encounter a continuous series of these low-probability disturbances for τ straight seconds, or in our discrete case $\tau/\Delta t$ time steps. Assuming the disturbances are indeed Gaussian and IID, and given our particular values for τ and Δt, our safety guarantees have only a $5.2 \times 10^{-53}\%$ probability of not holding.

Since the quadrotor's trajectory will not necessarily cover every grid cell in the state space, we must find a way to generalize the estimate of the disturbance from areas of the state space where measurements have been taken to areas where no measurements have yet been taken. One way to generalize would be to continue to use the a priori conservative estimate of the disturbance in grid cells in which no data has been measured. However since a strong (conservative) disturbance just outside the initial safe set could keep that set from growing, this method could prevent the quadrotor from ever expanding its initial safe set and thus discovering the true nature of the disturbances in nearby states.

To avoid this problem we make the assumption that the variation in the range of disturbances over the state space is

[1]Fortunately restricting our focus to altitude control means that the state space is two-dimensional, so it is reasonable to believe we will be able to recompute new unsafe sets based on learned data in a semi-online fashion. Thus it is feasible to use IGSOLR.

smooth. In essence this smoothness assumption means that we expect disturbances in neighboring cells to be similar in range. This assumption not only appears to be valid in practice for this particular application, but makes sense for many other applications including many of the ones mentioned in Section II-B. Thus once we have estimates of $\mathcal{D}(\mathbf{x})$ for cells in which measurements were taken, we set estimates for the cells in which no measurements were taken based on their distance from cells with measurements, increasing the possible disturbance range the further a cell is from a cell with data. More precisely, for a cell $c(k,l)$ in which no measurements have been taken, we set

$$\mathcal{D}(\mathbf{x} \in c(k,l)) = \delta\, \mathcal{D}(\mathbf{x} \in c(i,j))\, ||c(i,j) - c(k,l)||_2 \quad (6)$$

where $||c(i,j) - c(k,l)||_2$ is the Euclidean distance in the state space between cells $c(i,j)$ and $c(k,l)$ (i.e. $\sqrt{(x_i - x_k)^2 + (\dot{x}_j - \dot{x}_l)^2}$), $c(i,j)$ is the cell closest to cell $c(k,l)$ (in the Euclidean sense) in which measurements have been taken, $\mathcal{D}(\mathbf{x} \in c(i,j))$ is the disturbance range in that cell, and δ is a tuning parameter that adjusts how quickly we expect the disturbances to grow (in essence modeling how quickly we expect the disturbance estimates to change over the state space).

Finally given the full model of the disturbance, we can compute the unsafe set $Pre_\tau(\mathcal{K})$ and the optimal safe control actions u^*, which can then be used in future iterations of the IGSOLR controller.

In summary the algorithm for learning the worst-case disturbances online is:

1) Take $r+1$ measurements of the state $\mathbf{x}$ and r measurements of the input u.
2) Use linear regression to generate an estimate of the model of the system dynamics, given by $\hat{g}$ and $\hat{\gamma}$.
3) Use $\hat{g}$, $\hat{\gamma}$, and the measured data to generate the residuals $\hat{\varepsilon}_i$, and then use the means and standard deviations of the residuals to estimate the disturbance $\mathcal{D}(\mathbf{x})$ in cells where measurements were taken.
4) Propagate the information about the disturbance throughout the state space using Equation 6.
5) Use HJI reachability to generate the unsafe set $Pre_\tau(\mathcal{K})$ and the optimal safe control actions u^*.

Of course an initial conservative safe set must be generated for use while the first data set is recorded; Figure 2 shows this initial safe reachable set, based on conservative estimates of the worst-case disturbances generated by prior experience with the vehicle. Intuition helps explain the shape of the set: it is safer to travel at positive velocities than negative velocities because the quadrotor can always command zero thrust and count on gravity to slow it down. Additionally, the faster the vehicle is traveling in the positive direction the closer it can be to the ground (and the further it must be from the ceiling), and vice versa.

C. Performance via Reinforcement Learning

To learn a better altitude controller for the quadrotor we chose to use Policy Gradient with the Signed Derivative

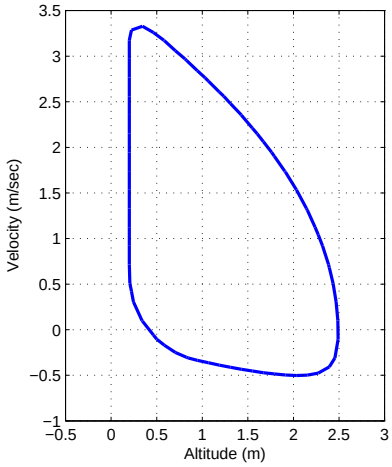

Fig. 2. The initial safe reachable set generated by a conservative estimate of the range of possible values for the disturbance. The safe area is the interior of the region; the unsafe area is the exterior. Note that the safe altitudes range from 0.2 m (since the quadrotor's origin is approximately 0.2 m above its lowest point) and 2.5 m (just below the maximum height at which the VICON cameras in our lab can accurately detect the vehicle).

(PGSD) [8]. This choice was motivated by several reasons: the fact that PGSD has been demonstrated on hardware-in-the-loop systems in the past (albeit ones in which there was very little danger of severely damaging people or hardware); the wish to demonstrate the flexibility of IGSOLR with respect to the choice of learning algorithm (i.e. to show that IGSOLR works even when the learning algorithm is model-free, as PGSD is); and because of PGSD's ease of implementation.[2] We will now briefly review PGSD before describing the experimental platform itself. As with the section on HJI reachability, our review of PGSD is necessarily brief; for a more detailed description of the algorithm we refer the reader to Kolter and Ng [8], from which this review is closely derived.

PGSD is a simple method for performing approximate gradient descent to learn a control law for a Markov Decision Process (MDP). In PGSD, we begin by assuming that the cost function of being in state $\mathbf{x}_i$ and taking action u_i is quadratic,

$$C(\mathbf{x}_t, u_t) = (\mathbf{x}_t - \mathbf{x}_t^*)^\top Q (\mathbf{x}_t - \mathbf{x}_t^*) + u_i^\top R u_t \quad (7)$$

where $\mathbf{x}_t^*$ is the desired state of the system at time t (i.e. a reference trajectory), and Q and R are diagonal positive-semidefinite matrices that penalize deviation from the reference trajectory and control input respectively.

PGSD also assumes the control law is linear in the state features, i.e. $u = \theta^\top \phi(\mathbf{x})$, where θ is a vector of parameters that PGSD will attempt to learn, and $\phi(\mathbf{x})$ is a vector of features that map from the state to scalar values. If we define

[2]It is undoubtedly true that other more advanced learning algorithms could have learned an even better altitude controller; we remind the reader that the purpose of this example was not to advance the state of the art in quadrotor altitude control, but to demonstrate how the IGSOLR framework functions, and we believe PGSD was sufficient for that purpose.

the sum of one-step costs of executing the policy θ starting at $\mathbf{x}_0$ over a horizon l as

$$J(\mathbf{x}_0, \theta) = \sum_{t=1}^{l} C(\mathbf{x}_t, u_t),\ u_t = \theta^\top \phi(\mathbf{x}_t) \tag{8}$$

then the standard approach for performing policy gradient descent on the parameters θ is to update them according to

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\mathbf{x}_0, \theta) \tag{9}$$

where α is the learning rate and $\nabla_\theta J(\mathbf{x}_0, \theta)$ is the gradient of the cost function with respect to θ. When computing $\nabla_\theta J(\mathbf{x}_0, \theta)$ one encounters Jacobians of the system model with respect to the control input, i.e. terms of the form $\partial(x_t)_i / \partial(u_t)_j$. PGSD approximates these terms with a signed derivative matrix S, in which each entry, $S_{i,j}$, is $+1$ if increasing the jth control will increase the ith state, or -1 if the opposite is true. Additionally, only one entry in each row of S is non-zero, corresponding to whichever control has the most dominant effect on the given state. Given this signed derivative matrix, PGSD then prescribes a way for computing the approximate gradient, $\tilde{\nabla}_\theta J(\mathbf{x}, \theta)$.

In summary the algorithm for performing RL via PGSD is:

1) Execute the policy $u = \theta^\top \phi(\mathbf{x})$ for l steps and record the states $\mathbf{x}_0, \ldots, \mathbf{x}_l$ and control inputs $u_0, \ldots, u_l$.
2) Compute the approximate gradient of the cost function with respect to the controls, $\tilde{\nabla}_\theta J(\mathbf{x}, \theta)$, using the signed derivative matrix S.
3) Update the parameters θ according to Equation 9.

Of course as with most reinforcement learning algorithms, design choices must be made before PGSD can be successfully executed, including what features $\phi(\mathbf{x})$ to use, what initial values θ_0 to use, as well as what values to use for the cost matrices Q and R. For our experiment, we chose to use

$$\phi(\mathbf{x}_t) = \begin{bmatrix} 1 \\ \max(x_t - x_t^*, 0) \\ \min(x_t - x_t^*, 0) \\ \max(\dot{x}_t, 0) \\ \min(\dot{x}_t, 0) \\ \Delta t (x_t - x_t^*) \\ x_t \end{bmatrix}, \quad Q = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = [0], \tag{10}$$

This choice was motivated by the fact that the quadrotor's dynamics were likely to depend on the direction of its velocity [7], so it made sense to have different features (and thus different parameters) for positive and negative velocity values; similar reasoning motivated splitting the altitude error into positive and negative portions. Additionally, the last term in $\phi(\mathbf{x})$ was chosen to allow the controller to depend on the quadrotor's altitude in order to account for ground effect. Finally, the first term in $\phi(\mathbf{x})$ is intended to represent the nominal thrust required to keep the quadrotor aloft, and the second to last term is an approximation for integral error. Although past work has found the use of an acceleration term critical [7, 15], we did not find it necessary.

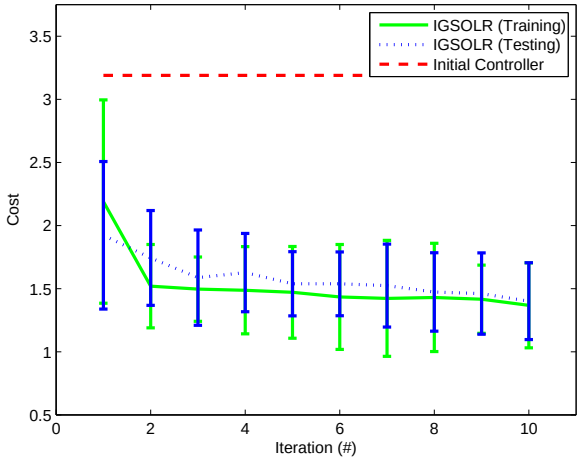

Fig. 3. Average cost of running the IGSOLR controller over the training and testing trajectories versus iteration number, along with 95% confidence intervals, averaged over ten runs. The average cost of running the initial hand-tuned controller on the testing trajectory is plotted as a horizontal line for comparison.

IV. Experimental Results

Code for running IGSOLR and PGSD was developed in MATLAB and controlled the quadrotor at a rate of 20Hz. (One important note is that it was not necessary for the learning horizon for the reachability recalculations, r, and the learning horizon for the RL algorithm, l, to be the same; we chose to set $r = 100$ and $l = 7$ seconds.) The algorithm was run on a training trajectory (in altitude only) that lasted 60 seconds, which was designed to cover as wide a range of the vehicle's state space as possible. (Whenever the training trajectory ventured outside the learned safe set, it was clipped to remain inside.) This training trajectory was repeated ten times for a total of ten minutes of training. (To account for noise, the results below show the average of ten runs over this ten minute training period.) For comparison the controllers that resulted after each iteration of training were also run on a randomly generated testing trajectory. Both the learned controllers and the initial controller (which is currently the default used on our quadrotors, and was hand-tuned for performance) were run on the testing trajectory ten times and their results recorded.

A. Results: Performance

Figure 3 shows the performance of the IGSOLR controller versus iteration number, where each iteration corresponds to one complete run of the training trajectory. As would be expected the reinforcement learning quickly converges to a much better controller than the hand-tuned one, whose cost (averaged over ten runs) is also plotted for comparison. Note that even the first iteration of the IGSOLR controller does better than the hand-tuned controller since the learning horizon was much shorter than the entire trajectory, enabling the system to learn within the first few seconds of the run.

Figure 4 shows a sample run of the hand-tuned and IGSOLR controllers over one run of the testing trajectory. Despite a

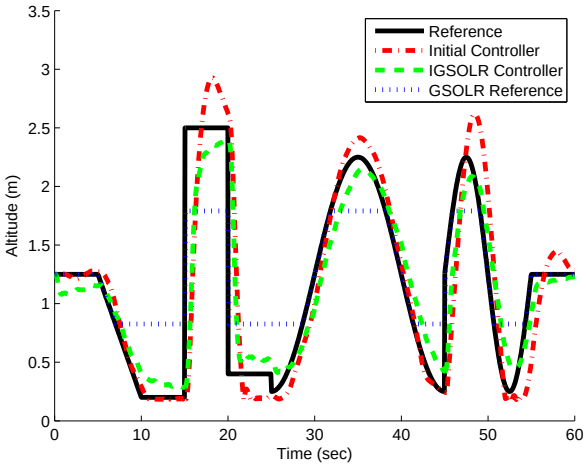

Fig. 4. Sample trajectory of the quadrotor's altitude using the hand-tuned controller versus the IGSOLR controller. A reference trajectory truncated to be safe under GSOLR is also shown for comparison.

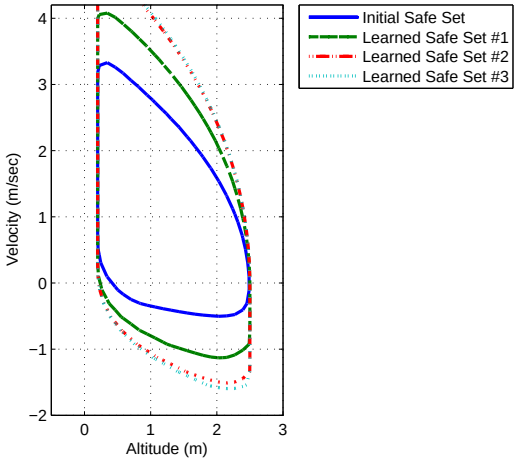

Fig. 5. The initial safe reachable set and the first three safe reachable sets learned by the IGSOLR algorithm for one of the runs. Note that under GSOLR the quadrotor would only be allowed to operate in the interior of the initial safe set.

great deal of hand-tuning, the initial controller shows dramatic overshoot both when ascending to a higher altitude as well as to a lesser extent when descending to a lower altitude; the learned IGSOLR controller does not show such dramatic errors. Additionally, the tracking performance of the IGSOLR controller appears to be better than that of the hand-tuned controller.

B. Results: Safety

Figure 5 shows the initial safe reachable set generated by a conservative estimate of the maximum possible range of the disturbances as well as the safe reachable sets that were learned iteratively by the IGSOLR algorithm as the quadrotor explored its state space. As would be expected, the initial safe set is smaller than the first learned safe set

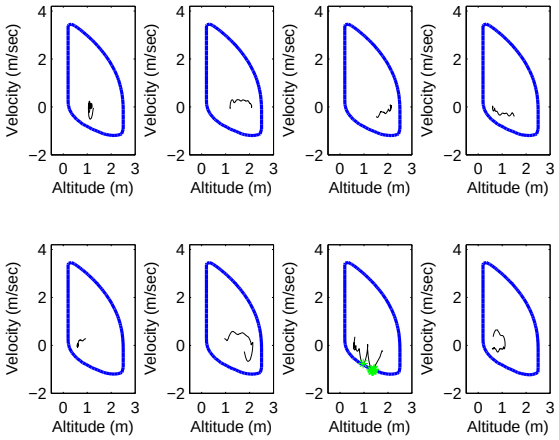

Fig. 6. Eight successive safe reachable sets learned during a typical run of the IGSOLR controller; the thinner lines are the traces of the state during the period in which the given safe set was used, and the asterisks indicate points along the state trace where IGSOLR used the safety-ensuring control instead of the controller being learned.

since the conservative disturbances are larger than the learned disturbances. This trend continues (the first is smaller than the second, etc.) as the system learns that the disturbances are in fact smaller than its prior estimate throughout the state space.

It is worth emphasizing that had GSOLR been used instead of IGSOLR, the vehicle would have been restricted to the initial safe set pictured in Figure 5. Although additional experiments showed that PGSD is able to learn a controller with comparable performance in this restricted state space, the vehicle itself would not be able to operate safely in as wide a range of the state space, as shown by the GSOLR reference trajectory in Figure 4 which has been truncated to always stay inside the initial safe set. This shows the major improvement of IGSOLR over GSOLR: that by learning the disturbances online the system is able to operate in a wider area of the state space.

Figure 6 shows a set of eight successive safe reachable sets generated by the IGSOLR algorithm, along with traces of the state during the period in which each safe set was being used. As would be expected the state trace remains inside the safe set at all times, with occasional use of the safety-ensuring control when the controller being learned takes the system state too close to the edge of the safe set.

V. Conclusion

This paper has proposed a novel extension to GSOLR known as Iterated Guaranteed Safe Online Learning via Reachability. In IGSOLR the disturbances are modeled in a state-dependent manner and learned online, and the safe sets are periodically recomputed (in parallel with whatever machine learning is being run online to control the system). Additionally IGSOLR was demonstrated in a real-world scenario in which RL (in particular, PGSD) was used to learn a policy for quadrotor altitude control, while HJI reachability analysis

guaranteed safety. PGSD was able to quickly converge to a controller which out-performed the hand-tuned controller while the safety of the system was tuned automatically to match the actual performance of the system, resulting in a high-performance system which avoided putting expensive vehicle hardware at risk.

In the future we hope to expand on this work in a variety of ways. One area that we are beginning to investigate is how to do exploration expressly for the purpose of learning disturbances. In the example in this paper the machine learning running in parallel was trying to learn a better controller, however it would be interesting to use instead a controller that attempts to explore the state space to learn more about the disturbances. A related idea that we are exploring is the interplay between learning and safety, in particular how different models for the disturbance can affect the learned reachable safe sets, which can then affect where in the state space the system is able to explore and learn new information. Another facet of this interplay that we are looking into is the question of how to generate a controller that better synthesizes the two goals of safety and learning; instead of simply replacing the learning-based control when the reachable sets dictate, we would like to more closely couple the safety and learning so that the machine learning algorithm automatically weighs safety considerations when deciding how to control the vehicle.

As the future work described above indicates there is clearly still a great deal to be done in order to truly integrate safety and learning. It is our hope that by continuing to develop tools like IGSOLR online machine learning will soon be able to be used in safety-critical situations.

VI. Acknowledgments

The authors wish to thank the reviewers for their extremely detailed, insightful, and critical reviews, which helped a great deal in strengthening this paper. Additionally the authors wish to thank the open source community for making the experiments in this paper much easier to implement than would otherwise be possible. This research was funded in part by the "Smart Adaptive Reliable Teams for Persistent Surveillance (SMARTS)" project administered by the ONR under MURI grant #N00014-09-1-1051.

References

[1] AscTec Pelican — Ascending Technologies. URL http://www.asctec.de/asctec-pelican-3/.

[2] Pieter Abbeel, Adam Coates, and Morgan Quigley. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[3] Anil Aswani, Humberto Gonzalez, S. Shankar Sastry, and Claire Jennifer Tomlin. Provably Safe and Robust Learning-Based Model Predictive Control. *ArXiv e-prints*, 2011.

[4] Jerry Ding et al. Hybrid Systems in Robotics: Toward Reachability-Based Controller Design. *IEEE Robotics & Automation Magazine (RAM)*, (September), 2011.

[5] Jeremy Hugh Gillula and Claire Jennifer Tomlin. Guaranteed Safe Online Learning of a Bounded System. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, September 2011.

[6] Jeremy Hugh Gillula and Claire Jennifer Tomlin. Guaranteed Safe Online Learning via Reachability : Tracking a Ground Target Using a Quadrotor. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, May 2012.

[7] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire Jennifer Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference (GNC)*, volume 4, Hilton Head, SC, August 2007. Citeseer.

[8] J Zico Kolter and Andrew Y. Ng. Policy Search via the Signed Derivative. In *Proc. of Robotics: Science and Systems (RSS)*, Seattle, WA, 2009.

[9] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 593—-600, Bonn, Germany, 2005.

[10] Ian M. Mitchell, Alexandre M. Bayen, and Claire Jennifer Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50(7):947–957, July 2005. ISSN 0018-9286. doi: 10.1109/TAC.2005.851439.

[11] Andrew Y. Ng and H. Jin Kim. Stable adaptive control with online learning. In *Proc. of Neural Information Processing Systems (NIPS)*, number 1, 2005.

[12] Theodore J. Perkins and Andrew G. Barto. Lyapunov Design for Safe Reinforcement Learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

[13] John W Roberts, Ian R Manchester, and Russ Tedrake. Feedback Controller Parameterizations for Reinforcement Learning. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2011.

[14] Jacob Steinhardt and Russ Tedrake. Finite-time Regional Verification of Stochastic Nonlinear Systems. In *Proc. of Robotics: Science and Systems (RSS)*, June 2011.

[15] Steven L. Waslander, Gabriel M. Hoffmann, Jung Soon Jang, and Claire Jennifer Tomlin. Multi-agent quadrotor testbed control design: integral sliding mode vs. reinforcement learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3712–3717, Edmonton, AL, 2005. Ieee. ISBN 0-7803-8912-3. doi: 10.1109/IROS.2005.1545025.

[16] J.F. Ziegler et al. IBM experiments in soft fails in computer electronics (1978-1994). *IBM Journal of Research and Development*, 40(1):3–18, 1996.