2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Reinforcement Learning for Autonomous Dynamic Soaring in Shear Winds

Corey Montella and John R. Spletzer

***Abstract*— Dynamic soaring (DS) is an aerobatic maneuver whereby a gliding aircraft harnesses energy from horizontal wind that varies in strength and/or direction to support flight. Typical approaches to dynamic soaring in autonomous unmanned aerial vehicles (UAVs) use nonlinear optimizers to generate energy-gaining trajectories, which are then followed using traditional controllers. The effectiveness of such a strategy is limited by both the local optimality of the generated trajectory, as well as controller tracking errors. In this paper, we investigate a reinforcement learning (RL) approach working in continuous space to control a DS aircraft flying in shear wind conditions. The RL controller operates in two stages: In the first stage, it observes a traditional sample-based controller flying a locally optimal DS trajectory generated *a priori*. In the second stage, the sample-based controller is removed and authority is passed to the RL algorithm. We show that by deviating from the original planned trajectory, the RL controller is able to achieve better performance than its baseline teacher controller.**

I. Introduction and Motivation

Dynamic soaring (DS) is an aerobatic maneuver routinely performed by seabirds like the albatross to extract energy from horizontal winds that vary in strength or direction. Recent attention has been paid to DS for autonomous unmanned aerial vehicles (UAVs) due to the potential to drastically extend mission duration. For instance, a gliding aircraft in the jet steam could act as a low orbit communication relay by performing a continual cycle of dynamic soaring maneuvers, which could potentially support flight indefinitely [1].

While no autonomous DS flights have been recorded, several authors have proposed feasible solutions to demonstrate the maneuver. These generally involve locally optimal trajectories generated offline by a nonlinear optimizer [2], and a controller capable of tracking them. Flanzer proposes a robust trajectory optimization approach which aims to ensure safety while maintaining an energy neutral trajectory [3]. He further proposes a cyclical controller which uses information gained on each DS cycle to correct errors in the next cycle. Lawrance and Sukkarieh employ a sample-based controller to track global targets selected from a wind field map [4].

In this paper we are interested in the problem of dynamic soaring in shear winds. Shear layers are the result of ground features, such as ridges, which block prevailing winds and create a gradient in which DS maneuvers can be performed. Above the ridge, wind blows at a constant speed, while below the ridge the air is still. Thus a wind gradient exists in this region which we parameterize by a maximum wind strength w_{max}, and a height Δh. Since shear layers are local phenomena, we are interested in exploring loiter-type trajectories, where the soaring aircraft returns to its starting position and orientation after it executes a DS maneuver, ready for a second cycle. The aircraft can be thought to orbit a fixed point, which we use as a zero reference for the coordinate system in this work. Figure 1 depicts an example loiter trajectory in a shear wind field.

The aforementioned trajectory following techniques rely on the idea that a generated trajectory is at least locally optimal, so following one should net the aircraft a near optimal energy gain. However, wind gusts, tracking errors, or inaccurate sensor measurements can often cause the aircraft to deviate from the planned path. For an energy-maximizing aircraft in these situations, returning to the planned trajectory can potentially cost more energy than allowing for some deviation to accommodate the perturbed state.

Reinforcement learning (RL) is a machine learning paradigm which aims to learn a controller based on observed actions. Wharington specifically approaches several soaring problems, including thermal soaring, essing, and dolphin soaring using RL controllers [5]. However, he stops short of RL for dynamic soaring, noting that the problem is probably intractable for hardware circa 1998. The aim of this paper is to revisit this possibility, and develop an RL controller that will learn from a conventional teaching controller, and eventually evolve to outperform it. Similar approaches have been employed to learn aerobatic helicopter control [6]. More

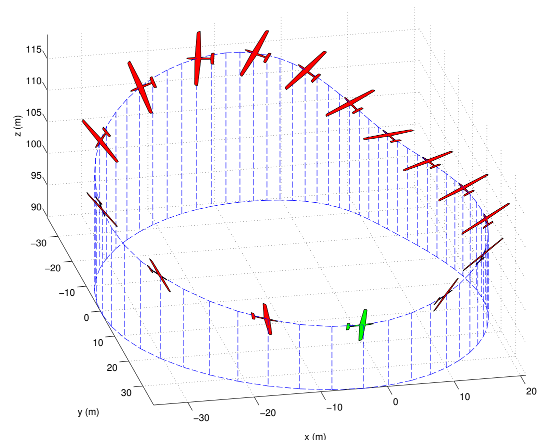

Fig. 1: An example dynamic soaring trajectory, depicting several states of the UAV along the path. The scale of the glider is magnified 3x for clarity. In this case, winds prevail from the -y direction at $w_{max} = 9m/s$, with the shear region between 100 m and 110 m in altitude.

This material is based upon work supported by the National Science Foundation under Grant No. 1065202.

The authors are with the VADER Laboraty in the Computer Science and Engineering Department of Lehigh University, Bethlehem, Pennsylvania, United States cmontella@ieee.org, spletzer@cse.lehigh.edu

specifically related to this work, Barate et. al. use motion primitives to define a rough DS controller, which is refined by a genetic algorithm [7].

The rest of this paper is organized as follows: Section II contains a brief introduction to the RL framework and the techniques we will use in this paper. Section III details our RL implementation, and Section IV describes the process used to bootstrap our RL algorithm with a conventional tracking controller. Simulation results that compare the RL controller to the teaching controller for two simulation environments are presented in Section V. We conclude and posit future directions in Section VI.

II. Reinforcement Learning

Reinforcement learning is a problem formulation which aims to maximize the performance of an agent by observing its actions and assigning rewards based on their outcomes. The goal of the agent is then to maximize the cumulative sum of its rewards over a specified time horizon.

The essential RL model consists of a set of environment states, $s_t \in S$; a set of actions, $a_t \in A$, that an agent can perform at each state; and a set of scalar reinforcement signals, r_t, achievable by the agent. The purpose of RL is to find a mapping from states to actions, called a policy π, which maximizes the cumulative reward of the agent over time, expressed as

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

where $0 < \gamma < 1$, known as the discount factor, reduces the value of future rewards. Equation 1 is known as the discounted infinite horizon reward, and is appropriate for an agent that will act for an undetermined number of actions. This is well suited for a DS task, and is the reward function we will use in this paper.

While there are many approaches to solving RL problems, in this work we will use the Q-Learning algorithm. Introduced by Watkins in 1989, Q-Learning is one of the most popular RL algorithms due to its simplicity and proven convergence properties [8]. Q-Learning is a dynamic programming algorithm that encodes the optimal policy π^* in a so-called Q-function. This is typically implemented as a matrix, where each row corresponds to a state, each column corresponds to an action, and each element holds the discounted reward associated with the state/action pairing. The Q-function is initialized arbitrarily and is updated according to the following iterative rule

$$\begin{aligned} Q_{t+1}(s_t, a_t) \leftarrow & Q_t(s_t, a_t) + \\ & \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q_t(s_t, a_t)) \end{aligned} \tag{2}$$

where s_t are states, a_t are actions, r_{t+1} is a reward, and the parameter $0 < \alpha < 1$ is known as the learning rate.

As shown by Watkins, as long as every state is visited continually, the algorithm converges to the optimal Q-function irrespective of how actions are chosen. The algorithm is therefore termed ”off-policy”, meaning the learned Q-function is independent of the policy followed during the learning process. We will exploit this feature of Q-Learning in this work.

This overview has covered only the elements of RL essential to this work. For a comprehensive introduction to reinforcement learning, see [9]. A recent survey of the application of RL to robotics problems can be found in [10].

III. Reinforcement Learning Implementation

As an inherently high dimensional, continuous state space problem, DS is not a drop-in candidate for most RL algorithms, including Q-Learning. In this section, we discuss the implementation details of the reinforcement learning approach used for our DS task, which were inspired by [11]. First we discuss our state space formulation. Then, we detail how we used a function approximator to adapt Q-Learning for a continuous state space. Finally, we detail next state selection and our reward function.

A. State/Action Space

An intuitive space to use for the learner is the position and attitude of the aircraft

$$s_t = [v_t, x_t, y_t, z_t, \psi_t, \gamma_t, \mu_t] \tag{3}$$

$$a_t = [\gamma_{t+1}, \mu_{t+1}] \tag{4}$$

where v_t is the airspeed; x_t, y_t and z_t are Cartesian coordinates relative to the loiter point; and ψ_t, γ_t, and μ_t are the yaw, pitch and roll angles. The action space consists of the commanded pitch and roll angles for the next state. However, this means the Q-function will have 9 dimensions, which is undesirable. Using a conventional controller (described in Section IV-B) we collected the states and actions for an entire trajectory to try and reduce the dimensionality of the state space through statistical analysis.

The most obvious dimensions to eliminate from the state space are γ_t and μ_t; since our time step is small (0.1 s) these dimensions are highly correlated to the action space dimensions γ_{t+1} and μ_{t+1} ($\rho > 0.99$ for each). We can further reduce the dimensionality by performing a principal component analysis (PCA) on the remaining state space dimensions. We find that the first principal component accounts for 93% of the variability in the data, and is aligned along the ψ_t axis (with a coefficient value of 0.98). Next in order are x_t (5.5%), y_t (0.9%), v_t (0.2%), and lastly h_t (0.1%). Thus by removing v_t and h_t we maintain 99.7% of the variability in the state space, for a final reduced-dimension state/action space:

$$x = [s_t, a_t]^\top = [x_t, y_t, \psi_t, \gamma_{t+1}, \mu_{t+1}]^\top \tag{5}$$

B. Generalized Regression Function Approximation

As mentioned in Section II, the Q-function is typically implemented as a matrix. In continuous spaces this is not feasible, so function approximation techniques such as neural networks are typically used to represent the Q-function [12]. Unfortunately, the immense training time needed to create neural networks means the Q-function can only be learned offline, after a robot has collected a large amount

of experience. Further, since many neural networks cannot adapt to new data, the robot might not be able to learn incrementally.

We use an approach with minimal training time that can be incrementally updated, with the trade-off of requiring a larger memory footprint. Our approach is inspired by generalized regression neural networks [13], which calculate the distances from an input vector to a set of training vectors to arrive at a value approximation. We implement our function approximator as follows: Consider a query point x_q, which is a state/action vector. We want to determine its Q-value based on a set of N state/action training vectors, X, and their associated Q-values Y. We first calculate the Euclidean distance d from x_q to each training vector $x \in X$. Then the K nearest neighbors are selected, and their distances are weighted using a radial basis function (RBF) of the form

$$w_k = e^{-(d_k/h)^2} \quad \forall k \in K \tag{6}$$

where the parameter h is the RBF bandwidth. The selection of this parameter is the only training time needed in the algorithm, and is performed using a leave-one-out optimization technique.

We then multiply these weights by the Q-values corresponding to the K selected vectors and calculate the predicted Q-value

$$q_p = \frac{\sum_k^K Y_k w_k}{\sum_k^K w_k} \times \max_k w_k \tag{7}$$

We multiply by the maximum weight to ensure that poorly supported vectors have a diminished Q-value. Without this, it is possible to extrapolate a q_p that is not properly justified by observed data. We further reduce this possibility by choosing a threshold τ, such that if an observation's maximum weight is less than τ, q_p is set to zero. In our experiments, we chose $\tau = 0.6$.

The most significant drawback of this technique is that every training observation needs to be stored, and X needs to be searched and sorted for minimum distances. Thus, as the aircraft gains experience, finding the K nearest neighbors becomes the bottleneck of the algorithm. To reduce this problem, we store X in a k-d tree [14] for fast searching. Adding to a k-d tree iteratively can negate its fast lookup properties, and re-balancing on each iteration is computationally prohibitive for large data sets. Therefore, new observations are added to a temporary vector which is searched in parallel with the tree. When the temporary vector reaches a threshold size, it is combined with the k-d tree data set, and the tree is rebalanced incorporating the new training observations.

Because X grows unbounded, searching still eventually becomes a bottleneck, even with a k-d tree. Therefore, we remove redundant points (i.e. closely clustered points have the same predictive power as a single point) using a leave-one-out optimization technique that aims to minimize the predictive impact of the removed observation. Again, since this is an expensive procedure it is not performed every iteration, but only when X grows to a set capacity.

C. Selecting the Next Best Q-Value

The maximization involved in Equation 2 is straight forward in the matrix case. For the continuous case, finding the next best action typically involves an optimization procedure that can be time-intensive. We solve the problem by sampling around the current μ and γ, since again we note that the time step between states is small, and thus our next action (the roll and pitch command) will closely resemble the UAV's current attitude. We sample 20 values each from the ± 5 degree regions surrounding the current μ and γ. We take the combination of these samples and the current μ and γ for a total of 441 action pairs. These actions are then concatenated with a vector of the current state, and fed into the function approximator described in the previous section. We then simply take the maximum of the resulting predicted Q-values for use in the Q-function update.

We note here that during the learning phase described in Section IV, the aircraft is off-policy, i.e. it does not follow the Q-maximizing μ and γ commands. When the RL controller is granted authority over the aircraft, the algorithm is then on-policy, and follows these commands.

D. Rewards

There are several options suitable for rewards in a DS task, such as path error or energy gain per time step. For loitering trajectories, the simplest reward function is sparse, meaning that the UAV earns zero reward at every state except the one where it completes a loop. An obvious choice for the loop completion reward suitable for our work is the energy gain over the completed cycle

$$r_{t_f} = \frac{1}{2} m(v_f^2 - v_i^2) + mg(h_f - h_i) \tag{8}$$

where v_f and v_i are the final and initial airspeeds of the UAV, h_f and h_i are the final and initial altitudes, m is the UAV mass, and g is the acceleration due to gravity.

Due to the sparse nature of this reward function, we increase the rate of learning by recording the states and actions of the aircraft over the trajectory. When a reward is granted at the end of the cycle, we replay these states in reverse order and update the Q-value with Equation 2 until the reward propagates to the start of the trajectory, as suggested by [15].

IV. The Learning Process

Q-Learning typically operates in a series of episodes, where the learning agent performs actions until a goal state is reached, it is manually reset, or it fails (e.g. the glider stalls). When the algorithm starts, the agent has no knowledge of the Q-function, so it must act at random. If it is not guided by a continuous reward function, a significant amount of time can be devoted to finding the first reward. For our DS problem, finding a reward accidentally through a series of random actions is highly improbable, not to mention unsafe for a system as unstable as an aircraft. Therefore, we must direct the agent to interesting regions of the state space through some other means. We do this through a two

stage learning process. First, the RL controller observes a teaching controller as it demonstrates several correct DS trajectories. During this process, the RL controller has no authority over the aircraft, so it passively constructs a Q-function corresponding to the teaching controller's states and actions. In the second stage, authority over the aircraft is granted to the RL controller. Now, the RL controller engages the Q-function it built while while observing the teaching controller.

In this section we detail the necessary components needed to enable this learning process, including how DS trajectories are generated and how they are followed by the teaching controller. Before we do, we note that through this process we are not merely encoding the decisions of the teaching controller within the RL controller. As will become clear in the experimental results, the RL controller is simply gaining experience, which it will use to fly a better trajectory than any demonstrated by the teaching controller.

A. Trajectory Generation

In [16] we presented a method for finding a locally optimal trajectory in a fixed, known wind field. Each trajectory Π consists of a series N collocation points, each of which are associated with a state and action of the UAV at time t of the flight

$$\Pi = [s_1, a_1, ...s_N, a_N]^\top \tag{9}$$

In our experiments, we use $N = 64$ collocation points. We use a nonlinear optimization technique which aims to maximize the energy gained over the cycle under constraints which ensure the path observes periodicity of the trajectory and boundary values on all state variables. We relate collocation points using nonlinear constraints formed from point mass glider kinematics. This is the same model used for simulation in Sections IV-B and V-B.

Previously we assumed a wind gradient constant in altitude. Now, we relax this condition and assume a more complex sigmoid wind field, which is more representative of a shear type wind

$$W_x = \frac{w_{max}}{1 + e^{b(a-z)}} \tag{10}$$

$$\frac{dW_x}{dz} = \frac{b\, w_{max}\, e^{b(a+z)}}{(e^{a+b} + e^{bz})^2} \tag{11}$$

where w_{max} is the maximum wind speed at the top of the shear layer, a is the center altitude of the shear layer, b is a parameter that controls the steepness of the shear gradient, and z is the aircraft's current altitude.

The trajectories are parameterized by a maximum shear wind strength w_{max}, the shear region height Δh, and an initial aircraft velocity v_0.

B. Teaching Controller

A teaching controller based on trajectory rollout [17] was used to demonstrate correct DS trajectories to the RL algorithm. This is a sample-based approach on the input space of pitch and bank rates $(\dot{\gamma}, \dot{\mu})$. A hypothetical sample trajectory $T = \{x_0, \gamma_1, \mu_1, ..., \gamma_M, \mu_M\}$ was specified by the current aircraft state s_0, and M pitch and bank angles determined by fixing the sampled pitch and bank rates over the M-step control horizon

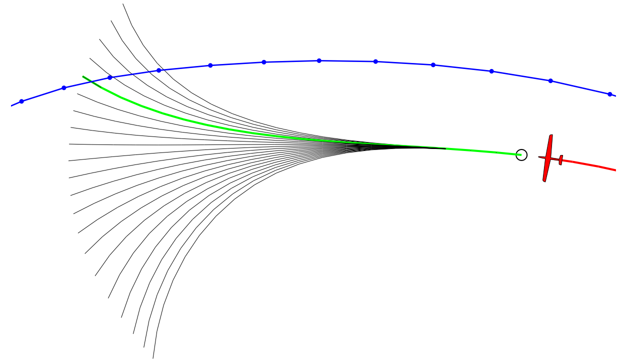

Fig. 2: Diagram of the sample-based controller with a 2 second planning horizon. The blue line indicates the DS trajectory. The green highlighted line is the lowest score planned aircraft trajectory. The black circle ahead of the glider demonstrates the latency planning feature (see Section V-C). While the controller works in 3D, this diagram is 2D for illustration purposes.

The angular velocities were then integrated forward in time yielding a projected path over the chosen control horizon, updating the hypothetical trajectory to $T = \{s_0, s_1, \gamma_1, \mu_1, ..., s_M, \gamma_M, \mu_M\}$. An advantage of sampling the control rates is that we ensure each trajectory is feasible in terms of the aircraft kinematics and constraints placed on the roll and bank angles. Trajectories that violate our roll and pitch limits were discarded.

Each trajectory T_i was then assigned a score, equal to the distance from the pose of the aircraft at the end of the trajectory to the closest line segment in Π. This is not the distance to the closest collocation point in Π, but is the normal projection to the line segment connecting the two closest collocation points in Π.

The optimal trajectory $T^* = \arg\min(D(T, \Pi))$ was then selected, and the associated pitch and bank angles $(\gamma_1^*, \mu_1^*) \in T^*$ were issued as commands to the aircraft autopilot. The whole process is repeated again at the next time step. A diagram of this controller is depicted in Figure 2.

The equations of motion used to integrate the aircraft state forward are derived in [16], and are not repeated here in the interest of brevity.

V. Simulated Experiments

In this section we present simulation results for the ridge-based soaring task outlined in Section I. For these experiments, we assumed a fixed, known wind field. First, we describe the simulated aircraft model, which is based on our low altitude DS development airframe. Next we present simulation results of our point mass model, which are used to establish baseline performance of the teaching and RL controllers. Finally we present simulation results from a high-fidelity 6 DoF aircraft simulator that interfaces with our autopilot hardware.

A. Development Platform

Our ultimate goal is to demonstrate DS on an actual aircraft. To this end, the aircraft model used for trajectory

generation and all simulation experiments is based upon our development platform for low-altitude soaring. The airframe is a heavily modified EScale Fox commercial off-the-shelf radio-controlled sailplane. While not the most aerobatic glider from a DS perspective, it features a large fuselage necessary for housing the Cloud Clap Piccolo autopilot, the PC/104+ on-board flight computer, as well as batteries for powering the avionics and aircraft motor. The Fox has a 2.8m wingspan and a loaded mass of 4.06kg.

B. Point Mass Model Simulations

Our first set of experiments employed a simulator based on the same point mass model used to generate DS trajectories. While this model takes into account basic aerodynamic forces, it does not not simulate more complex dynamics. Regardless, it offers three key benefits that are not available in our alternative simulation environment: First it can be configured to run in faster-than-real-time, which accelerates the learning process. Second, the aircraft can be initialized to any state, which allows us to demonstrate arbitrary experiences to the RL controller. Finally, it provides us with baseline performance for the teaching controller, since the sample-based controller model is identical to the simulation environment model.

For this test we generated a DS trajectory with maximum shear strength $w_{max} = 9$ m/s, shear region height $\Delta h = 10$ m, and initial UAV airspeed $v_0 = 16$ m/s. For the teaching phase, we chose the initial location of the UAV as the first collocation point on the DS trajectory. We then demonstrated a series of 10 flights to the RL algorithm using the teaching controller. Since the teaching controller is deterministic, at the start of each episode we corrupted the UAV's starting position, attitude, and airspeed with random Gaussian noise to ensure a variety of trajectories. The mean energy gain for these 10 flights was 352.62 J, with a maximum energy gain of 359.39 J; while the mean RMS path error was 0.93 m. A controller which perfectly tracks this particular trajectory should gain 364.96 J of energy. Thus, the difference between this value and the mean energy gain is the result of controller tracking error.

When these 10 flights concluded, authority was then granted to the RL controller. Figure 3 depicts the energy gain of the RL controller as it learned new trajectories. Learning ceased when energy gain between episodes was below a target threshold. In this case, it happened also to be 10 episodes. An interesting feature of this result is the decrease in performance seen on the RL controller's initial flight. This phenomena mirrors that reported by [18].

Once learning ended, we fixed the Q-function and ran 10 more episodes with the RL controller, each with starting con-

	Teaching	RL	Percent Gain
Mean Energy Gain (J)	352.62	389.96	11%
Maximum Energy Gain (J)	359.39	398.98	11%
Mean RMS Path Error (m)	0.93	4.64	-76%

TABLE I: Results for the point mass model simulation after learning converged. Means are computed for 10 episodes for each controller.

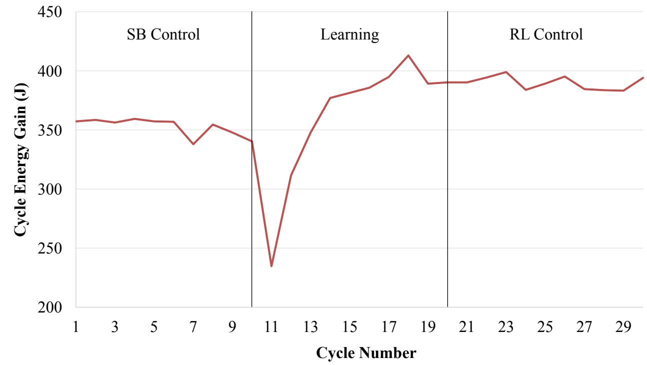

Fig. 3: Training results for the RL algorithm. On its first attempt the RL controller performs significantly worse than the teaching controller. However, performance quickly surpasses the teaching controller after only a few iterations.

ditions corresponding to the first 10 flights on the teaching controller. We saw a mean cycle energy gain of 389.96 J with a maximum of 398.98 J; while the mean RMS path error was 4.63 m. The foregoing statistics are summarized in Table I.

From these results we can see that by teaching the RL controller example trajectories, we were not simply encoding the actions of the teaching controller for the RL algorithm to replay. Instead, the RL controller used that experience to find distinct trajectories that were more efficient than any of the demonstrated trajectories.

C. High Fidelity Simulations

Cloud Cap provides a high fidelity 6 DoF simulator for use with their Piccolo autopilots. This features a "hardware-in-the-loop" (HiL) configuration wherein the simulator connects directly to the avionics via an external CAN interface, and sends simulated sensor data. The loop is closed when the simulator reads actuator positions, applies them to the dynamics model, and calculates new sensor data for the avionics.

The Cloud Cap simulator features models for aerodynamics, sensors, actuators, and inertia. The aerodynamics are derived from AVL, which we used to model the control surfaces and lift surfaces of our Fox glider. New states are then estimated by taking the current state, the information from the dynamics model, and the position of all control surfaces.

Before we present results, we note one modification made to the teaching controller for HiL simulation experiments. Due to latency in the autopilot, commands are executed approximately 600 ms after they are sent to the avionics. Therefore, we project the current state forward 6 time steps (our controller runs at 10 Hz) using the point mass model. Thus, when the aircraft eventually reaches the projected state, it will execute the command planned for that state.

Figure 4 depicts two HiL soaring trajectories and the same DS trajectory used in the previous simulations. The green trajectory employed the teaching controller, which experienced an energy gain of 238.09 J, a percent decrease of

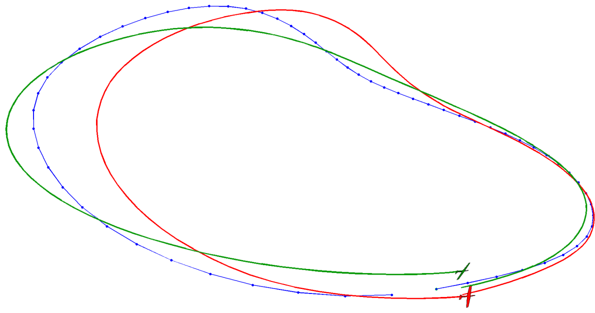

Fig. 4: Soaring results for the HiL simulation. The target DS trajectory is depicted in blue, with the RL controller path in red, and the sample-based controller path in green. The RL Controller does not track the blue trajectory, but creates similar path from learned experience.

32.48% compared to the mean energy gain in the point mass simulator. There are three primary reasons for this: First, latency in responding to commands, which we already noted. Second, the integrated states are based on an incomplete model. Finally, the planning horizon was increased from 1 s to 2 s. Using the 1 s horizon, the controller exhibits overshoot and never converges to the trajectory. Increasing the horizon to 2 s eliminates this behavior, but also serves to "round out" the UAV path, making it more ellipsoidal instead of kidney bean shaped.

The red trajectory was followed by the RL controller, and experienced an energy gain of 354.34 J, a percent decrease of 9.13% compared to the point mass simulator. Compared to the HiL sample-based controller, the RL controller was able to extract 49% more energy. The UAV did not start on the planned trajectory, but instead of tracking to it and losing energy, the UAV used its learned Q-function to find another path which closely resembles the planned one. An interesting result here is the RL controller was not re-trained in the high fidelity simulator model; the Q-function learned from the point mass model seems to have generalized well to the higher fidelity model, but more tests are needed to verify this.

VI. Conclusions

This paper presented a reinforcement learning controller for a dynamic soaring application. We demonstrated that the RL controller not only outperforms our conventional tracking planner, but by transferring a learned Q-function between simulation environments, we showed it is also robust to shortcomings in the model. This feature of the controller will be beneficial when transferring from the HiL environment to a true flying aircraft.

The results shown herein are promising, but require a great deal of work before RL can be employed on an actual UAV. First, in this paper we assumed a fixed, known wind field. In reality, the wind field is neither fixed nor precisely known. Modifying the RL controller to accommodate uncertain and changing wind conditions is paramount for a working system.

Second, we have not addressed the issue of exploration vs. exploitation, which is a central tenant of RL. Here, the RL controller purely exploits the Q-function, and any new states are found by inferring favorable Q-values from the function approximator. For a proper RL controller, we need to specify how to handle exploration, which could possibly lead to faster convergence or better steady-state performance.

Finally, when exploration capabilities are added, it is important to prevent the aircraft from entering states which may yield dangerous control strategies. This safety concern must be addressed before field testing on our development platform.

Acknowledgments

The authors would like to thank Professor Héctor Muñoz-Avila for his direction and guidence on reinforcement learning.

References

[1] J. Grenestedt and J. R. Spletzer, "Optimal energy extraction during dynamic jet stream soaring," in *Proc. of the AIAA Guidance, Navigation and Control Conf.*, Toronto, Canada, August 2010.
[2] Y. Zhao, "Optimal patterns of glider dynamic soaring," *Optimal Control Applications and Methods*, vol. 25, no. 2, pp. 67 – 89, 2004.
[3] T. Flanzer, "Robst trajectory optimization and control of a dynamic soaring umanned aerial vehicle," Ph.D. dissertation, Dept. of Aeronautics and Astronautics, Stanford University, 2012.
[4] N. Lawrance and Sukkarieh, "Path planning for autonomous soaring flight in dynamic wind fields," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.
[5] J. Wharington, "Autonomous control of soaring aircraft by reinforcement learning," Ph.D. dissertation, Royal Melbourne Institute of Technology, 1998.
[6] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Proc. of the Neural Information Processing Systems Conf.*, Vancouver, Canada, December 2006.
[7] R. Barate, S. Doncieux, and J. arcady Meyer, "Design of a bio-inspired controller for dynamic soaring in a simulated unmanned aerial vehicle," *Bioinspiration & Biomimetics*, vol. 1, no. 3, pp. 76 – 88, 2006.
[8] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, 1989.
[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artificial Intelligence Research*, vol. 4, pp. 237 – 285, 1996.
[10] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. of Robotics Research (IJRR)*, vol. 32, no. 11, pp. 1238 – 1274, 2013.
[11] W. D. Smart, "Making reinforcement learning work on real robots," Ph.D. dissertation, Dept. of Computer Science, Brown University, 2002.
[12] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, pp. 55 – 73, 2009.
[13] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, no. 1, pp. 109–118, 1990.
[14] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 5, pp. 509 – 517, 1975.
[15] L. Long-Ji, "Self-improving reactive agents based on reinforcement learning, planning, and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.
[16] J. Grenestedt and J. R. Spletzer, "Towards perpetual flight of a gliding unmanned aerial vehicle in the jet stream," in *Proc. of the IEEE Int. Conf. on Decision and Control (CDC)*, Atlanta, United States, December 2010.
[17] B. P. Gerkey and K. Konolige, "Planning and control in unstructure terrain," in *Proc. of the ICRA Workshop on Path Planning and Costmaps*, Passadena, United States, May 2008.
[18] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Washington D.C., United States, May 2002.