Learning Swing-free Trajectories for UAVs with a Suspended Load in Obstacle-free Environments

Aleksandra Faust¹, Ivana Palunko², Patricio Cruz³, Rafael Fierro³, and Lydia Tapia¹

Abstract—Attaining autonomous flight is an important task in aerial robotics. Often flight trajectories are not only subject to unknown system dynamics, but also to specific task constraints. We are interested in producing a trajectory for an aerial robot with a suspended load that delivers the load to a destination in a swing-free fashion. This paper presents a motion planning framework for generating trajectories with minimal residual oscillations (swing-free) for rotorcraft carrying a suspended load. We rely on a finite-sampling, batch reinforcement learning algorithm to train the system for a particular load. We find the criteria that allows the trained agent to be transferred to a variety of models, state and action spaces and produce a number of different trajectories. Through a combination of simulations and experiments, we demonstrate that the inferred policy is robust to noise and to the unmodeled dynamics of the system. The contributions of this work are 1) applying reinforcement learning to solve the problem of finding a swing-free trajectory for a rotorcraft, 2) designing a problem-specific feature vector for value function approximation, 3) giving sufficient conditions that need to be met to allow successful learning transfer to different models, state and action spaces, and 4) verification of the resulting trajectories in simulation and to autonomously control quadrotors.

I. INTRODUCTION

Unmanned aerial vehicles (UAVs) play an increasing role in a wide number of missions such as remote sensing, transportation, and search and rescue missions. Cargoes may consist of food and supply delivery in disaster struck areas, patient transport, or spacecraft landing. Planning motions for a UAV carrying a load is complex because load swing is difficult to control. However, it is necessary for the safety and success of the mission.

Helicopters and quadrotors are ideal candidates for autonomous cargo delivery tasks because they are highly maneuverable, holonomic vehicle’s with the abilities of vertical takeoff and landing, and single-point hover. However, they are inherently unstable systems with complicated, nonlinear dynamics. Furthermore, the added suspended load changes the dynamics of the system.

Our goal is to find a fast trajectory with minimal residual oscillations (swing-free) for a rotorcraft aerial robot carrying a suspended load as described in [8]. In addition, swing control during the flight is desired. We assume that we know the goal state of the vehicle, and the initial state can be arbitrary. Furthermore, we assume that we have a black box simulator (or a generative model) available, but we make no assumptions about the dynamics of the system while designing the algorithm.

We apply a machine learning approach to obtain a swing-free trajectory. We learn the task using an approximate value iteration (A VI) reinforcement learning algorithm. The value function is parametrized with problem-specific feature vectors. The learning and trajectory generation are separated in two distinct phases. In the first phase, we learn the value function approximation for a particular load. Once the value function is learned, we can use it to generate any number of trajectories. These trajectories can have different starting and ending positions and use different (but compatible) models (see Figure 2). We find the sufficient criteria to allow the transfer of the learned, inferred policy to a variety of situations. We demonstrate that the approach produces a swing-free trajectory to the desired state regardless of the starting position, that is robust to noise.

To verify our approach, we learn a value function approximation for swing-free flight using a generic holonomic model of the aerial vehicle with a suspended load as a simulator. Then we generate trajectories using two models; the same holonomic model used to learn parameters, and a noisy holonomic model. We demonstrate that the trajectories are feasible by using them for autonomous control of a Hummingbird quadrotor shown in Figure 1. The summary of the results is presented here, for the full results please see [4].

II. RELATED WORK

1) Quadrotor Trajectory Tracking and Creation: Quadrotor control have been active area of research. Schoellig et al. in [9] use an expectation-maximization learning algorithm to achieve quadrotor trajectory tracking. Lupashin et al.
apply policy gradient descent to perform aggressive quadrotor multi-flips.

Palunko et al. applied dynamic programming to solve swing-free trajectories for quadrotors [8]. While dynamic programming requires pre-calculating each trajectory, the approach presented here allows us to learn the problem once, and generate any number of different trajectories. The module uses the same feature vectors, but can rely on different simulators to find the best action in any given state. The produced trajectory is sent to a robot. The green blocks are external to the learning algorithm and considered to be unknown.

2) Swing-free Trajectories in Manufacturing: Swing-free trajectories are important in industrial robotics with applications such as cranes in construction sites and for cargo loading in ports [1]. Residual oscillation reduction has applications in manufacturing environments where parts need to be transported in a limited space, [12].

3) Reinforcement and Transfer Learning: To accomplish swing-free trajectories for rotorcraft with a suspended load, we rely on approximate value iteration [3] with a specifically designed feature vector for value function approximation. Taylor and Stone [11] propose value function transfer between the tasks in different state and action spaces using behavior transfer function to transfer the value function to the new domain. Sherstov and Stone in [10] examine action transfer between the tasks, learning the optimal policy and transferring only the most relevant actions for the optimal policy. McMahan et al. [6] suggested learning a partial policy for fixed start and goal states.

III. METHODS

A. Reinforcement Learning for Swing-Free Trajectories

The approximate value iteration algorithm [3] produces an approximate solution to a Markov Decision Process (MDP) in continuous state spaces with a discrete action set. We approximate the value function with a linearly parametrized feature vector. It is in an expectation-maximization (EM) algorithm which relies on a sampling of the state space transitions, an estimation of the state value function using Bellman equation [2], and a linear regression to find the parameters that minimize the least square error.

In our implementation, the state space is a 10-dimensional vector \( s = [p \, v \, \eta \, \dot{\eta}] \) of the vehicle’s position \( p = [x \, y \, z]^T \) relative to the goal state, vehicles linear velocity \( v = [\dot{x} \, \dot{y} \, \dot{z}]^T \), load displacement angles \( \eta_L = [\phi_L \, \theta_L]^T \) and their respective angular speeds \( \dot{\eta}_L \). From Figure 3, \( L \) is the length of the suspension cable. Since \( L \) is constant in this work, it will be omitted.

The samples are uniformly, randomly drawn from a hypercube centred in the goal state at equilibrium. The action space is a linear acceleration vector \( a = [\dot{x} \, \dot{y} \, \dot{z}]^T \) discretized using equidistant steps centered around zero acceleration.

The state value function \( V \) is approximated with a linear combination of the feature vector \( F(s) \). The feature vector chosen for this problem consists of four basis functions; squares of vehicles distance to the goal, its velocity magnitude, and load displacement and velocity magnitude as shown in (1):

\[
V(s) = \psi^T \ast F(s)
\]

\[
F(s) = [||p||^2 \, ||v||^2 \, ||\eta||^2 \, ||\dot{\eta}||^2]^T
\]

where \( \psi \in \mathbb{R}^4 \).

The reward function penalizes the distance from the goal state, and the size of the load displacement. It also penalizes the negative \( z \) coordinate to provide a bounding box and enforce that the vehicle must stay above the ground. Lastly, the agent is rewarded when it reaches equilibrium. The reward function \( R(s) = c^T r(s) \) is a linear combination of basis rewards \( r(s) = [r_1(s) \, r_2(s) \, r_3(s)]^T \), weighted with vector \( c = [c_1 \, c_2 \, c_3]^T \), for some constants \( a_1 \) and \( a_2 \), where:

\[
r_1(s) = -||p||^2
\]

\[
r_2(s) = \begin{cases} a_1 & ||F(s)|| < \epsilon \\ -||\eta||^2 & \text{otherwise} \end{cases}
\]

\[
r_3(s) = \begin{cases} -a_2 & z < 0 \\ 0 & z \geq 0 \end{cases}
\]

To obtain the state transition function samples \( P(s_0, a) = s \), we rely on a simplified model of the quadrotor-load system, where the quadrotor is represented by a holonomic model of a UAV widely used in the literature [7]. The simulator returns the next system state \( s = [p \, v \, \eta \, \dot{\eta}] \) when an action \( a \) is applied to a state \( s_0 = [p_0 \, v_0 \, \eta_0 \, \dot{\eta}_0] \). Equations (2) and (3) describe the simulator. \( g = [0 \, 0 \, 0]^T \) is gravity force, \( L \) is the length of the suspension cable, and \( \tau \) is the length of the time step.

\[
\begin{align*}
\dot{p} &= \dot{x} \\
\dot{v} &= a \\
\dot{\eta} &= \dot{\phi}_L \\
\dot{\theta} &= \dot{\theta}_L \\
\end{align*}
\]
\[
v = v_0 + \tau \alpha; \quad p = p_0 + \tau v_0 + 0.5 \tau^2 a
\]
\[
\dot{\eta} = \dot{\eta}_0 + \tau \ddot{\eta}; \quad \eta = \eta_0 + \tau \dot{\eta}_0 + 0.5 \tau^2 \ddot{\eta}
\]
where
\[
\ddot{\eta} = \begin{bmatrix}
\sin \theta_0 \sin \phi_0 - \cos \phi_0 \cos \theta_0 \sin \phi_0 L^{-1} \\
- \cos \theta_0 \cos \phi_0 & 0 & \cos \phi_0 \sin \theta_0 L^{-1}
\end{bmatrix}(a - g')
\]

To learn the approximation of the state value function, AVI starts with an arbitrary vector \(\psi\). In each iteration, the state space is randomly sampled to produce a set of state samples \(S\). New estimate of the state value function is calculated according to \(V(s) = r(s) + \gamma \max_a \psi^T F(P(s, a))\) for all samples \(s \in S\). \(0 < \gamma < 1\) is a discount factor. A linear regression then finds a new value of \(\psi\) that fits the calculated estimates \(V(s)\) into quadric form \(\psi^T F(s)\). The process repeats until the maximum number of iterations is reached.

### B. Trajectory Generation

An approximated value function induces a greedy policy \(\pi\) that is used to generate the trajectory and control the vehicle. Given a state \(s\), policy \(\pi(s)\) returns an action \(a\). The policy is determined by \(\pi(s) = \arg\max_a (\psi^T F(P(s, a)))\), where \(P\) is the state transition function described in (2) and (3). When applied to the system the resulting action moves the system to the state associated with the highest estimated value. The algorithm starts with the initial state. Then it finds an action according to the policy \(\pi\). The action is used to transition to the next state. The process repeats until the goal is reached or when the trajectory exceeds a maximum number of steps.

### C. Analysis

The Proposition III.1 gives sufficient conditions that the value function approximation, action state space and system dynamics need to meet to guarantee a plan that leads to the goal state.

**Proposition III.1.** Let \(s_0\) be the goal state. If vector \(\psi\) is negative definite, and action space \(A\) maps state space such that \(\forall s \in S \setminus \{s_0\}, \exists a \in A \text{ that } V(\pi_A(s)) > \epsilon + V(s)\), for some \(\epsilon > 0\), then the system is asymptotically stable in the sense of Lyapunov, and coincidently for an arbitrary start state \(s \in S\), greedy policy with respect to \(V\) leads to the goal state \(s_0\). In other words, \(\forall s \in S, \exists n, \pi^n_A(s) = s_0\).

**Proof.** To show that the system is asymptotically stable, we need to find a discrete time control Lyapunov function \(W(s)\), such that \(W(s(k)) > 0\), for \(\forall s(k) \neq s_0\), \(W(s_0) = 0\), \(\Delta W(s(k)) = W(s(k + 1)) - W(s(k)) < 0\), and \(\Delta W(s_0) = 0\), for all \(k\), where \(s_0 = [0 0 0 0 0 0 0 0 0 0]^T\).

Let \(W(s) = -V(s) = -\psi^T (||s||^2, ||v||^2, ||\dot{\eta}||^2, ||\ddot{\eta}||^2)^T\). Then \(W(0) = 0\), and for all \(s \neq s_0\), \(W(s) > 0\), since \(\psi < 0\).

\(\Delta W(s(k)) = -(V(s(k + 1)) - V(s(k))) < 0\) because of the assumption that for each state there an action to takes the system to a state with a higher value.

Thus, \(W\) is a Lyapunov function with no constraints on \(s\), and is globally asymptotically stable. Therefore any policy following function \(W\) (or \(V\)) will lead the system to the unique equilibrium point.

Proposition III.1 connects state value approximation with Lyapunov stability theory. If \(V\) satisfies the criteria, the system is globally approximately stable. We empirically show that the criteria is met. Proposition III.1 requires that \(\psi\) is negative definite for the value function \(V\) described in (1) to have a unique maximum. As we will see in IV-B, the empirical results show that is the case. These observations lead to several practical properties of the induced greedy policy that we will verify empirically:

1) **The induced greedy policy is robust to some noise:** as long as there is a transition to a state with a higher value, an action could be taken and the goal will be attained, although not optimally. Section IV-B presents the empirical evidence for this property.

2) **The policy is agnostic to the simulator used:** The simulator defines the transition function and along with the action space defines the set of reachable states. Thus, as long as the conditions of Proposition III.1 are met, we can switch the simulators we use. This means that we can train on a simple simulator and generate a trajectory on a more sophisticated model that would predict the system better.

3) **Learning on the domain subset:** As we will show experimentally in Sections IV-B and IV-C, we can learn the model on a small subset of the state space around the goal state, and the resulting policy will work on the whole domain where the criteria above hold, i.e., where the value function doesn’t have other maxima. This property makes the method a good choice for a local planner.

4) **Changing action space:** Lastly, the action space between learning and the trajectory generation can change, and the algorithm will still produce a trajectory to the goal state. For example, to save computational time, we can learn on the smaller, more coarse discretization of the action space to obtain the value function parameters, and generate a trajectory on a more refined action space which produces a smoother trajectory. We will demonstrate this property during the multi-waypoint flight experiment.

## IV. RESULTS

In this section we verify the convergence of the proposed algorithm as well as its effectiveness in simulation and experiment. Section IV-A assesses the approximate value iteration convergence. Section IV-B shows the results of trajectory generation in simulation for the expanded state and action space. Lastly, Section IV-C presents results of experiments with the quadrotor in expanded state and action space. The experiments assess the discrepancy between the simulation and the actual swing encountered during the flight, and make a comparison between a cubic trajectory (trajectory where position is a 3\textsuperscript{rd} order polynomial function of time) and our method. Only the summary of the results is presented here. The detailed results with full implementation details are available in [4].

### A. Value Function Approximation Learning Results

We run AVI in two configurations: 2D and 3D (see Table I). Both configurations use the same discount parameter \(\gamma < 1\) to ensure that the value function is finite. The
The agent is trained in 3D configuration (see Table I). For trajectory generation, we use a fine-grain discretized 3D action space $A = (-3 : 0.05 : 3)^3$. This action space is ten times per dimension finer.

To assess how well a policy adapts to different starting positions, we choose one fixed position, and two randomly-drawn variable positions. One measures how well the agent performs within the sampling box. The rest of the positions are well outside of the sampling space used for the policy generation, and assess how well the method works for trajectories outside of the sampling bounds with an extended state space.

Table II presents the averaged results. With the exception of the noisy holonomic simulator at the starting position (-20,-20,15), all experiments complete the trajectory with the minimal load displacement.

The results show that trajectories generated under noisy conditions take a bit longer to reach the goal state, and the standard deviation associated with the results is a bit larger. This is expected, given the random nature of the noise. However, all of the noisy trajectories reach the goal with about the same accuracy as the non-noisy trajectories. This finding matches our prediction from Section III.

The maximum load displacement during its entire trajectory for all 100 trials inversely depends on the distance from the initial state to the goal state. For short trajectories within the sampling box, the swing always remains within $4^\circ$, while for the very long trajectories it could go up to $46^\circ$. This makes sense, given that the agent is minimizing the combination of the swing and distance. When very far away from the goal, the agent will move quickly towards the goal state and produce increased swing. Once the agent is closer to the goal state, the swing component becomes dominant in the value function, and the swing reduces.

C. Experimental Results

The experiments were performed using the MARHES multi-aerial vehicle testbed. This testbed and its real-time controller are described in detail in [8]. We first trained an agent in 2D configuration (see Table I). Once the agent was trained, we generated trajectories. To generate trajectories, we used a fine-grain discretized 3D action space. These trajectories were sent to the quadrotor with a suspended load. Figure 5 compares the vehicle and load trajectories for a learned trajectory as flown and in simulation, with cubic and DP trajectories of the same length and duration.

Comparison with Simulation: Looking at the load trajectories in Figure 5 (b), we notice the actual flown trajectory naturally contains more oscillations that the simulator didn’t model for. Despite that, the limits, boundaries, and the profile of the load trajectory are close between the simulation and the flown trajectory. This verifies the validity of the simulation results: the load trajectory predictions in the simulator are reasonably accurate.

Comparison with Cubic: Comparing the flown learned trajectory with a cubic trajectory, we see a different swing profile and significantly higher load oscillations. The most notable difference happens after the destination is reached during the hover (after 3.5 seconds in Figure 5 (b)). In this

![Fig. 4. Convergence of feature parameter vector $\psi$'s norm over 1000 iterations. The results are averaged over 100 trials. One and two standard deviations are shown. After initial learning phase, $\psi$ stabilizes to a constant value.](image-url)

#### TABLE I

**APPROXIMATE VALUE ITERATION ALGORITHM HYPERPARAMETERS.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>3D Configuration</th>
<th>2D Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>Min action</td>
<td>(-3, -3, -3)</td>
<td>(-3, -3, 0)</td>
</tr>
<tr>
<td>Max action</td>
<td>(3, 3, 3)</td>
<td>(3, 3, 0)</td>
</tr>
<tr>
<td>Action step</td>
<td>0.5</td>
<td>0.05</td>
</tr>
<tr>
<td>Min sampling space</td>
<td>$p = (-1, -1, -1)$, $v = (-3, -3, -3)$, $\eta = (-10^6, 10^6, 10^6)$, $\eta = (-10, -10)$</td>
<td></td>
</tr>
<tr>
<td>MAX sampling space</td>
<td>$p = (1, 1, 1)$, $v = (3, 3, 3)$, $\eta = (10^6, 10^6, 10^6)$, $\eta = (10, 10)$</td>
<td></td>
</tr>
<tr>
<td>Sampling</td>
<td>Linear</td>
<td>Constant (200)</td>
</tr>
<tr>
<td>Simulator</td>
<td>Holonomic</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>50Hz</td>
<td></td>
</tr>
<tr>
<td>Number of iterations</td>
<td>1000</td>
<td>800</td>
</tr>
<tr>
<td>Number of trials</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>Reward function</td>
<td>$c_1 = 10000$, $c_2 = 750$, $c_3 = 1$, $a_1 = 14$, $a_2 = 10000$, $\epsilon = 0.05$</td>
<td></td>
</tr>
</tbody>
</table>

The 3D configuration trains the agent with a coarse three-dimensional action vector. Each direction of the linear acceleration is discretized in 13 steps, resulting in $13^3$ total actions. In this phase of the algorithm we are shaping the value function, and this level of coarseness is sufficient.

To assess the stability of the approximate value iteration, we ran that AVI 100 times, for 1,000 iterations in the 3D configuration. Figure 4 shows the trend of the norm of value parameter vector $\psi$ with respect to $L_2$ norm. We can see that the $\|\psi\|$ stabilizes after about 200 iterations. The empirical results show that the algorithm is stable and produces a consistent policy over different trials.

B. Simulation Results

We access the quality and robustness of a trained agent in simulation by generating trajectories from different distances for two different simulators. The first simulator is a generic holonomic aerial vehicle with suspended load simulator, the same simulator we used in the learning phase. The second simulator is a noisy holonomic aerial vehicle simulator, which adds up to 5% uniform noise to the predicted state. Its intent is to simulate the inaccuracies and uncertainties of the real hardware. We compare the performance of our learned generated trajectories with model-based dynamic programming (DP) and cubic trajectories.
part of the trajectory, the cubic trajectory shows a load swing of $5 - 12^\circ$, while the learned trajectory controls the swing to under $4^\circ$.

**Comparison with DP:** Figure 5 (b) shows that load for the trajectory learned with reinforcement learning stays within the load trajectory generated using dynamic programming at all times: during the flight (the first 3.4 seconds) and the residual oscillation after the flight.

![Fig. 5. Quadrotor (a) and load (b) trajectories as flown, created through learning compared to cubic, dynamic programming, and simulated trajectories.](image)

### V. Conclusions

In this work, we presented a motion planning framework for producing trajectories with minimal residual oscillations for a rotorcraft UAV with a freely suspended load. The framework relies on reinforcement learning to learn the problem characteristics for a particular load. We found conditions that if met allow the learned agent to be applied to produce a wide variety of trajectories. We discussed the learning convergence, assessed the produced motion plans in simulation, and their robustness to noise. Lastly, we implemented the proposed algorithm on a quadrotor type UAV in order to demonstrate its feasibility and to assess the accuracy of the simulation results.

### Acknowledgements

This work was in part supported by NSF grant ECCS #1027775 and the Department of Energy URPR Grant #DE-FG52-04NA25590. Lydia Tapia is supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. Ivana Palunko is supported by the European Community’s Seventh Framework Programme under grant No. 285939 (ACROSS).

### References