Autonomous inverted helicopter flight via reinforcement learning

Andrew Y. Ng[1], Adam Coates[1], Mark Diel[2], Varun Ganapathi[1], Jamie Schulte[1], Ben Tse[2], Eric Berger[1], and Eric Liang[1]

[1] Computer Science Department, Stanford University, Stanford, CA 94305
[2] Whirled Air Helicopters, Menlo Park, CA 94025

Abstract. Helicopters have highly stochastic, nonlinear, dynamics, and autonomous helicopter flight is widely regarded to be a challenging control problem. As helicopters are highly unstable at low speeds, it is particularly difficult to design controllers for low speed aerobatic maneuvers. In this paper, we describe a successful application of reinforcement learning to designing a controller for sustained inverted flight on an autonomous helicopter. Using data collected from the helicopter in flight, we began by learning a stochastic, nonlinear model of the helicopter's dynamics. Then, a reinforcement learning algorithm was applied to automatically learn a controller for autonomous inverted hovering. Finally, the resulting controller was successfully tested on our autonomous helicopter platform.

1 Introduction

Autonomous helicopter flight represents a challenging control problem with high dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics, and helicopters are widely regarded to be significantly harder to control than fixed-wing aircraft. [3,10] But helicopters are uniquely suited to many applications requiring either low-speed flight or stable hovering. The control of autonomous helicopters thus provides an important and challenging testbed for learning and control algorithms.

Some recent examples of successful autonomous helicopter flight are given in [7,2,9,8]. Because helicopter flight is usually open-loop stable at high speeds but unstable at low speeds, we believe low-speed helicopter maneuvers are particularly interesting and challenging. In previous work, (Ng et al.,2004) considered the problem of learning to fly low-speed maneuvers very accurately. In this paper, we describe a successful application of machine learning to performing a simple low-speed aerobatic maneuver—autonomous *sustained* inverted hovering.

2 Helicopter platform

To carry out flight experiments, we began by instrumenting a Bergen industrial twin helicopter (length 59", height 22") for autonomous flight. This

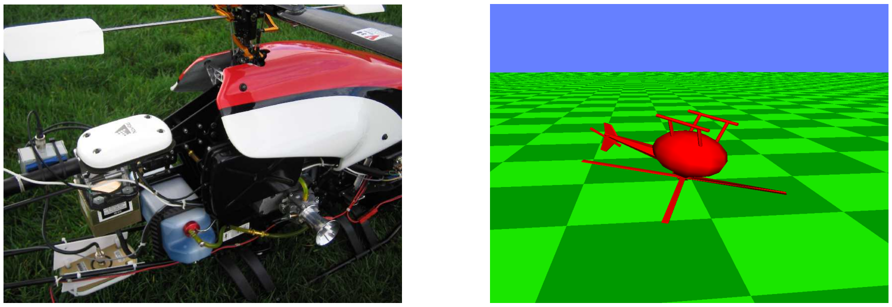
Fig. 1. Helicopter in configuration for upright-only flight (single GPS antenna).

helicopter is powered by a twin cylinder 46cc engine, and has an unloaded weight of 18 lbs.

Our initial flight tests indicated that the Bergen industrial twin's original rotor-head was unlikely to be sufficiently strong to withstand the forces encountered in aerobatic maneuvers. We therefore replaced the rotor-head with one from an X-Cell 60 helicopter. We also instrumented the helicopter with a PC104 flight computer, an Inertial Science ISIS-IMU (accelerometers and turning-rate gyroscopes), a Novatel GPS unit, and a MicroStrain 3d magnetic compass. The PC104 was mounted in a plastic enclosure at the nose of the helicopter, and the GPS antenna, IMU, and magnetic compass were mounted on the tail boom. The IMU in particular was mounted fairly close to the fuselage, to minimize measurement noise arising from tail-boom vibrations. The fuel tank, originally mounted at the nose, was also moved to the rear. Figure 1 shows our helicopter in this initial instrumented configuration.

Readings from all the sensors are fed to the onboard PC104 flight computer, which runs a Kalman filter to obtain position and orientation estimates for the helicopter at 100Hz. A custom takeover board also allows the computer either to read the human pilot's commands that are being sent to the helicopter control surfaces, or to send its own commands to the helicopter. The onboard computer also communicates with a ground station via 802.11b wireless.

Most GPS antenna (particularly differential, L1/L2 ones) are directional, and a single antenna pointing upwards relative to the helicopter would be unable to see any satellites if the helicopter is inverted. Thus, a single, upward-pointing antenna cannot be used to localize the helicopter in inverted flight. We therefore added to our system a second antenna facing downwards, and used a computer-controlled relay for switching between them. By examining the Kalman filter output, our onboard computer automatically selects the upward-facing antenna. (See Figure 2a.) We also tried a system in which



(a) (b)

Fig. 2. (a) Dual GPS antenna configuration (one antenna is mounted on the tail-boom facing up; the other is shown facing down in the lower-left corner of the picture). The small box on the left side of the picture (mounted on the left side of the tail-boom) is a computer-controlled relay. (b) Graphical simulator of helicopter, built using the learned helicopter dynamics.

the two antenna were simultaneously connected to the receiver via a Y-cable (without a relay). In our experiments, this suffered from significant GPS multipath problems and was not usable.

3 Machine learning for controller design

A helicopter such as ours has a high center of gravity when in inverted hover, making inverted flight significantly less stable than upright flight (which is also unstable at low speeds). Indeed, there are far more human RC pilots who can perform high-speed aerobatic maneuvers than can keep a helicopter in sustained inverted hover. Thus, designing a stable controller for sustained inverted flight appears to be a difficult control problem.

Most helicopters are flown using four controls:

- $a[1]$ and $a[2]$: The longitudinal (front-back) and latitudinal (left-right) cyclic pitch controls cause the helicopter to pitch forward/backwards or sideways, and can thereby also be used to affect acceleration in the longitudinal and latitudinal directions.
- $a[3]$: The main rotor collective pitch control causes the main rotor blades to rotate along an axis that runs along the length of the rotor blade, and thereby affects the angle at which the main rotor's blades are tilted relative to the plane of rotation. As the main rotor blades sweep through the air, they generate an amount of upward thrust that (generally) increases with this angle. By varying the collective pitch angle, we can affect the main rotor's thrust. For inverted flight, by setting a negative collective pitch angle, we can cause the helicopter to produce negative thrust.
- $a[4]$: The tail rotor collective pitch control affects tail rotor thrust, and can be used to yaw (turn) the helicopter.

A fifth control, the throttle, is commanded as pre-set function of the main rotor collective pitch, and can safely be ignored for the rest of this paper.

To design the controller for our helicopter, we began by learning a stochastic, nonlinear, model of the helicopter dynamics. Then, a reinforcement learning/policy search algorithm was used to automatically design a controller.

3.1 Model identification

We applied supervised learning to identify a model of the helicopter's dynamics. We began by asking a human pilot to fly the helicopter upside-down, and logged the pilot commands and helicopter state s comprising its position (x, y, z), orientation (roll ϕ, pitch θ, yaw ω), velocity $(\dot{x}, \dot{y}, \dot{z})$ and angular velocities $(\dot{\phi}, \dot{\theta}, \dot{\omega})$. A total of 391s of flight data was collected for model identification. Our goal was to learn a model that, given the state s_t and the action a_t commanded by the pilot at time t, would give a good estimate of the probability distribution $P_{s_t a_t}(s_{t+1})$ of the resulting state of the helicopter s_{t+1} one time step later.

Following standard practice in system identification [4], we converted the original 12-dimensional helicopter state into a reduced 8-dimensional state represented in body coordinates $s^b = [\phi, \theta, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\omega}]$. Where there is risk of confusion, we will use superscript s and b to distinguish between spatial (world) coordinates and body coordinates. The body coordinate representation specifies the helicopter state using a coordinate frame in which the x, y, and z axes are forwards, sideways, and down relative to the current orientation of the helicopter, instead of north, east and down. Thus, $\dot{x}^b$ is the forward velocity, whereas $\dot{x}^s$ is the velocity in the northern direction. (ϕ and θ are always expressed in world coordinates, because roll and pitch relative to the body coordinate frame is always zero.) By using a body coordinate representation, we encode into our model certain "symmetries" of helicopter flight, such as that the helicopter's dynamics are the same regardless of its absolute position and orientation (assuming the absence of obstacles).[1]

Even in the reduced coordinate representation, only a subset of the state variables need to be modeled explicitly using learning. Specifically, the roll ϕ and pitch θ (and yaw ω) angles of the helicopter over time can be computed exactly as a function of the roll rate $\dot{\phi}$, pitch rate $\dot{\theta}$ and yaw rate $\dot{\omega}$. Thus, given a model that predicts only the angular velocities, we can numerically integrate the velocities over time to obtain orientations.

We identified our model at 10Hz, so that the difference in time between s_t and s_{t+1} was 0.1 seconds. We used linear regression to learn to predict, given

[1] Actually, by handling the effects of gravity explicitly, it is possible to obtain an even better model that uses a further reduced, 6-dimensional, state, by eliminating the state variables ϕ and θ. We found this additional reduction useful and included it in the final version of our model; however, a full discussion is beyond the scope of this paper.

$s_t^b \in \mathbb{R}^8$ and $a_t \in \mathbb{R}^4$, a sub-vector of the state variables at the next timestep $[\dot{x}_{t+1}^b, \dot{y}_{t+1}^b, \dot{z}_{t+1}^b, \dot{\phi}_{t+1}^b, \dot{\theta}_{t+1}^b, \dot{\omega}_{t+1}^b]$. This body coordinate model is then converted back into a world coordinates model, for example by integrating angular velocities to obtain world coordinate angles. Note that because the process of integrating angular velocities expressed in body coordinates to obtain angles expressed in world coordinates is nonlinear, the final model resulting from this process is also necessarily nonlinear. After recovering the world coordinate orientations via integration, it is also straightforward to obtain the rest of the world coordinates state. (For example, the mapping from body coordinate velocity to world coordinate velocity is simply a rotation.)

Lastly, because helicopter dynamics are inherently stochastic, a deterministic model would be unlikely to fully capture a helicopter's range of possible behaviors. We modeled the errors in the one-step predictions of our model as Gaussian, and estimated the magnitude of the noise variance via maximum likelihood.

The result of this procedure is a stochastic, nonlinear model of our helicopter's dynamics. To verify the learned model, we also implemented a graphical simulator (see Figure 2b) with a joystick control interface similar to that on the real helicopter. This allows the pilot to fly the helicopter in simulation and verify the simulator's modeled dynamics. The same graphical simulator was subsequently also used for controller visualization and testing.

3.2 Controller design via reinforcement learning

Having built a model/simulator of the helicopter, we then applied reinforcement learning to learn a good controller.

Reinforcement learning [11] gives a set of tools for solving control problems posed in the Markov decision process (MDP) formalism. An MDP is a tuple $(S, s_0, A, \{P_{sa}\}, \gamma, R)$. In our problem, S is the set of states (expressed in world coordinates) comprising all possible helicopter positions, orientations, velocities and angular velocities; $s_0 \in S$ is the initial state; $A = [-1, 1]^4$ is the set of all possible control actions; $P_{sa}(\cdot)$ are the state transition probabilities for taking action a in state s; $\gamma \in [0, 1)$ is a discount factor; and $R : S \mapsto \mathbb{R}$ is a reward function. The dynamics of an MDP proceed as follows: The system is first initialized in state s_0. Based on the initial state, we get to choose some control action $a_0 \in A$. As a result of our choice, the system transitions randomly to some new state s_1 according to the state transition probabilities $P_{s_0 a_0}(\cdot)$. We then get to pick a new action a_1, as a result of which the system transitions to $s_2 \sim P_{s_1 a_1}$, and so on.

A function $\pi : S \mapsto A$ is called a policy (or controller). It we take action $\pi(s)$ whenever we are in state s, then we say that we are acting according to π. The reward function R indicates how well we are doing at any particular time, and the goal of the reinforcement learning algorithm is to find a policy

π so as to maximize

$$U(\pi) \doteq \mathrm{E}_{s_0,s_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t) \mid \pi\right], \tag{1}$$

where the expectation is over the random sequence of states visited by acting according to π, starting from state s_0. Because $\gamma < 1$, rewards in the distant future are automatically given less weight in the sum above.

For the problem of autonomous hovering, we used a quadratic reward function

$$\begin{aligned} R(s^s) = -(&\alpha_x(x-x^*)^2 + \alpha_y(y-y^*)^2 + \alpha_z(z-z^*)^2 \\ &+\alpha_{\dot{x}}\dot{x}^2 + \alpha_{\dot{y}}\dot{y}^2 + \alpha_{\dot{z}}\dot{z}^2 + \alpha_\omega(\omega-\omega^*)^2), \end{aligned} \tag{2}$$

where the position (x^*, y^*, z^*) and orientation ω^* specifies where we want the helicopter to hover. (The term $\omega - \omega^*$, which is a difference between two angles, is computed with appropriate wrapping around 2π.) The coefficients α_i were chosen to roughly scale each of the terms in (2) to the same order of magnitude (a standard heuristic in LQR control [1]). Note that our reward function did not penalize deviations from zero roll and pitch, because a helicopter hovering stably in place typically has to be tilted slightly.[2]

For the policy π, we chose as our representation a simplified version of the neural network used in [7]. Specifically, the longitudinal cyclic pitch $a[1]$ was commanded as a function of $x^b - x^{*b}$ (error in position in the x direction, expressed in body coordinates), $\dot{x}^b$, and pitch θ; the latitudinal cyclic pitch $a[2]$ was commanded as a function of $y^b - y^{*b}$, $\dot{y}^b$ and roll ϕ; the main rotor collective pitch $a[3]$ was commanded as a function of $z^b - z^{*b}$ and $\dot{z}^b$; and the tail rotor collective pitch $a[4]$ was commanded as a function of $\omega - \omega^*$.[3] Thus, the learning problem was to choose the gains for the controller so that we obtain a policy π with large $U(\pi)$.

Given a particular policy π, computing $U(\pi)$ exactly would require taking an expectation over a complex distribution over state sequences (Equation 1). For nonlinear, stochastic, MDPs, it is in general intractable to exactly compute this expectation. However, given a simulator for the MDP, we can approximate this expectation via Monte Carlo. Specifically, in our application, the learned model described in Section 3.1 can be used to sample $s_{t+1} \sim P_{s_t a_t}$

[2] For example, the tail rotor generates a sideways force that would tend to cause the helicopter to drift sideways if the helicopter were perfectly level. This sideways force is counteracted by having the helicopter tilted slightly in the opposite direction, so that the main rotor generates a slight sideways force in an opposite direction to that generated by the tail rotor, in addition to an upwards force.

[3] Actually, we found that a refinement of this representation worked slightly better. Specifically, rather than expressing the position and velocity errors in the body coordinate frame, we instead expressed them in a coordinate frame whose x and y axes lie in the horizontal plane/parallel to the ground, and whose x axis has the same yaw angle as the helicopter.

for any state action pair s_t, a_t. Thus, by sampling $s_1 \sim P_{s_0\pi(s_0)}$, $s_2 \sim P_{s_1\pi(s_1)}$, $\ldots$, we obtain a random state sequence $s_0, s_1, s_2, \ldots$ drawn from the distribution resulting from flying the helicopter (in simulation) using controller π. By summing up $\sum_{t=0}^{\infty} \gamma^t R(s_t)$, we obtain one "sample" with which to estimate $U(\pi)$.[4] More generally, we can repeat this entire process m times, and average to obtain an estimate $\hat{U}(\pi)$ of $U(\pi)$.

One can now try to search for π that optimizes $\hat{U}(\pi)$. Unfortunately, optimizing $\hat{U}(\pi)$ represents a difficult stochastic optimization problem. Each evaluation of $\hat{U}(\pi)$ is defined via a random Monte Carlo procedure, so multiple evaluations of $\hat{U}(\pi)$ for even the same π will in general give back slightly different, noisy, answers. This makes it difficult to find "$\arg\max_\pi \hat{U}(\pi)$" using standard search algorithms. But using the PEGASUS method (Ng and Jordan, 2000), we can turn this stochastic optimization problem into an ordinary deterministic problem, so that any standard search algorithm can now be applied. Specifically, the computation of $\hat{U}(\pi)$ makes multiple calls to the helicopter dynamical simulator, which in turn makes multiple calls to a random number generator to generate the samples $s_{t+1} \sim P_{s_t a_t}$. If we fix in advance the sequence of random numbers used by the simulator, then there is no longer any randomness in the evaluation of $\hat{U}(\pi)$, and in particular finding $\max_\pi \hat{U}(\pi)$ involves only solving a standard, deterministic, optimization problem. (For more details, see [6], which also proves that the "sample complexity"—i.e., the number of Monte Carlo samples m we need to average over in order to obtain an accurate approximation—is at most polynomial in all quantities of interest.) To find a good controller, we therefore applied a greedy hillclimbing algorithm (coordinate ascent) to search for a policy π with large $\hat{U}(\pi)$.

We note that in earlier work, (Ng et al., 2004) also used a similar approach to learn to fly expert-league RC helicopter competition maneuvers, including a nose-in circle (where the helicopter is flown in a circle, but with the nose of the helicopter continuously pointed at the center of rotation) and other maneuvers.

4 Experimental Results

Using the reinforcement learning approach described in Section 3, we found that we were able to extremely quickly design new controllers for the helicopter. We first completed the inverted flight hardware and collected (human pilot) flight data on 3rd Dec 2003. Using reinforcement learning, we completed our controller design by 5th Dec. In our flight experiment on 6th Dec, we successfully demonstrated our controller on the hardware platform by having a human pilot first take off and flip the helicopter upside down, immediately

[4] In practice, we truncate the state sequence after a large but finite number of steps. Because of discounting, this introduces at most a small error into the approximation.

Fig. 3. Helicopter in autonomous sustained inverted hover.

after which our controller took over and was able to keep the helicopter in stable, sustained inverted flight. Once the helicopter hardware for inverted flight was completed, building on our pre-existing software (implemented for upright flight only), the total time to design and demonstrate a stable inverted flight controller was less than 72 hours, including the time needed to write new learning software.

A picture of the helicopter in sustained autonomous hover is shown in Figure 3. To our knowledge, this is the first helicopter capable of sustained inverted flight under computer control. A video of the helicopter in inverted autonomous flight is also at

`http://www.cs.stanford.edu/~ang/rl-videos/`

Other videos, such as of a learned controller flying the competition maneuvers mentioned earlier, are also available at the url above.

5 Conclusions

In this paper, we described a successful application of reinforcement learning to the problem of designing a controller for autonomous inverted flight on a helicopter. Although not the focus of this paper, we also note that, using controllers designed via reinforcement learning and shaping [5], our helicopter is also capable of normal (upright) flight, including hovering and waypoint following.

We also found that a side benefit of being able to automatically learn new controllers quickly and with very little human effort is that it becomes significantly easier to rapidly reconfigure the helicopter for different flight applications. For example, we frequently change the helicopter's configuration (such as replacing the tail rotor assembly with a new, improved one) or payload (such as mounting or removing sensor payloads, additional computers, etc.). These modifications significantly change the dynamics of the helicopter, by affecting its mass, center of gravity, and responses to the controls. But by using our existing learning software, it has proved generally quite easy to quickly design a new controller for the helicopter after each time it is reconfigured.

Acknowledgments

We give warm thanks to Sebastian Thrun for his assistance and advice on this project, to Jin Kim for helpful discussions, and to Perry Kavros for his help constructing the helicopter. This work was supported by DARPA under contract number N66001-01-C-6018.

References

1. B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall, 1989.
2. J. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Int'l Conf. Robotics and Automation*. IEEE, 2001.
3. J. Leishman. *Principles of Helicopter Aerodynamics*. Cambridge Univ. Press, 2000.
4. B. Mettler, M. Tischler, and T. Kanade. System identification of small-size unmanned helicopter dynamics. In *American Helicopter Society, 55th Forum*, 1999.
5. Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, Bled, Slovenia, July 1999. Morgan Kaufmann.
6. Andrew Y. Ng and Michael I. Jordan. Pegasus: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intellicence, Proceedings of Sixteenth Conference*, pages 406–415, 2000.
7. Andrew Y. Ng, H. Jin Kim, Michael Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Neural Information Processing Systems 16*, 2004.
8. Jonathan M. Roberts, Peter I. Corke, and Gregg Buskey. Low-cost flight control system for a small autonomous helicopter. In *IEEE International Conference on Robotics and Automation*, 2003.
9. T. Schouwenaars, B. Mettler, E. Feron, and J. How. Hybrid architecture for full-envelope autonomous rotorcraft guidance. In *American Helicopter Society 59th Annual Forum*, 2003.

10. J. Seddon. *Basic Helicopter Aerodynamics*. AIAA Education Series. America Institute of Aeronautics and Astronautics, 1990.
11. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.