Autonomous Quadrotor Control with Reinforcement Learning

Michael C. Koval
mkoval@cs.rutgers.edu

Christopher R. Mansley
cmansley@cs.rutgers.edu

Michael L. Littman
mlittman@cs.rutgers.edu

Abstract

Based on the same principles as a single-rotor helicopter, a quadrotor is a flying vehicle that is propelled by four horizontal blades surrounding a central chassis. Because of this vehicle's symmetry and propulsion mechanism, a quadrotor is capable of simultaneously moving and steering by simple modulation of motor speeds [1]. This stability and relative simplicity makes quadrotors ideal for research in the application of control theory and artificial intelligence to aerial robotics [3].

Most prior work using quadrotors has applied low-level, manually-tuned control algorithms to complete specific tasks. This paper proposes an alternate approach for controlling a quadrotor through the application of continuous state-action space reinforcement learning algorithms by making use of the Parrot AR.Drone's rich suite of on-board sensors and the localization accuracy of the Vicon motion tracking system. With such high quality state information a reinforcement learning algorithm should be capable of quickly learning a policy that maps the quadrotor's physical state to the low level velocity parameters that are used to control a quadrotors's four motors. Once learning is complete, this policy will encode the information necessary to repeatably and accurately perform the desired high-level action without ever requiring a programmer to manually split the action into smaller components.

1 Introduction

Programming a complex robotic system has typically required a large amount of time from an interdisciplinary team of computer scientists, electrical engineers, and control theorists. This broad set of skills is required largely because a single high-level action (such as "catch the ball") requires thousands of decisions at different points in time based on the robot's sensor input and position. Reinforcement learning promises to simplify the otherwise time-consuming process of programming a robot's desired behavior by allowing the programmer to specify *what* action the robot should perform without ever detailing *how* it should perform the action [7]. This abstraction is especially beneficial when controlling a robot with multiple actuators such as aerial robot moving through a dynamic environment.

Quadrotor helicopters are one form of aerial vehicles that offer an excellent balance between maneuverability and stability. Similar in design to a traditional helicopter, a quadrotor is supported by four horizontally-oriented blades surrounding a central chassis that houses the vehicle's electronics and, potentially, its payload. By using four horizontal blades instead of one horizontal blade and one vertical blade, a quadrotor is capable of moving, strafing, and turning in three-dimensional space by altering the relative speeds of its rotors [1]. The specific methods of modulating motor outputs to achieve this type of motion is discussed in Section 4.

Even with its relatively simple design and relative stability, applying reinforcement learning to quadrotor control is a non-trivial problem. Unlike the discrete problems considered introductory reinforcement learning texts, a quadrotor's state is a function of its position, velocity, and acceleration: continuous variables that do not lend themselves to quantization. Similarly, the robot's actions are formed from a continuum of possible motor outputs. Moving from a discrete state space to a continuous state space greatly

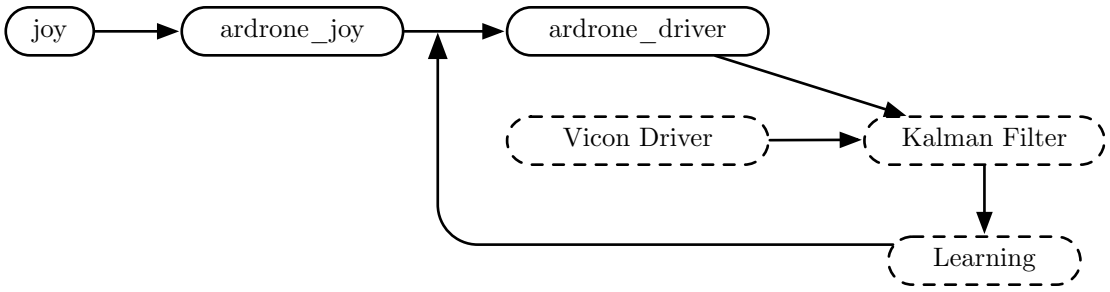


Figure 1: Architecture of the ROS nodes used in this project. Broken lines indicate unimplemented features.

increases the difficulty in learning an optimal policy and requires that one either quantize the state space or use a reinforcement learning algorithm designed to learn in continuous state-action spaces [6].

Regardless of the specific reinforcement learning algorithm selected, learning a mapping between states and actions is only as effective as the robot's state information. Because the quadrotor operates at high speeds and some tasks require fast response times, it is important to have a high-fidelity and high-sample rate for the quadrotor's state. This paper proposes that sufficiently high-quality data can be obtained by fusing data measured from onboard inertial sensors with the high-accuracy localization provided by a Vicon Motion Capture System.

2 Reinforcement Learning

Reinforcement learning is a subfield of machine learning in which an agent must learn an optimal behavior by interacting and receiving feedback from a stochastic environment. Unlike supervised learning where the agent is given knowledge as for *how* it should perform a task, reinforcement learning uses reward and punishment to teach the agent *what* the task is [2]. This approach is particularly useful when the agent is interacting with a large or highly dynamic environment that cannot be completely explored, such as when a robot is operating in a previously unknown environment [7].

In the standard model of reinforcement learning the environment is modeled as a Markov Decision Process (MDP) with a discrete set of states, actions, and a transition model. Using this model, the discrete set of *states* is connected by a collection of *actions*. In each state the agent performs one action, stochastically transitions to a new state, and receives a *reward* as dictated by the environment. Defined using this framework, the goal of reinforcement learning can be formally expressed as solving for a policy $\pi : S \rightarrow A(s)$ that, when followed, maximizes the agent's expected cumulative reward [8].

Traditionally, reinforcement learning algorithms have been categorized by whether they learn the transition model of the environment as an intermediate step towards learning the optimal policy. *Model-based* methods use the agent's interaction with the environment to first learn a model of the environment, then use that model to derive an optimal policy. Conversely, *model-free* algorithms directly learn the value of each state-action pair without ever learning an explicit model of the environment [2].

3 Robot Operating System

Willow Garage's Robot Operating System, ROS, is a framework for finding, building, and seamlessly running robot control code across multiple computers. In its essence, ROS is a common API for publishing and subscribing to data in a standard format [5]. Standardizing the format for data transmission and the interaction between nodes has allowed ROS to accumulate a large repository of drivers, libraries, and robotics al-

gorithms. In addition to granting access to already written code, this framework is a useful way of formally defining the interactions between the various tasks discussed in this paper (see Figure 1).

Interfacing directly with AR.Drone over 802.11 wifi, a modified version of Brown University's `ardrone_driver` is responsible for converting ROS messages to and from Parrot's custom packet format. Most importantly, this is the node that allows ROS to control the drone by specifying its desired linear and angular velocities. In addition to control, this node converts the drone's raw sensor readings into standard ROS messages that can easily be used and interpreted by the other ROS nodes involved in the project. See Section 4.2 for a detailed discussion of integration of Parrot's official SDK into a ROS node for communication with the quadrotor.

Because these sensors return measurements relative to the quadrotor's coordinate frame, they are unable to accurately localize the quadrotor relative to a global coordinate frame. This information is crucial for building an accurate representation of the robot's state and is a prerequisite for effective learning to occur. This limitation demands that the quadrotor's position be measured by a suite of external sensors, each of which has a known position relative to the world. Because of its mature software framework, high accuracy, and unparalleled sample rate of 125 Hz, the Vicon Motion Tracking System was selected for this task.

While the Vicon system provides an extremely accurate measurement of the drone's spatial coordinates, it can be further refined using the inertial measurements collected by the drone's onboard sensors (see Section 4.1). Effective fusion of these two estimates of the drone's position will be achieved through direct application of an enhanced Kalman filter [9]. This filtered output can then be directly used as state information for a reinforcement learning algorithm as previously discussed in Section 2.

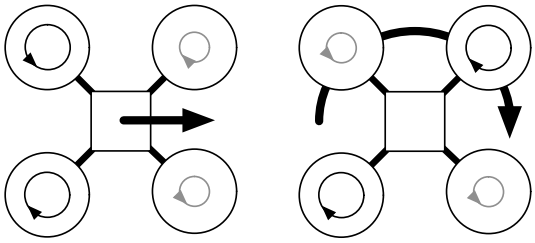
Figure 2: Parrot AR.Drone quadrotor in flight.

4 Parrot AR.Drone

The AR.Drone is an inexpensive, mass-produced quadrotor helicopter designed and manufactured for consumer use by Parrot Corporation. As discussed in Section 1, a quadrotor features four horizontally-oriented rotors evenly spaced around a central chassis and is much more stable than traditional flying vehicles. Modulating the relative speeds of a quadrotor's four motors adjusts the net force and torque applied to the robot and allows it to move through space.

To remain in a stationary hover the net force and torque on the quadrotor must both be zero. This is achieved by driving the pairs of opposite motors in the same direction, or equivalently, driving adjacent motors with equal speeds in opposite directions. Any deviation from this stable configuration will cause translation or rotation to occur in the plane of the quadrotor.

Slowing two adjacent rotors by equal amounts, as one would expect, causes the net force on the robot to increase and the torque to remain zero (Figure 3a). This configuration allows the drone to make translational motion with no rotation about the drone's center. Conversely, decreasing the speed of two opposing motors causes net force on the drone to remain zero while perturbing the net torque. This, as expected, causes the drone to rotate about its center with zero translational velocity (Figure 3b). More complex modulations of motor speed allow one to induce simultaneous translation and rotation by combining these two primitive actions. [1].



(a) Translational Motion (b) In-Place Rotation

Figure 3: Movement through motor speed modulation. Smaller arrows indicate slower motor speeds.

4.1 Sensing Capabilities

Despite the quadrotor's conceptual simplicity, modulating four motor speeds to control the vehicle's motion is too difficult for direct human teleoperation. To assist the driver in managing this complexity, the AR.Drone features an array of sensors that are used as the input to on-board control algorithms. These algorithms translate commands in the form of rotational and translation velocities into modulated motor's speeds. To assist the driver in this manner, the following sensors are present on the AR.Drone: [4]

1. Two Ultrasonic Altitude Sensors
2. Forward-Facing Camera[1]
3. Downward-Facing Camera[2]
4. Six DoF Inertial Measurement Unit (IMU),

Reading the IMU yields a gyroscope measurement of angular velocity, an accelerometer measurement of linear acceleration, and a composite estimate of the unit's pose. Similarly, directly reading from the drone's downward-facing ultrasonic range sensors yields the drone's altitude and applying optical flow techniques to the downward-facing camera produces an accurate measurement of translational velocity. Note, as discussed in Section 3, all of these measurements are relative to the robot's coordinate frame and provide no information about the global reference frame.

4.2 ROS Integration

While there is not a complete ROS node available that publishes all of the AR.Drone's internal sensor readings as ROS topics, a research group at Brown University developed a thin wrapper for Parrot's official software development kit that allows for basic teleoperation of the AR.Drone through standard ROS topic and services.

This node was extended to publish the full breadth of sensor data discussed in Section 4.1 as standard ROS messages such a `Pose`, `Twist`, and `Imu`[3]. Using multiple standard messages instead of a single custom message improves the re-usability of the `ardrone_driver` node at the cost of a slightly increased overhead. Using this convention, the modified ROS node publishes the following topics:

- `battery`: battery life as a percentage
- `imu`[4]: orientation, linear acceleration, and angular velocity
- `pose`: orientation and altitude
- `twist`: linear and angular velocity
- `image_raw`: forward-facing camera

All of these messages are designed to be conformant to REP-103, the ROS Enhancement Proposal (REP) that defines the unit and coordinate frame standards that the built-in ROS nodes follow. Specifically, this REP states that distance is expressed in meters, velocity in m/s, acceleration in m/s^2, and Euclidean rotations as quaternions. All of these measurements are expressed relative to a right-handed coordinate frame fixed to the origin of the robot where the positive x-axis is forward, the positive y-axis is to the left, and the positive z-axis is up when in its resting orientation.

[1] 640×480 pixels, 93° wide-angle lens, 15 Hz frequency
[2] 167×144 pixels, 64° wide-angle lens, 60 Hz frequency
[3] Message in the standard `sensor_msgs` node.
[4] Topic used to publish `Imu` messages.

4.3 Control Algorithms

Using its full repertoire of on-board sensors, the AR.Drone uses a suite of control algorithms to keep itself in a stable hover. These control algorithms use the optical flow velocities calculated from the downward-facing camera to minimize the drone's translational velocity and, theoretically, cause it to hover over a fixed point relative to the ground. Empirically these control algorithms are fairly effective at keeping the drone at a constant altitude, but less effective at stopping translational drift.

This is quantitatively confirmed with the optical flow velocity data plotted in Figure 4. In this plot the take-off and landing occur, respectively, at 3 and 9 seconds and the time in-between represents an unaltered hover. During the time in flight, the controls were not touched by a human operator and the drone's baseline drift has been established. The same drift qualitatively observed above is visible in the plot by a large negative bias visible in the drone's forward velocity. Because this drift is visible by both an outside observer and in the optical flow velocities, a reinforcement-learning algorithm or improved control algorithm should be capable for correcting for this drift when controlling the quadrotor.

5 Conclusion

Quadrotor helicopters, as discussed in Section 1, are a much more stable platform for artificial intelligence research than traditional helicopters and planes. This stability has been exploited by control theorists to obtain impressive results in specific situations [3], but this research does not simplify the process of programming the quadrotor to respond in novel situations. This learning process closely matches the goal of reinforcement learning described in Section 2: to train an agent by telling it *what* the desired goal is instead of *how* to reach it [2].

Thanks to a large library of pre-existing code available and the common message-passing interface specified by ROS, the experimental setup for reinforcement learning requires a minimal amount of custom software. In particular, Brown University's `ardrone_driver` node provide a ROS interface to Parrot's official SDK and allows for common ROS messages to control the drone. With the addition of custom modifications (discussed in Section 4.2), this node also publishes the data collected from the Drone's cameras, distance sensors, and the IMU. Finally, the inertial data returned by this node will be combined with the position data returned by the Vicon Motion Tracking system using an enhanced Kalman filter and used to obtain an extremely accurate estimate of the drone's state [9].

Using this information as the drone's state and its motor outputs as actions, the problem of training the quadrotor to perform a high-level action is in the same form as the MDPs discussed in Section 2. Future work will focus on using a continuous state-action reinforcement learning algorithm to learn complex behavior more quickly than would be possible through manual programming.

6 Future Work

Because reinforcement learning algorithms are only as good as the accuracy and timeliness their knowledge of the world, it is first necessary to obtain accurate measurements of the drone's position, velocity, and acceleration in three-dimensional space. While the customized ROS node discussed in Section 4.2 provides information about the drone's velocity and acceleration, it is unable to provide information about the drone's spatial coordinates relative to a fixed reference frame. Using the ROS node for the Vicon Motion Tracking System developed by researchers at the University of California, Berkeley, the next step towards having a fully operational experimental setup is to set up and calibrate the Vicon motion tracking system to track the drone's position and orientation.

This measured pose will then be combined with the drone's on-board sensors to produce a single estimate of the quadrotor's position, orientation, velocity and acceleration with an enhanced Kalman filter [9]. This vector of data (or, perhaps, a subset of it) will then

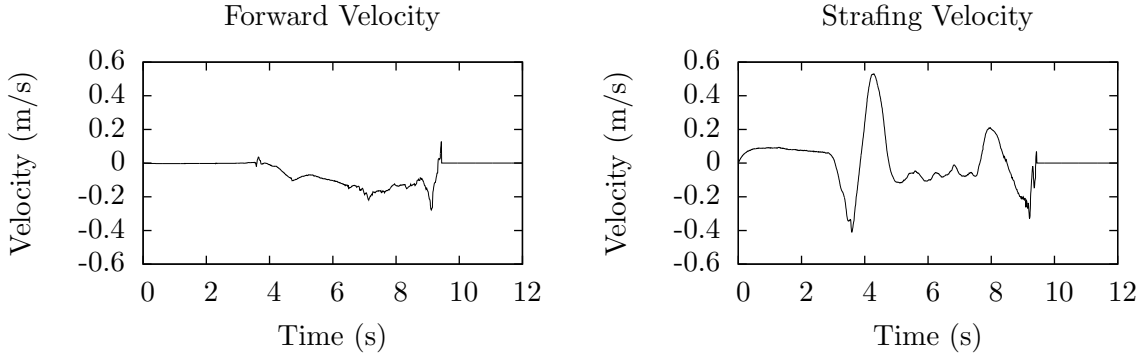


Figure 4: Translational velocity of the AR.Drone while in a stable hover as measured by optical flow.

be used as state information for a reinforcement learning algorithm. At each instant in time, the quadrotor's action is a vector of desired linear and angular velocities which are interpreted with the AR.Drone's built-in control software. Given the appropriate reward function and an ample amount of data, a reinforcement learning algorithm designed for continuous state-action spaces should be capable of relatively quickly learning high-level actions similar to those achieved by pure control theory.

One such application of this learning would be to train a quadrotor to perform adaptive high-performance maneuvers similar to those done by control theorists at the Swiss Federal Institute of Technology Zurich. Examples of these "tricks" include performing flips, catching a ball in a cup, or jugging a light ball using a modified chassis [3]. With the appropriate equipment and research investment, reinforcement learning could replace or reduce the amount of extensive manual tuning required to learn new maneuvers.

References

[1] G.M. Hoffmann, H. Huang, S.L. Waslander, and C.J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pages 2007–6461. Citeseer, 2007.

[2] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

[3] S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648. IEEE, 2010.

[4] Stephane Piskorski. *AR.Drone Developers Guide SDK 1.5*. Parrot, 2010.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *International Conference on Robotics and Automation*, 2009.

[6] J.C. Santamaría, R.S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163, 1997.

[7] W.D. Smart and L.P. Kaelbling. Reinforcement learning for robot control. *Mobile Robots XVI (Proc. SPIE 4573)*, 2001.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[9] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics: Intelligent Robotics and Autonomous Agents*. The MIT Press, 2001.