**Quadrotor Physics 101**

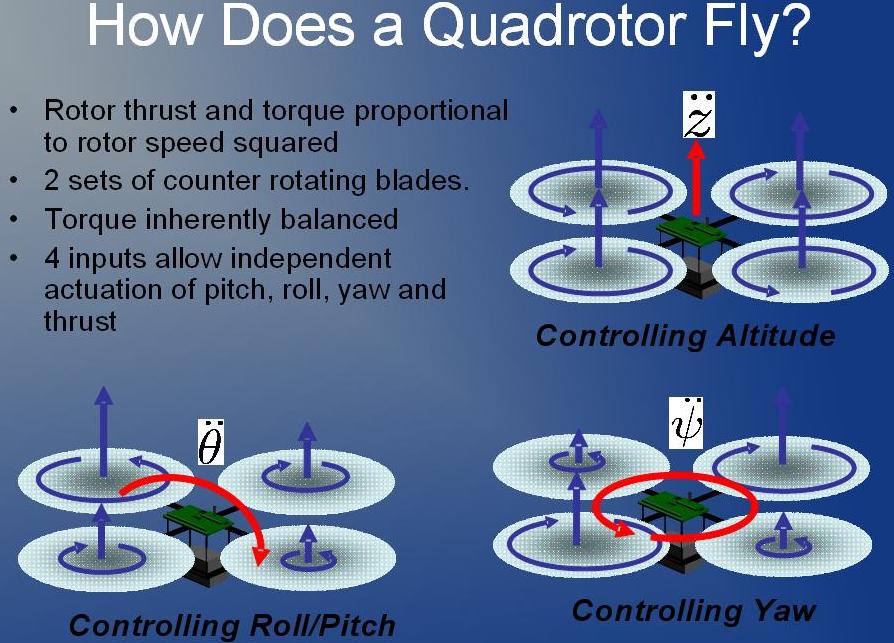
OK, it's finally time to make good on the promises I made in the first post and describe quadrotor physics. I will start off pretty slow, just so we're all on the same page.   
  
Quadrotors (and many other objects of engineering interest, flying or otherwise) can be modeled via Newton's Laws (unless your quad is sub-atomic or can fly close to the speed of light). The overall governing equation is F = m a (hopefully this sounds familiar to you), where:  
F is the net force in pounds, ounces, or Newtons  
m is the mass in slugs (yes, slugs!), grams or kilograms  
a is the acceleration in feet/second/second, or meters/s/s or centimeters/s^2 (all of these units are just common examples; there are others)  
  
In words, net forces cause masses to accelerate. This is why larger vehicles tend to be more stable - it takes larger forces to move them. Similarly, smaller craft tend to be more agile. Stability vs agility is a frequent design engineering trade off - at least before feedback augmentation was available.  
  
A certain amount of force has to be generated just to maintain a steady condition. For example, for the quad to hover it must provide sufficient force to overcome gravity. For it to maintain forward flight, the quad must overcome both gravity and air resistance (also known as aerodynamic drag). When aircraft are in this steady condition, they are said to be "trimmed" or "in trim". Only forces that are in excess of the forces required to trim the quad will cause it to accelerate.  
  
Acceleration is, by definition, the rate of change of velocity.Velocity has typical units of feet/sec, meters/sec, miles per hour or kilometers/hour. If you're familiar with calculus, acceleration is the derivative of velocity: a = dv / dt. So, for the quad to change its velocity, it must:  
1) apply forces over and above the trim forces to accelerate  
2) wait until the desired velocity is attained, then remove the additional forces,   
3) then finally supply the trim forces to maintain steady flight at the new condition.   
  
Velocity is, by definition, the rate of change of position. Or, v = dx/dt. Similarly, acceleration is the second derivative of position, a = d^2x/dt^2. Changing the position of the quad is therefore more complicated. One way it can be done is as follows:  
  
1) - 3) are the same steps as above  
4) coast at this velocity for a certain period of time.  
5) apply a force in the opposite direction to decelerate the quad (or take advantage of existing forces, such as gravity or drag)  
6) time the deceleration in step 5) such that zero velocity is reached precisely at the new desired position  
7) adjust forces to maintain the new trim condition.  
  
I have enclosed a cartoon showing the relationship between acceleration, velocity, and position for the above maneuver. I have simplified it by leaving out any considerations of trim forces. Also, in the real world, it would not be possible to change the acceleration as abruptly as shown in the figure.  
  
Its important to recognize that, since we live in a three dimensional world (or at least a world with 3 spatial dimensions, no matter what the string theorists would have you believe) acceleration, velocity, and position are all vectors in 3 dimensions. Each one has three components, which are usually referred to as X, Y, and Z; or lateral, longitudinal, and vertical; or left/right, back/forth, and up/down.  
  
Bottom line: in order for the quad to follow a desired trajectory in space and time, its control system (and/or the human flying it) must determine the correct forces to apply and when to apply them according to the above equations, which are repeated here:  
  
F = m a  
a = dv/dt = d^2x/dt^2  
v = dx/dt  
  
The last two equations are often called kinematics. They describe motion without explaining how it happens. The first is called the dynamic equation, since forces are what cause motion to take place. Together, they are known as mechanics, or "classical" mechanics, to distinguish them from quantum or relativistic mechanics.  
  
Are you with me so far? Did I lose anyone, or did I just bore you to death?  
  
- Roy

**Quadrotor Dynamics 102: Simple Rotations**

The kinematics and dynamics of rotational motion are much more complicated than that of translational motion, which I discussed last time. For now, I'll just confine myself to simple rotation, where the axis of rotation always points in the same direction for the duration of the rotation. We'll get to see at least some more of the complexity when I explain the sensing of orientation in a future post.  
  
The governing equation for rotational dynamics is similar to the translational equation we encountered last time (F = m a). It is T = I alpha, where:  
  
T = torque, which is twisting force, or force times a moment arm. Typical units include foot-pounds, ounce-inches, Newton-meters, etc.  
I = moment of inertia, which depends on not just the amount of mass, but the distribution of mass in the body. Units are kilogram-meters^2, or slug-feet^2, and so on  
alpha (this is supposed to be a Greek letter) = angular acceleration, in degrees per second per second, or radians/s/s (read on to find out what a radian is)  
  
You may be familiar with torque since many motors (electric and otherwise) are often rated by the maximum torque they can output. Our quadrotor motors and props also put out a varying torque due to the electrical power we supply, the aerodynamic forces on the propellers, friction in the motor, and other factors. The torque output depends in general on the speed of the motors. The steady torque that results from the props spinning at a constant RPM is not the torque of T = I alpha, since there is no angular acceleration in a steady rotation. The steady torque is a "trim" torque. But, as the props change RPM, then they must accelerate or decelerate and the T = I alpha formula tells us how much additional torque is required. Another familiar use of torque is the wrench we might find in our toolbox. If we can't unscrew a stubborn bolt with a particular wrench, we can use one with a longer handle that allows us to apply more torque to the bolt. Hence, torque is proportional to the applied force times the moment arm.  
  
I hope you'll recognize the formula for the circumference of a circle based on its radius r. It is, of course 2 pi r. This suggests a more natural unit to express the angle subtended by a circular arc, which is called the radian. There are 2 pi radians in a complete circle (we only use 360 degrees to describe the complete circle because the ancients once thought there were only 360 days in a year. Also, it a convenient since 360 can be divided evenly by many numbers). So, if we rotate our quad by a particular angle theta (another Greek letter) measured in radians, then any point along the quad will travel a distance of r theta, where r is the distance from the center of rotation to the point in question. Similarly, if the quad is rotating with an angular velocity of omega (yet another Greek letter) radians/sec, then that point would be traveling with a velocity of r omega. If r is in meters, then the angular velocity would be in meters/s (since a radian is a ratio of the arc length and the radius, it is unitless). Finally, if the quad were accelerating angularly at alpha rad/s/s, then that same point would be accelerating along its circular arc at r alpha.  
  
Let's put it all together. The torque is the force F times the moment arm r, T = F r. But since F = m a, T = m a r. And, since a = r alpha, we have T = m r^2 alpha. If we set I = m r ^2, we have the equation we started with: T = I alpha. From I = m r^2, we see that the further the mass is away from the center of rotation, the more torque will be required to change the angular velocity. And the closer the mass is to the center of rotation, the easier it is to change the speed of rotation.  
  
So, if we want to change the rotation rate or angular position of a quad, we need to follow the same steps as in the previous post, just substituting "torque" for "force" and putting "angular" in front of "acceleration", "velocity", and "position". For completeness, here are the steps written out:  
  
1) apply torques over and above the trim torque to accelerate angularly  
2) wait until the desired angular velocity is attained, then remove the additional torque  
3) then finally supply the trim torque to maintain steady rotation at the new condition.  
  
If we just want to change the angular velocity, we'd stop there. To change the angular position, follow the rest of these steps:  
  
4) coast at this angular velocity for a certain period of time.  
5) apply a torque in the opposite direction to decelerate the quad (or take advantage of existing torques, such as drag)  
6) time the deceleration in step 5) such that zero angular velocity is reached precisely at the new desired angular position  
7) adjust torque to maintain the new trim orientation.  
  
And here are the formulas:  
  
T = I alpha  
I = m r^2  
alpha = d omega / dt  
omega = d theta / dt  
a = r alpha  
v = r omega  
x = r theta  
  
- Roy

**Quadrotor Dynamics 103: Applying the Principles to the Quadrotor**

Most of what we've said thus far could apply to just about any physical object. Let's focus in on the specifics of the quadrotor. As wikipedia will tell you, a quadrotor is just a cross with a propeller (and probably a motor) at end of each of the four arms. Each propeller produces a force, which we call thrust, and a torque, both of which increase as the speed of rotation increases (we'll get into more details about thrust and torque generation in future posts).   
  
There's another Law of Newton that we haven't covered yet, which is "If a force (or torque) acts upon a body, then an equal and opposite force (or torque) must act upon another body". This is how the props work: they apply a force to the air to accelerate it downward, and the air applies an equal and opposite force on the prop to accelerate it upward. Similarly, the motors provide a torque to the propellers and the propellers supply an equal an opposite torque to the motor - and anything the motor is connected to, like the frame of the quadrotor. So, the frame will tend to spin in the opposite direction as the props. The inertia of the props is smaller than that of the frame, so the props will accelerate more than the frame (recall the equation T = I alpha? The torques (T) are the same, but since the moments of inertia (I) are different, so are the accelerations (alpha)).   
  
In a typical quad, adjacent props will spin in opposite directions, so that the prop on the other end of the axis will spin in the same direction (see picture). In other words, two of the props will spin in a clockwise direction and the other two will spin counter-clockwise. In a stationary trimmed hover condition, the torques from the clockwise spinning motors balance out the torques from the counter-clockwise ones and the net yawing moment is zero ("moment" is another way of saying "torque"). In other words, there is no tendency for the quad to spin about its vertical axis when it is trimmed. All the thrust forces are balanced as well, so there are no pitching or rolling moments. That is, the quad does not rotate about either of the two axes that are represented by the arms that connect two motors. It is this symmetry and balance of the forces and torques that allow the quadrotor to work.  
  
But what if we *want*the quad to rotate? Referring to the lower left corner of the picture, we can increase the thrust on one prop - lets call it the left one - and decrease the thrust on the opposite (right) prop by the same amount. The yaw torques are still balanced: the sum of the axial torque of the left and right propellers is exactly the same as it was during the trimmed hover. Hence, it is still balanced by the torque of the other propellers (the front and rear ones). Similarly, the net thrust is exactly the same as it was during the hover. The only difference, relative to trim, is that the props have applied a rolling moment to the airframe, which will cause it to accelerate angularly about the roll axis. Due to the symmetry of the quad, the "pitch" axis behaves exactly the same as the roll axis.   
  
How about if we want to yaw the quad? As you can see in the lower right figure in the picture, we can, for example, increase the RPM of the two clockwise rotating motors and decrease the speed of the two counter-clockwise propellers. Once again, the net thrust is the same as for the hover case, and the net roll and pitch moments are also balanced. Since the yaw moment is unbalanced, the quad will yaw counter-clockwise (opposite to the direction of the net propeller yawing moment). To yaw the quad clockwise, we just reverse the situation.  
  
To increase (or decrease) the altitude of the quadrotor, all we have to do is increase (or decrease) the speed of all 4 props equally, as shown in the upper right corner of the picture. As before, pitch, roll, and yaw torques are all balanced, so the quad will climb (or descend) but remain at a level, flat attitude without yawing.  
  
The longer the arms of the quadrotor are and the heavier the motors are, the larger the moments of inertia will be and the less maneuverable the quad will be. This is especially true for the yaw axis. The propellers we typically use are designed to produce lots of thrust but not very much torque. After all, the more torque the props require, the more torque the motors must supply and the shorter the flight times will be.  
  
We now know how to rotate the quad about all three axes (pitch, roll, and, yaw) and how to climb or descend, that is how to move or translate in the z axis. How do we get the quad to move in the other two axes (x and y)? We'll find out next time.  
  
- Roy



**Quadrotor Dynamics 104 - Moving right along.**

I feel confident that someone reading this thread has thought, "Why can't I just connect the receiver directly to the motors? Why do I need sensors and a microprocessor and PID and all that stuff?" I won't come out and say that this is impossible to fly - I work with test pilots, so I know just how adaptable human beings can be. However, most people who tried this for themselves probably found out just how difficult it is to control an unaugmented quadrotor. Let's see why this is so.  
  
Imagine we have our quad perfectly trimmed in hover over a particular spot of ground, and we want it to hover at the same altitude, but several yards from where it is now. What do the propellers have to do to achieve this objective? Recall in what follows, that it does not matter how the rotors were commanded - either from a sophisticated feedback control algorithm, or via direct commands from the r/c receiver. The rotors have to do something similar to what I'm describing.   
  
We already know what has to happen: we need to accelerate the quad in the direction we want to go, coast at a particular velocity, then decelerate so the quad comes to a stop exactly at the desired final hover spot. In order to accelerate, we need to tilt (pitch or roll) the quad so some of the thrust of the rotors is pointing in the direction we want it to go. As it rolls (or pitches), we have to increase the overall thrust to balance gravity, otherwise it will accelerate into the ground. Of course, every time we adjust the attitude, we'll also have to adjust the overall thrust to maintain altitude.  
  
Next, we'll have to roll the quad back to level to stop its acceleration over the ground and have it coast at a constant velocity. But, as its coasting along, there will be air resistance (drag) slowing it down. Depending on how fast and how far we want the quad to travel, we may have to tilt the quad slightly into the direction of motion to provide sufficient force to overcome the drag. As it nears the destination, we have to roll the quad back toward the direction it came to decelerate it. Once the the quad comes to a halt we have to roll it back to level to keep it there.  
  
And how to we get it to roll to a particular angle? Again, we already know. We have to apply a rolling moment (spin up the rotors on one side and spin them down on the opposite side), remove the moment (let it coast), then apply a rolling moment in the opposite direction to halt the angular motion when it arrives at the desired angular position. This three step process has to happen every time we want to alter the angular position. Does this sound like a lot of work for a simple re-positioning maneuver? There's more.  
  
If the quad is flying outside it (or you, the "pilot") will have to cope with wind gusts. Even indoors, the propeller wakes will interact with the arms and other structures (and the floor and walls if you get too close). The quad components and/or each prop/motor are probably not perfectly balanced either, which may result in a requirement for offset trim. Finally, even if you have all these factors completely solved, the quadrotor is dynamically unstable. The situation is analogous to balancing a broomstick on the palm of your hand. This means that if you (or the electronics) aren't actively controlling it, the quad will tend to deviate from its hovering position and eventually spiral out of control and crash.  
  
To summarize:  
- Since the quad only has four motors, it can only directly control 4 degrees of freedom: pitch, roll, yaw, and thrust. That is, it is *underactuated*.  
- To control the other two degrees (longitudinal and lateral translation), we must tilt the quadrotor. In other words, the translational and rotational dynamics are *coupled*. To control the position of the quad over the ground, we have to control its orientation or attitude.  
There's not too much we can do about these (unless we want to add additional propellers or control surfaces).   
  
However, there are some undesirable aspects of the dynamics that we can correct:  
- Mother Nature (or Newton, if you prefer), only provides us with forces and torques to control the quad - meaning we can directly control accelerations. But, we'd rather have control over velocities and positions.  
- The quad is susceptible to unwanted disturbances, both external (wind) and internal (mass balance)  
- The quad is unstable and requires constant attention to hover.  
  
These are the very items that control theory can address, as we'll discuss next time.  
  
- Roy  
  
p.s. I had originally planned to illustrate the translation of the quad from one hovering position to another in a series of figures, clearly depicting the forces involved. But, I suspect that this might be too "trivial". Let me know if there's interest in this and I'll try to create it in the future.

**Control Theory 100 – Some Introductory Remarks**

Finally - the topic you've all been waiting for - Control Theory! An auspicious topic for the 100th post of this thread! (it sure took me long enough to get here)  
  
Before we begin, I hope that no one reading this thinks they can become an expert controls engineer by reading a few blog posts. After all, it takes a one-semester sophomore level course in a typical engineering undergraduate program just to understand the *language*of control systems.   
  
Why is it so difficult to study Control Systems?  
  
- You need to have a solid understanding of the physics of the systems you are trying to control.  
  
- The theory is based on a lot of mathematics: differential equations, difference equations, complex analysis (in the sense of imaginary numbers), linear algebra, probability, information theory, optimization, transforms, and more. It is not necessary to master all these subjects, but a solid knowledge of the requisite components of these topics is required.  
  
- There is a considerable amount of “art” involved. Despite the overwhelming amount of mathematics and the existence of sophisticated computer aided engineering packages, it is still not really possible to press a button and have the computer design a practical control system. There are so many tools in the controls toolbox that it requires some experience and knowledge to know exactly which one is the best to use for the problem at hand. And there is almost always some tweaking that must be done to ensure that every condition is covered. As many of you have found out, it is entirely possible to ignore all the math and just wrap a PID loop around the problem and continually tweak it until it basically works – which is an art unto itself.  
  
Why is it so difficult to teach Control Theory? (in case you care)  
  
- There is no obvious hierarchy of the mathematical concepts; they’re all just needed at once. This makes it difficult to determine an order or sequence to properly teach the subject, and I think many professors and textbook authors choose the wrong path. My intended audience is, of course, not control engineers. But if you are an undergraduate suffering through a controls program, let me know. I might be convinced to write some supplemental posts.  
  
Why is control theory so useful?  
  
- The theory is pretty much the same no matter what the system is that you are trying to control. Essentially the same theory that works for flying quadrotors also applies to: robots, the cruise control of your car (and the engine controls, too), temperature control of your living room (or of a bio-reactor), or even the economy.   
- Much of the theory overlaps with other subjects, such as Digital Signal Processing. In fact, a control algorithm running on a microcontroller can be viewed as a specialized digital signal processor.  
- I would claim that it is difficult to get very far in the study of other sciences, such as biology, without a firm grasp of the concept of feedback, which plays a central role in control theory.   
  
So, presuming I haven’t scared you off, let’s get started!  
  
- Roy

**Control Theory 101 – The Joy of Feedback**

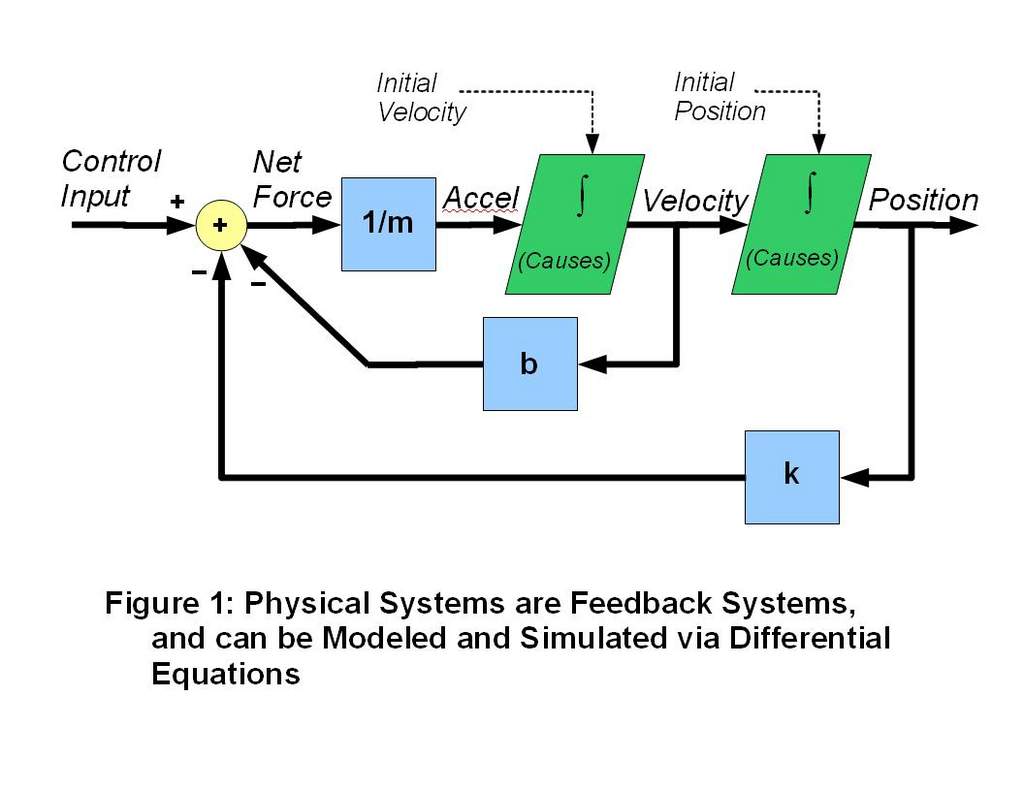
As we indicated in post 99, the objectives of control systems include the following:  
  
1) Make the system (a quadrotor in our case) follow a desired trajectory – that is, an evolving sequence of velocities and/or positions over time. It does not matter if the trajectory is predetermined (like a sequence of waypoints) or generated in real-time (from r/c joystick commands, for example)  
2) Reject any disturbances, such as might be caused by wind gusts or an imbalanced configuration  
3) Stabilize the system – that is, alter the tendency for the system to diverge from trim  
  
Practically, we have to figure out how to adjust the 4 propeller thrusts (which are the only means we have to influence the quad’s behavior) to achieve these objectives. How can we accomplish this?  
  
To find the answer, we have to go back to the physics fundamentals that we discussed earlier. Forces cause masses to accelerate (and the angular equivalent: torques cause inertias to accelerate angularly). Acceleration, by definition, is the time rate of change of velocity: a = dv/dt for those of you who have had some calculus. Velocity is, in turn, the time rate of change of position: v= dx/dt (remember all that?). The last part of the puzzle is that the inherent forces (that is, the forces other than the ones coming from the control system itself) are themselves functions of velocity and position. For example, we said earlier that the drag force on the quad is a function of its velocity through the air. Similarly, the direction of the propeller thrust vector depends on the angular position of the quad. So, to sum up, forces cause acceleration, which cause velocity to change, which in turn cause position to change. These new velocities and positions result in new forces being generated, and the loop repeats itself endlessly. Computer programmers out there may recognize this concept as “recursion”. Control engineers call it “feedback”, and it is the key to control theory.  
  
Besides “recursion” and “feedback”, this structure is also known as a differential equation, because it defines relationships between variables and their rates of change. Recall  
  
F = m a  
  
The total force F comes from the control, which we’ll call “u” (the prop-motor thrusts in our case), plus a function of velocity plus a function of position.   
  
F\_total = u + F\_velocity + F\_position  
  
For simplicity, we’ll represent the force due to velocity as just a constant (b) times the velocity v.   
  
F\_velocity = -b v  
  
In general, this function may be more complicated than this. In fact, the actual aerodynamic drag force is proportional to velocity squared. We will discuss later when this type of simplification can be justified. The minus sign indicates that the force opposes the direction of motion, similar to the way the drag force opposes the velocity (assuming b is positive).  
  
We’ll also write the position-based force equation as just a constant (k) times the position  
  
F\_position = - k x  
  
As with the velocity case, reality may be more complicated. The negative sign indicates the force is acting to bring the mass back to zero displacement, like a spring, if k is positive. However, if k is negative, the force will tend to pull the mass away from zero displacement, like an inverted pendulum (which is a fancy way of describing balancing a broomstick on your hand).  
  
For the calculus savvy, we’ve said earlier that acceleration is the derivative of velocity. This means that velocity is the integral of acceleration. Similarly, since velocity itself is the derivative of position, position is therefore the integral of velocity. If these terms don’t mean anything to you, don’t worry. We will be getting to a place soon where we can simplify some of the details of calculus. For now, you can just think of it this way: forces cause acceleration, which causes velocity to change, which causes position to change.  
  
Putting it all together:  
  
(-k x - b v + u) /m = a  
v = (integral of) a  
x = (integral of) v  
  
If we put this into a diagram form, as in Figure 1, the feedback loops become apparent. I’ve included the integral signs, for those of you who understand such things. If you think of this as a computer simulation program, all you have to do is give it an initial velocity and position and supply it with the control input at each instant in time and let the computer crank through the program to determine the system behavior for all future time. This is how flight simulators and other physics based sims work. Its recursive because the current position and velocity depends on the earlier position and velocity, which depends in turn on the v and x before that, and so on - all using the same function at each time step (but with potentially different inputs each time).  
  
The behavior of the system is characterized by the parameters k, b, and m. If our quad is very streamlined, for example, then the b term will be smaller, since aerodynamic drag won’t affect us as much as it would for a less streamlined configuration. If we don’t like how the system behaves, we have to change these parameters somehow. For example, as we discussed earlier, if the sign of k is negative, the system will diverge, like the broomstick balanced in our hand. In the past, such changes had to be accomplished entirely in the physical domain. For example, this is why fixed wing airplanes, like passenger jet liners, have tails (vertical and horizontal stabilizers). Without them, wind gusts would blow them completely off course. The tail provides a restoring force to keep the plane pointing in the right direction, like a weathervane. In other words, tails make the negative k of the fuselage and wings into an overall positive k of the airplane as a whole.  
  
Today, we can alter the system parameters using computers. Figure 2 shows us the basic premise. If we can sense the position and velocity, the control system (which is essentially just a computer) can simply multiply them by whatever numbers we want, and use these values to provide additional forces via the control inputs (up to the limits of our control actuators, of course). If the k of the physical system is negative, we can use this electronic or digital feedback to make the overall k value positive. This is how the F117A Stealth Fighter can get away with such a relatively small tail: it employs a sophisticated digital “fly by wire” feedback control system. We refer to the combination of the system itself plus the control system as a “closed loop system”, for obvious reasons.  
  
We have discussed how feedback control can stabilize an unstable system (objective # 3 above). What about the first two objectives, trajectory tracking and disturbance rejection? Once again, feedback comes to the rescue. If we alter our feedback loops from Figure 2 to compare the sensed position and velocity with desired position and velocity, we can drive the control system with the errors. The control system (if properly designed) can cause the errors to converge to zero, or at least be minimized. In this manner, the quadrotor will follow the desired trajectory (to a greater or lesser extent) See Figure 3. Note that if the desired trajectory is zero, the system is identical to the one in figure 2. As in the earlier figures, the blocks labeled Kv and Kp may be more complicated in practice than just constant values. This diagram is meant as a notional illustration of the feedback concept, and shouldn’t necessarily be taken as a practical design solution.  
  
Summary:  
  
- Physical systems are feedback systems, and can be modeled via differential equations.   
- The behavior of the system depends on its parameters (the coefficients of the differential equations)  
- Feedback control systems can be used to manipulate the inherent parameters, thus altering the behavior of the physical system.  
- Feedback control systems can also be used to compel the system to track a desired trajectory and reject unwanted disturbances.  
  
Next time, we will go into more detail on the physics and mathematics of the all-important feedback loop.  
  
-Roy

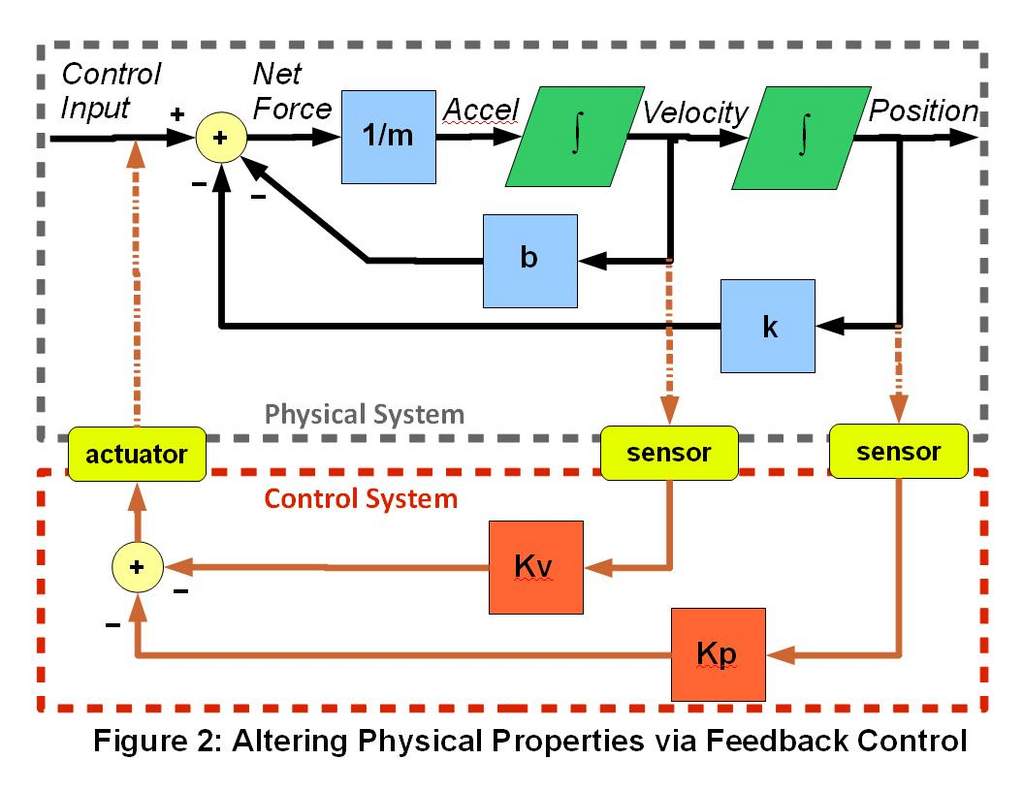
Attached Thumbnails

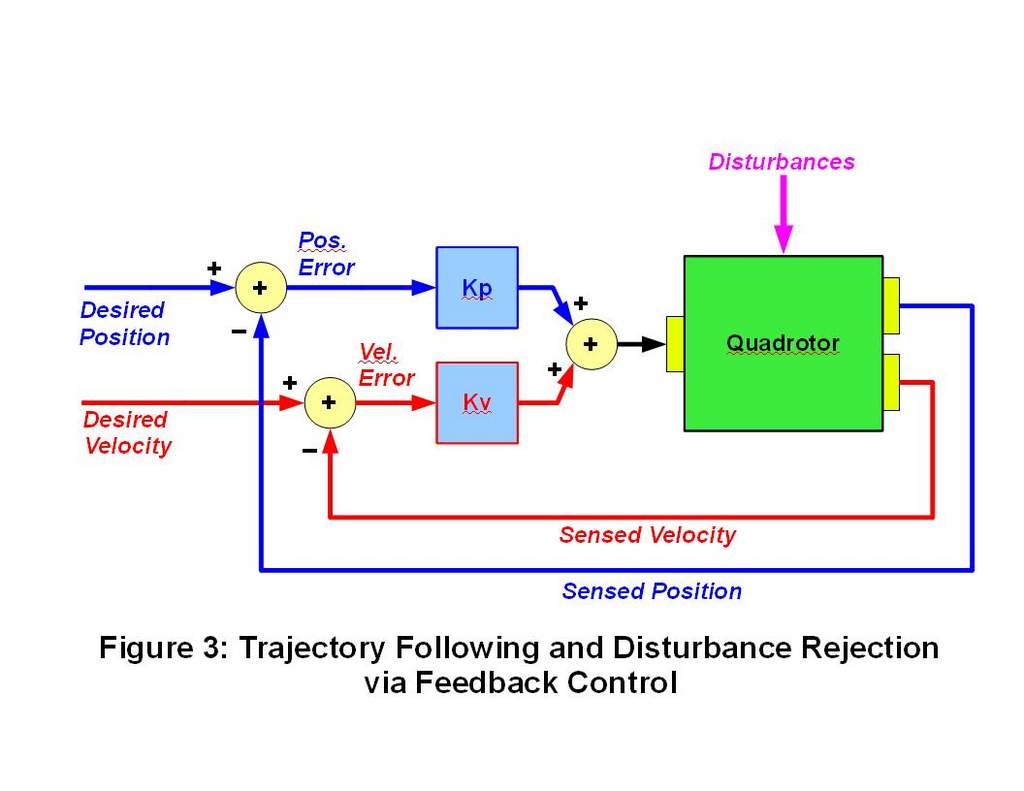
* [Name: Figure 1 Physical systems are Feedback systems.jpg
  Views: 836
  Size: 48.8 KB
  Description: ](http://www.rcgroups.com/forums/attachmentNew.php?attachmentid=3596859)48.8 KB · Views: 836

* [Name: Figure 2 Altering Physical Properties via Feedback control systems.jpg
  Views: 758
  Size: 68.9 KB
  Description: ](http://www.rcgroups.com/forums/attachmentNew.php?attachmentid=3596861)68.9 KB · Views: 758

* [Name: Figure 3 Using Feedback to Follow a Trajectory and Reject Disturbances.jpg
  Views: 677
  Size: 46.4 KB
  Description: ](http://www.rcgroups.com/forums/attachmentNew.php?attachmentid=3596862)46.4 KB · Views: 677







**Right Said Fred - Modeling 101**

Yes, I’m back – has it really been that long? Hopefully some of you are still with me. As a reminder of where we left off, I’m responding to gke’s request to create a model of the angular response of the quad from the data provided (from post 169 [http://www.rcgroups.com/forums/showp...&postcount=169](http://www.rcgroups.com/forums/showpost.php?p=17769860&postcount=169)).  
  
As I mentioned earlier, this is getting ahead of my initial "lesson plan". So, if you're confused, don't worry - I will backtrack and fill in the missing details later.  
  
Every control design has 4 steps 1) understanding the dynamics of the object you are trying to control; 2) understanding the desired behavior 3) choosing a controller architecture (based on the first two steps); and 4) tuning the gains. You may find that you just can’t obtain the desired performance in step 4, so you have to choose another controller. Or that you have to go back and revise your ideas about the desired behavior, if you just can’t get there. You may even have to go back and learn more about the dynamics. Very little is completely easy or straightforward when it comes to control design.   
  
I suspect that, for most people reading this, the architecture used is PID because that is the only option they know. The other three steps are conducted simultaneously under the guise of tuning the gains. Clearly this is not the optimal approach. A more analytical approach is to first derive a mathematical computer model of the quadrotor based on physics and experiments (that’s step 1 above). Having a model won’t preclude iteration and experimentation, but it is a lot safer and cheaper to experiment on a model than on a real quad. The model will also help guide the real-world experiments. Finally, there are many advanced control methods that require a model.  
  
Let’s take a look at the data that gke provided. First, a quick sanity check – does it make sense? Is the rate the derivative of the angle? Yes, it checks out. In half a second, the quad rolled about 0.6 radians = 34 deg, which is feasible. At that time it’s rolling at 2.5 radians/sec = 143 degrees/second, which is certainly possible. The rate trace exhibits what I’d call “higher order dynamics” (all that wiggling), but let’s ignore that at least for now. We should strive for the simplest model we can that adequately captures the important characteristics.   
  
Note that the average angular rate is a fairly straight line, as gke's excel plot shows (ignoring the wiggles). There’s no evidence of rate damping, at least as of the 0.5 second mark . Damping is a negative function of velocity. The faster the rotation, the more damping would resist the motion and the curve would start to flatten out. That does not happen in the 0.5 seconds worth of data we have.  
  
Recall from post 75 ([http://www.rcgroups.com/forums/showp...9&postcount=75](http://www.rcgroups.com/forums/showpost.php?p=16015629&postcount=75)) that the physics for this case is simply torque = inertia times angular acceleration. For convenience, let’s say the input is 5 PWM counts (gke said it was +5 to one motor and -5 to the opposite motor) This results in the ESCs changing the speed of the two opposed motors, which results in a changed thrust force in the two propellers. These (roughly) equal and opposite thrust forces (known as a “couple”) create a net torque when multiplied by the (unknown) motor-to-motor distance. This is then equal to angular acceleration multiplied by the (unknown) moment of inertia, per the physics. I’m leaving out a lot of non-linear dynamics and the algorithms inside the ESC. For reasons that I’ll explain later, what we often desire in the control world is a linear model.   
  
Sounds like too many unknowns, but it isn’t. What we are primarily concerned about is how the motor commands affect the rotation of the quad, and we have that. The angular acceleration (alpha) is just the slope of the angular velocity curve – in this case 5.2856 radians/second/second. And the quad achieved that alpha with a motor command of 5 PWM counts. All of those unknowns collapse into the 5.2856 number.   
  
If the dynamics were truly linear, we would expect that doubling the motor command to 10 PWM counts would produce double the acceleration, or 10.6 rad/s/s. But, since the response isn’t necessarily linear, this might not be the case. We’d have to get gke to perform more experiments to be sure. For now, we’ll ignore this inconvenient situation.  
  
So, the relevant equation is 5 PWM counts = 5.2856 rad/s/s. Since we want to get this in the form of figure 1 of post 129 ([http://www.rcgroups.com/forums/showp...&postcount=129](http://www.rcgroups.com/forums/showpost.php?p=16548841&postcount=129)), we have to solve for angular acceleration. Hence:  
  
alpha = 1.05712 U  
  
where alpha is the angular acceleration in rad/s/s and U is the motor command input in PWM counts.  
  
All we have to do next is integrate alpha to determine the angular velocity (omega) and integrate again to find the angular position (theta). As we’ve already mentioned, there’s no apparent damping, so the “b” term from figure 1 of post 129 is zero. There’s obviously no spring trying to pull theta to zero, so the “k” term is also zero. Our model is almost complete. There are just two more items I’d like to include, at least for now: the delay in the response and disturbance modeling.  
  
Although it isn’t shown on the graph, the input steps at time 0 (the input trace is in the excel file), but the quad doesn’t really start to respond until about 0.03 seconds later. This could be due to a pure delay. If the internal ESC control loop is only running at 50 Hz, it would take 0.02 sec for it to respond to a command. Or, the apparent lag in the response could be due to the fact that it takes time for the propeller to spool up to speed. Just like the quad itself, the motor supplies a torque, which results in angular acceleration of the prop, which means it takes time to change the prop speed. Which of these is the culprit behind the response delay? It is probably due to a combination of both effects. We may need more experimentation or details of the ESC to be certain. Thirty milliseconds may not seem like a lot of time, but either of these phenomena will have consequences for our controller design (as we shall see). So, they should be included in our model.  
  
Our quadrotor has to contend with more than just the forces and moments due to the motors that our control system is deliberately commanding. For example, there are also wind gusts and other undesirable aerodynamic phenomena (like blade flapping). Or, there may be a center of mass offset due to the configuration. All of these can result in uncommanded torques. Our model should include these effects, so we can test how well our control algorithms can reject these disturbances.  
  
Now we have enough information to create a model. As I mentioned earlier, I will implement it using the free ScicosLab tool in the next post. But, you can do it in MATLAB or other similar tool.  
  
- Roy

**Rate Feedback - Modeling 102**

The Model  
  
Yes, I’m still here – is anyone else listening? We were in the process of building a model based on gke’s data set. As I said last time (way back in post #177[http://www.rcgroups.com/forums/showp...&postcount=177](http://www.rcgroups.com/forums/showpost.php?p=18245672&postcount=177)), I prefer using the free ScicosLab package to the expensive MATLAB one. The programs are similar, but have significant differences. I will try to keep the discussion generic enough for any design package. One additional note, I use ScicosLab version 4.4.1 on Windows XP (<http://www.scicos.org/>) not Scilab (<http://www.scilab.org/>),. I can point you to some tutorials if you need them to get started  
  
The first figure shows the model that captures gke’s data (note all ScicosLab files are included in the enclosed zip file. These files were generated on a Windows XP platform).   
  
To build this from scratch, you’d first start with the chain of integrators that represent the rate and attitude of our linear, one-dimensional (single axis) quadrotor model and connect them. The input to the first integrator is the angular acceleration, which we found the formula for last time:  
  
alpha = 1.05712 x U  
  
where alpha is the angular acceleration in rad/s/s and U is the motor command input in PWM counts. We also implemented a 0.03 sec delay on the input as we discussed last time. There is no restoring spring force, and we didn’t find evidence of damping, so there are no “feedback” terms in this model (compare with figure 1 of post 129[http://www.rcgroups.com/forums/showp...&postcount=129](http://www.rcgroups.com/forums/showpost.php?p=16548841&postcount=129)). We’re also including a disturbance torque, which could represent a wind gust or an offset c.g, etc. The disturbance input will be set to zero initially. The model includes a delay based on the delay in the response (which in the real quad is a combination of ESC processing delay and propeller spool up lags). The Scicos model is in the file “Simple Quad Angle Model.cos”  
  
Plot 1 shows the response of this model to a 5 PWM count step, overplotted with gke’s original data (examining the Excel file that gke included in his post, he intended the input to be 10 PWM counts (+5 on one side and -5 on the other). His model would halve the 1.05712 gain). As you can see, it matches pretty well (as we said earlier, we weren’t trying to match the higher frequency dynamics of the rate trace).   
  
  
Feedback  
  
OK, let’s wrap some feedback around this model. (Note that this is not necessarily the best way to proceed with control system design, but we don’t have enough theory behind us to see why that’s the case. So, let’s just jump right in). We’ll start with “rate mode” - that is, the angular velocity command is proportional to the stick deflection. Since we’re (technically) only interested in controlling one of the states (one of the two integrators), we can use a single (proportional) gain. We’ll set the initial feedback gain to one PWM count per rad/s (see Scicos file Simple Quad Angle Model Rate Feedback.cos, or Figure 2). As you can see in Plot 2, the quad model does attain the commanded rate (which is one rad/s in this case), but it takes over 3 seconds to get there (technically, the output only asymptotically approaches the desired rate – it never actually reaches it. From a practical standpoint, we can use 90% or 95% of the final value to measure the “rise time”). I don’t think we’d be happy with this performance. Before we move on, I just want you to note the shape of the response. This kind of response to a step input that asymptotically (or exponentially) approaches its final value is called a “low pass filter” (because it allows lower frequencies in the signal to get through, but blocks higher ones) or a “lag” (because of the apparent delay in the reponse). If the input were changed, say to 2 rps instead of one (which you can try by double clicking on the “PWM” input block and updating the 1 to 2 – or just check Plot 2), it would still take the system over 3 seconds to reach the commanded value. The shape of the response curve is the same, except for the amplitude.  
  
Can we do better? Let’s try a gain of 10 PWM units / rps. Click on the gain block and increase the 1 that’s there to 10 (go ahead; I’ll wait). See Plot 3 for the results. That’s much better – less than a third of a second to reach final value. So, we find that increasing the proportional gain results in a faster response. Why? It’s because the control system will put out a larger PWM command to the motors for the same error, which results in more torque and thus higher angular acceleration. Plot 4 compares the PWM commands for the two gains.   
  
  
Observations  
  
This gives you a glimpse on how we could do automatic gain tuning, in principle. The system would just have to be able to measure the angular acceleration response to a steady PWM command (just like we did manually). Then, knowing the desired speed (rise time) of the closed-loop response, the system could compute the gain to achieve that response. This could be done inside the quad controller itself, in which case it would be called “adaptive control”. It could also be performed on a groundstation using post-flight telemetry (or logged) data, in which case it would be called “off line auto-tuning”.  
  
Another observation: if we could generate the PWM signals from Plot 4 in response to an input command, then we wouldn’t need feedback at all, at least in our simulator. Feedback can accomplish great things, but it can also cause harm if not used properly (as anyone who has heard the “feedback” from a microphone pushed too close to its speakers can attest). It also is not free – all those sensors and all the code to process them don’t just magically appear. Feedback should ideally be reserved for two contingencies: 1) unpredictable disturbances (like wind gusts), and 2) variation or uncertainty in the system itself.   
  
Looking at Plot 4, you see another “drawback” of linear feedback. The PWM command is proportional to the error, which starts out large initially and exponentially decreases as the quad catches up. When we increase the gain to get a faster response, most of the effect comes at the beginning of the cycle, when the PWM spikes very high Note that the slope of the error behaves the same way, so that derivative feedback would have the same general character. Integral feedback could get us more control, but once we reached the desired rate the integrator would have to “unwind”, which would only happen when the error changed sign – causing overshoots. However, the system could get a faster response if it just held a large constant value of the PWM command, until the quad came close to the desired rate. This is an example of how non-linear controls could yield performance improvements (but they are usually more complicated to design).  
  
  
Limitations  
  
Although our model does not include it, the real PWM command would be limited to some value (say 100, or 256, etc. Note that we could easily add a command limit to our model if we wanted to – Scicos has limit blocks.) . So the control system will eventually saturate so that additional error or higher gains will not result in increased speed of response. Control authority will, at some point, be a limiting factor on performance.  
  
Can we do better than our roughly 0.3 sec rise time (ignoring any limits on the command signal)? Try a gain of 40 (Plot 5), then 60 (Plot 6 – note that I’ve cut off the peaks, but the amplitude of the rate response is actually increasing without bound on every cycle.). What’s going on here? As the gain increases, the time delay becomes more significant (If you delete the delay block from the sim model, you’ll find that the oscillations will go away). Even though the system has actually reached the desired value, the system does not know it because it is responding to the state of a few moments ago, so it overshoots. If the gain is not too large, the system can eventually recover, but it may take several cycles of oscillation to do so. But, if the gain gets too large, the oscillations will continually grow until some limit is reached or something breaks. This demonstrates that it is possible for a control system to cause instability, which is why care must be taken during its design. It also shows that time delays can cause problems for feedback control systems.  
  
Next time – what about that disturbance block that we were so careful to include?  
  
  
- Roy  
  
  
p.s. The graphs shown here were produced using Scicos simulation data but were actually created by the (free student edition of) Magic Plot. <http://magicplot.com/>. Sclilab can produce plots natively, but they do not look as nice as these.

