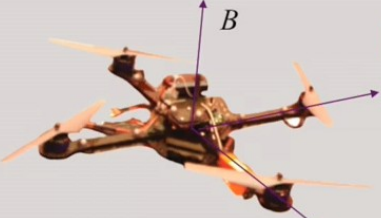


# Motion Planning for Quadrotors

We're now ready to apply what we have learned to quadrotors. Let's recall again the dynamic equations-of-motion from previous lessons:



**Newton-Euler Equations**

$${}^A\omega^B = p \mathbf{b}_1 + q \mathbf{b}_2 + r \mathbf{b}_3$$

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}$$

$u_1$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$u_2$

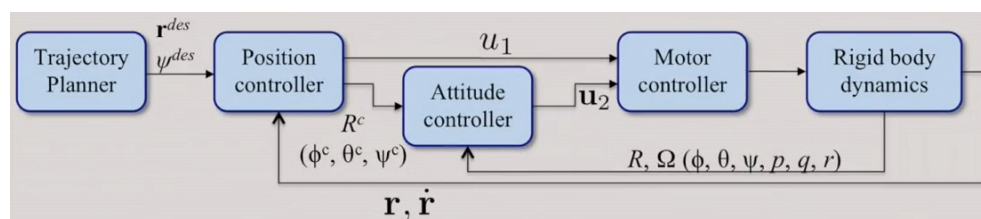
Components in the inertial frame along  $\mathbf{a}_1$ ,  $\mathbf{a}_2$ , and  $\mathbf{a}_3$

Rotation of thrust vector from  $B$  to  $A$

Components in the body frame along  $\mathbf{b}_1$ ,  $\mathbf{b}_2$ , and  $\mathbf{b}_3$ , the principal axes

$u_1$  and  $u_2$  are the two sets of inputs. We're now trying to generate trajectories in terms of the quadrotor position and trying to figure out how to control the motors to drive the vehicle to those trajectories.

The quadrotor controller consists of two loops. First, the outer position-control-loop that specifies  $u_1$  and then second, an inner attitude-control-loop that determines  $u_2$ :



In order to control the position, the inner loop, the attitude-control-loop, has to work perfectly.

Back to the equations.  $u_1$  and  $u_2$  are the two inputs. We can see from the first set of equations that the second-derivative of position depends on  $u_1$ .  $u_1$  is a scalar quantity. There's only one input to  $u_1$ , but we're trying to control the x, y and z components of acceleration.

Also, notice that the rotation matrix  $R$ , depends on the roll, pitch, and yaw angles.


The second set of equations relates the angular velocities,  $p$ ,  $q$ , and  $r$  to the derivatives of the roll, pitch, and yaw angles, and the third set of equations at the bottom relates the input  $u_2$  to the rate of change of angular velocity.

From the two sets of equations at the bottom, it is clear that the second derivative of the rotation matrix depends on  $u_2$ .

Now, if we turn our attention back to the first set of equations, we can see that the derivative of velocity, or the acceleration, depends on  $u_1$ , and that this dependence is linear. But through  $R$  it also depends on  $u_2$ . If we take into account the fact that this dependence of  $R$  on  $u_2$  is through the second-derivative, as we see on the bottom, then we can combine these facts to show that the fourth-derivative of position depends on  $u_2$ .

This might be easier to see if we consider the linearised model of the system

**Linearized Model**  $(\theta \sim 0, \phi \sim 0, \psi \sim 0)$   
 $(p \sim 0, q \sim 0, r \sim 0)$



$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}$$

$R(\theta, \phi, \psi)$   $u_1$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

$u_2$   $0$

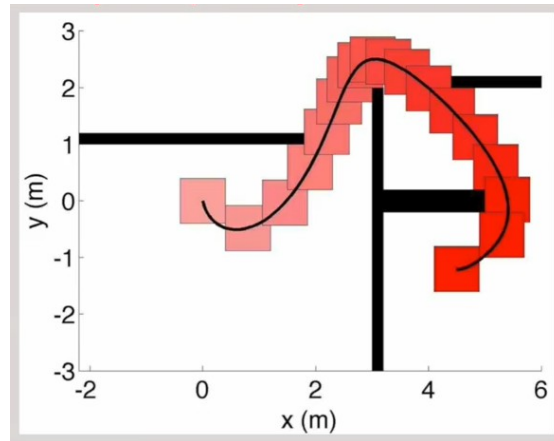
In the linearised model, the rotation matrix is the identity matrix. The angular-velocities are related to the rates of change of roll, pitch and yaw, through an identity matrix. And the non-linear inertia-dependent terms at the bottom disappear. We should go through these equations and convince ourselves that the second-derivative of position is proportional to  $u_1$  and the fourth-derivative of position is proportional to  $u_2$ .

In summary, the position control loop involves attitude control, and because of that, the position control system yields a fourth-order system. To specify trajectories, we only want to consider those trajectories that are differentiable at least four times. This motivates the use of a minimum-snap trajectory, which tries to minimize the fourth-derivative of position integrated over the time history.

$$x^*(t) = \arg \min_{x(t)} \int_0^T (x^{(iv)})^2 dt$$

We use minimum-snap trajectories for motion-planning for quadrotors. The video shows several examples of minimum-snap trajectories. Remember that this all only works if we have a robust inner attitude-control loop.

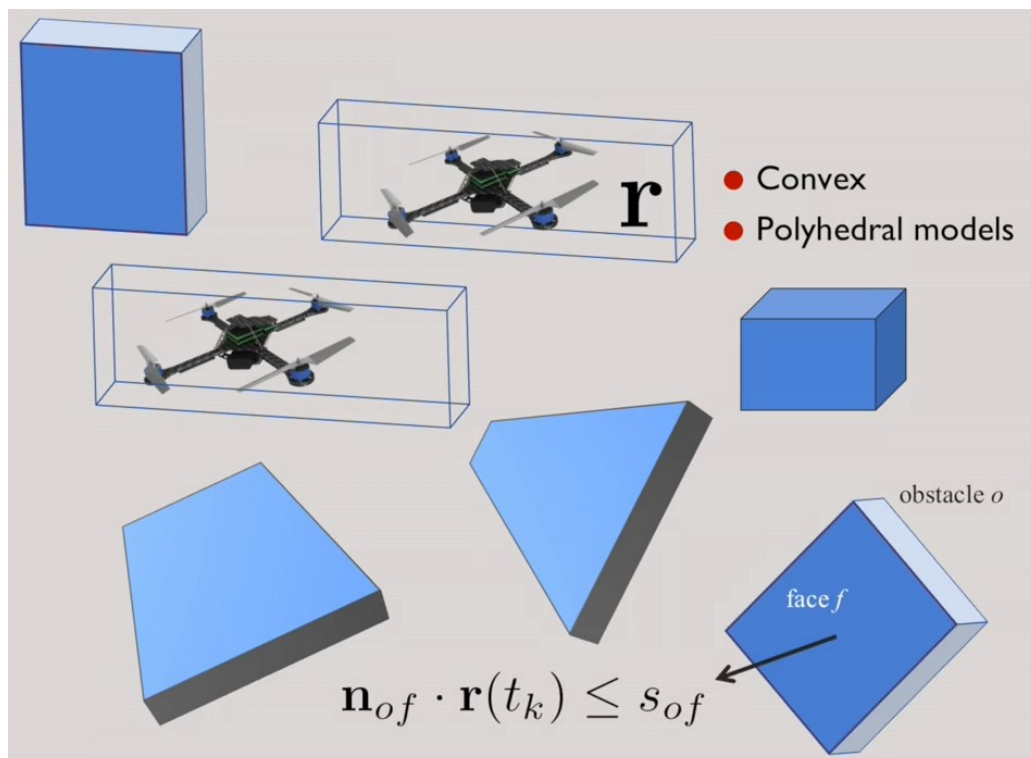
Let's consider what happens when we have obstacles in the environment. Here is an example of a safe minimum-snap trajectory:



A rectangular robot is guided through a set of walls from an initial configuration to a final configuration. The trajectory is obtained by splicing together many minimum-snap trajectories, and is very reminiscent of what we saw earlier when we synthesized cubic splines or  $n^{\text{th}}$ -order splines to go through many different waypoints.

In this case, there are no waypoints. Instead, the robot knows where the obstacles are, so the challenge is to synthesize minimum-snap trajectories in a known environment.

Imagine that we have obstacles in the environment, and in this case, that there's more than one quadrotor flying through the obstacle-filled environment at the same time:



We're going to box off each quadrotor inside virtual rectangular parallelepipeds. We assume that these rectangular parallelepipeds are known, and we want to ensure that they never intersect each other.

In addition, there are many obstacles. We assume that every obstacle is convex and that there are polyhedral models that can characterise the extent of each obstacle. If the obstacles are non-convex, we can always describe them as a union of convex obstacles.

Let's consider a generic obstacle,  $o$ . Because each obstacle is a convex polyhedron, we want to think about collisions in terms of the rectangular parallelepipeds that characterise the quadrotors, and each of the faces characterising the obstacle. For a generic face,  $f$ , on an obstacle,  $o$ , for a generic time instant  $t_k$ , we want to ensure that the convex polyhedron characterizing the robot, and the face  $f$ , do not intersect. That fact is written as a linear inequality:

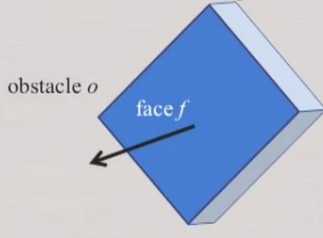
$$\mathbf{n}_{of} \cdot \mathbf{r}(t_k) \leq s_{of}$$

The inequality restricts the positions  $\mathbf{R}$  that can be occupied by the robot. It of course assumes that we know exactly where the obstacle is and we know the normal  $\mathbf{N}$  for each face,  $f$ , for all obstacles, in this case considering obstacle  $o$ .

We can write similar equations for every face, for every obstacle, and every robot. This gives us the set of equations at the top of the figure:

$$\mathbf{n}_{of} \cdot \mathbf{r}(t_k) \leq s_{of} + Mb_{ofk}, \quad \forall f = 1, \dots, n_f(o)$$

$o$	obstacle
$n_f$	number of faces
$t_k$	$k$ th time instant
$b_{ofk}$	binary variable
$M$	large positive constant



$$\mathbf{n}_{of} \cdot \mathbf{r}(t_k) \leq s_{of}$$

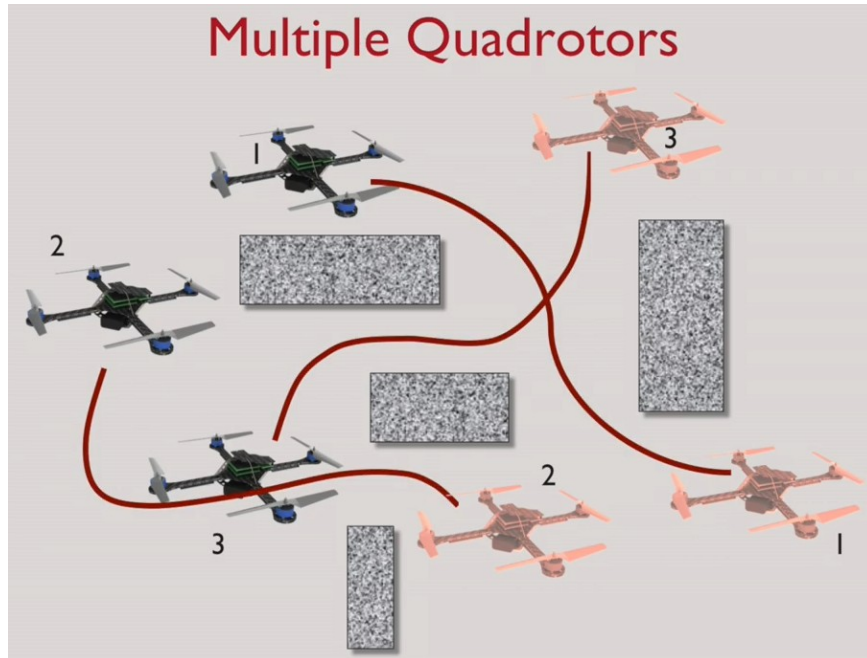
$$\sum_{f=1}^{n_f(o)} b_{ofk} \leq n_f(o) - 1$$

The equation has been modified by adding a positive term on the right hand side. This term consists of a large, positive constant  $M$  and a binary variable  $b$ . This binary variable is characterized by the obstacle  $o$ , the face  $f$ , and the time instant  $t_k$ . The reason for the extra term is quite simple. In order for the robot to avoid a particular obstacle  $o$ , it is enough that this inequality be satisfied for any one of the faces. It is not necessary that the inequality be satisfied for every face on the obstacle. That fact is captured in the inequality at the bottom of the figure.

If there are  $n_f$  faces, we want to ensure that at least one of these binary variables is 0. And that is captured in this inequality. This video shows an example where this approach has been used to synthesize trajectories for suspended payloads, where the vehicle is carrying a payload and the length of the suspended payload is longer than the height of the window it needs to go through. By carefully designing minimum-

snap trajectories, the robot is able to carry the payload through the narrow window without colliding with the environment.

The same technique can be extended to multiple quadrotors. Imagine we have an environment with obstacles. We want quadrotor 1 to go to position 1, quadrotor 2 to go to position 2, and quadrotor 3 to go to position 3, and we don't want these to intersect:



The video shows a simple example in which two quadrotors coordinate their flight in an obstacle filled environment, doing so safely. Once again, every trajectory shown is a minimum-snap trajectory. The approach has been extended to 16 to 20 quadrotors, again, all trajectories are minimum-snap trajectories, and they're guaranteed to be collision free.