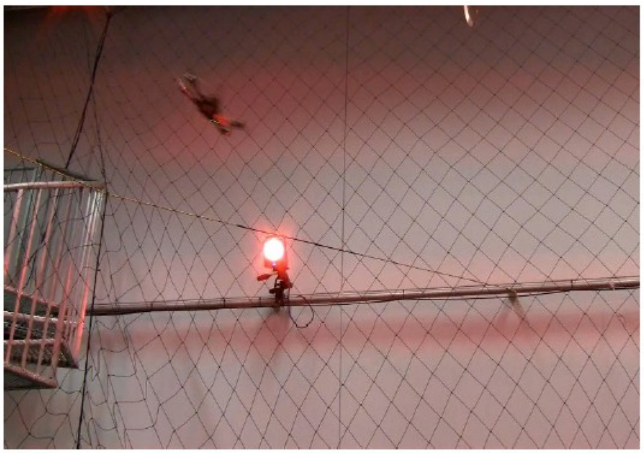
Lecture 4.2

Nonlinear Control

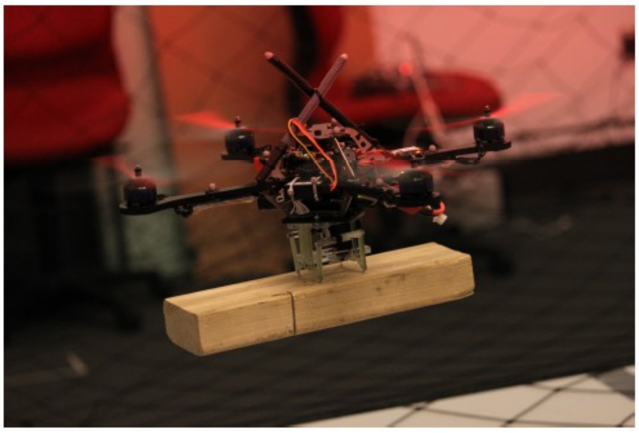
Up until now, we've considered the problem of controlling the quadrotor at states that are not too far from the hover equilibrium configuration. Because of this we've been able to make certain assumptions. We said that the roll and pitch angles were close to zero and that all the velocities were close to zero.

However, as we can see from these pictures, as the vehicle manoeuvres at high speeds and exhibits roll and pitch angles far away from zero, these kinds of controllers are unlikely to work.



We want to overcome these limitations. Specifically, we want to consider non-linear controllers that allow us to control the vehicle far away from the equilibrium (hover) state.

Let's recall the trajectory-control problem. Typically we're given a trajectory, its derivatives up to the second order, we're specified x(t), y(t), z(t), and the yaw angle φ(t).

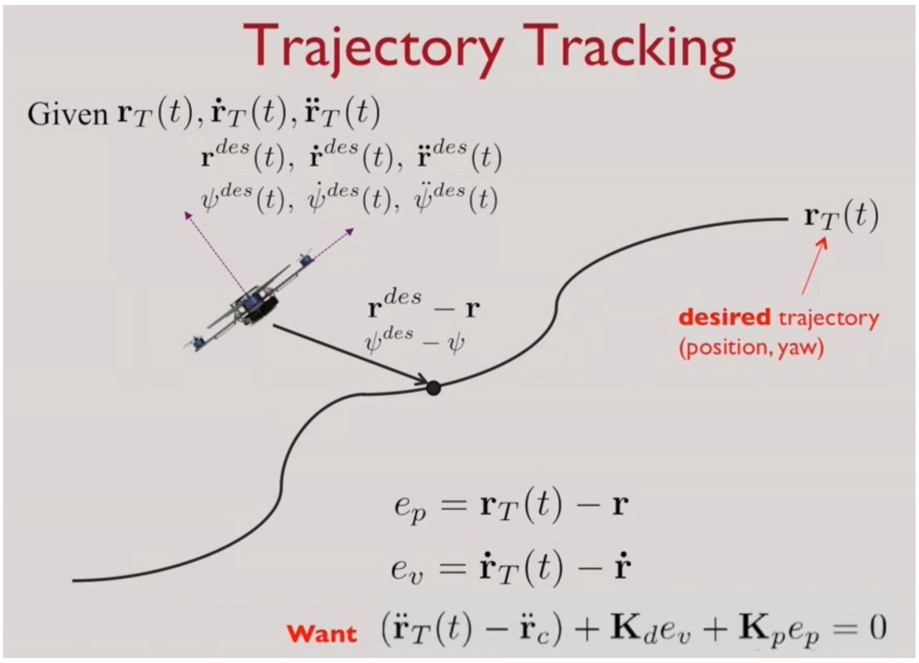


$$r_T(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \varphi(t) \end{bmatrix}$$

We're also able to differentiate them to get higher-order derivatives.

We define an error vector that describes where the vehicle is, relative to where it's supposed to be, and then we try to drive that error exponentially to zero. By insisting on this exponential convergence, we're able to determine the commanded accelerations, which then get translated to inputs that can be applied by the quadrotor through its motors.

This is shown in the figure below:

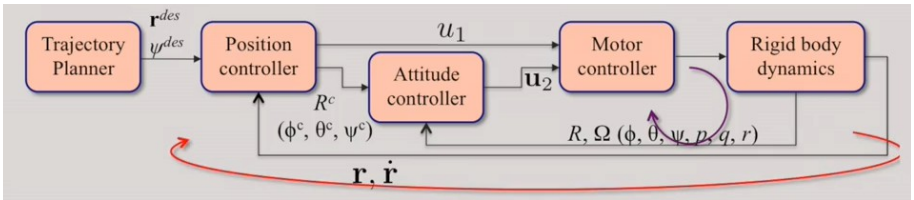

Trajectory Tracking
Given $\mathbf{r}_T(t), \dot{\mathbf{r}}_T(t), \ddot{\mathbf{r}}_T(t)$
$\mathbf{r}^{des}(t),\ \dot{\mathbf{r}}^{des}(t),\ \ddot{\mathbf{r}}^{des}(t)$
$\psi^{des}(t),\ \dot{\psi}^{des}(t),\ \ddot{\psi}^{des}(t)$
$\mathbf{r}^{des} - \mathbf{r}$
$\psi^{des} - \psi$
$\mathbf{r}_T(t)$
desired trajectory (position, yaw)

$$e_p = \mathbf{r}_T(t) - \mathbf{r}$$

$$e_v = \dot{\mathbf{r}}_T(t) - \dot{\mathbf{r}}$$

Want $(\ddot{\mathbf{r}}_T(t) - \ddot{\mathbf{r}}_c) + \mathbf{K}_d e_v + \mathbf{K}_p e_p = 0$

We approached the controller design problem in two stages. Specifically there's an inner loop that addresses the attitude control problem and an outer loop that then addresses the position control problem:



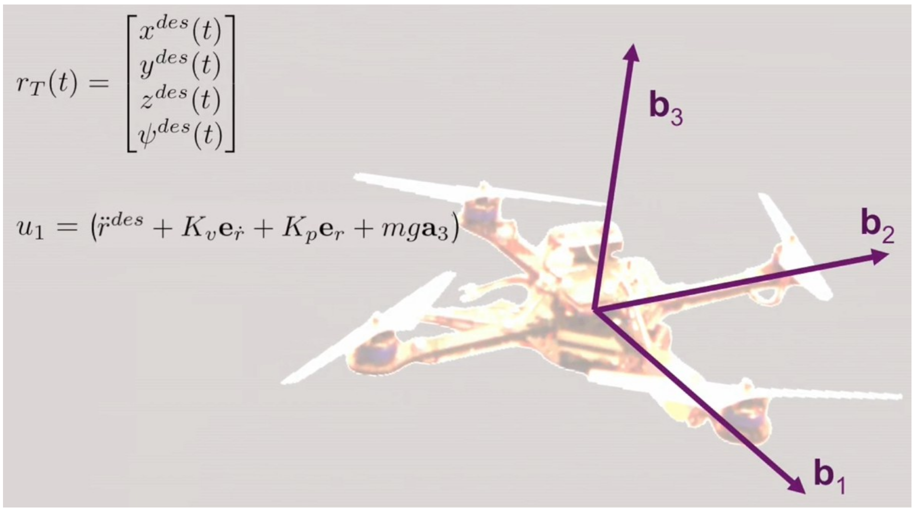
$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}$$

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

The inner loop compares the commanded orientation with the actual orientation and the angular velocities and determines the input u_2. The outer loop looks at the specified position and its derivatives, compares that to the actual position and its derivative, and computes the input u_1.

We want to do the same thing, except now we want to look at trajectories that can exhibit high velocities and high roll- and pitch-angles. That is, trajectories that are not close to the equilibrium or hover state.

We start with a very simple control law for the outer position loop:

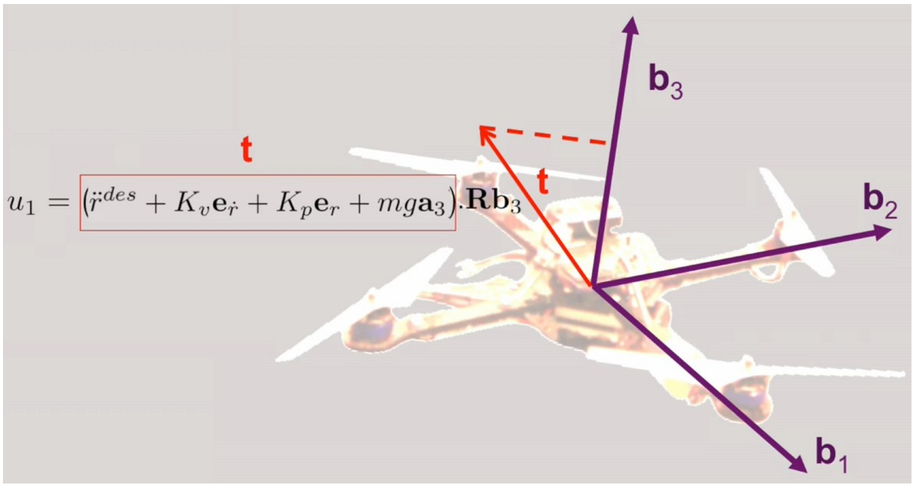


$r_T(t) = \begin{bmatrix} x^{des}(t) \\ y^{des}(t) \\ z^{des}(t) \\ \psi^{des}(t) \end{bmatrix}$
$u_1 = (\ddot{r}^{des} + K_v \mathbf{e}_{\dot{r}} + K_p \mathbf{e}_r + mg\mathbf{a}_3)$
$\mathbf{b}_3$
$\mathbf{b}_2$
$\mathbf{b}_1$

This is essentially a proportional plus derivative control law. We have the proportional term, K_p, which multiplies the error, e_r, and the derivative term, K_v, that multiplies the error, $e_{\dot{r}}$. In addition, there's a feed-forward term, $\ddot{r}^{des}$ which is essentially the acceleration of the specified trajectory. We're also compensating for the gravity vector.

If the system were fully actuated, in other words, if the system could move in any direction, this would essentially solve the control problem. u_1 would be a three dimensional vector and we could control the vehicle, so that the error in the position would go exponentially to zero. However, u_1 is a scalar quantity. In fact, the only direction in which the thrust can be applied is along the b_3 direction.

Because of that, we do the next-best thing. We take this vector and project it along the b_3 direction.



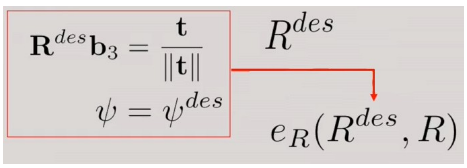
t
$u_1 = \boxed{(\ddot{r}^{des} + K_v \mathbf{e}_{\dot{r}} + K_p \mathbf{e}_r + mg\mathbf{a}_3)}.\mathbf{R}\mathbf{b}_3$
t
$\mathbf{b}_3$
$\mathbf{b}_2$
$\mathbf{b}_1$

That gives us a projection that is close to the desired input, but not quite, because **t** is never perfectly aligned with $\mathbf{b}_3$.

Next, we turn our attention to the attitude-control problem. Knowing that we want t to be aligned with b_3, and knowing that we have very little flexibility in the directions that t can point in; we do the next best thing which is to rotate the vehicle, so that b_3 is aligned with t.

$$R^{des} b_3 = \frac{t}{\|t\|}$$

This gives us the constraint. The desired rotation matrix should be such that the b_3 vector should point in the t direction. We also know what the desired yaw angle needs to be. Armed with these two pieces of information, it's possible to solve for the desired rotation matrix, $\mathbf{R}^{\mathbf{des}}$. Using this information, we can now compute the error in rotation by comparing the actual rotation, R, and the desired rotation, R^{des}:


$\mathbf{R}^{des}\mathbf{b}_3 = \frac{\mathbf{t}}{\|\mathbf{t}\|}$
$\psi = \psi^{des}$
R^{des}
$e_R(R^{des}, R)$

We next follow the moral equivalent of a proportional-plus-derivative control law:

$$u_2 = \omega + I\omega + I(-K_R e_R - K_\omega e_\omega)$$

There are two terms in the parenthesis. One is proportional to the error in rotation, the other is proportional to the error in angular velocity. We also have to compensate for the fact that the vehicle has a non-zero inertia, I, and there are gyroscopic terms, as we know from the Euler equations of motion.

This gives us the control inputs u_1 and u_2. There are a couple of steps in the derivation of the control-law that are not straightforward. First, how do we determine the desired rotation matrix?

We have two pieces of information. First, we want the vector b_3 to be aligned with the vector, t. t is known and we want to find R^{des}, so that this equation is satisfied:

$$R^{des} b_3 = \frac{t}{\|t\|}$$

Second, we want the yaw angle to be equal to the desired yaw angle φ^{des}. We also know from the theory of Euler-angles that the rotation matrix has the form shown here:

$$R = \begin{bmatrix} \cos\varphi\cos\theta - \sin\phi\sin\varphi\sin\theta & -\cos\phi\sin\varphi & \cos\varphi\sin\theta + \cos\theta\sin\phi\sin\varphi \\ \cos\theta\sin\varphi + \cos\varphi\sin\phi\sin\theta & \cos\phi\cos\varphi & \sin\varphi\cos\theta - \cos\theta\sin\phi\cos\varphi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

So the question is: can we use the two equations shown above to solve for the two unknown angles, the roll angle and pitch angle.

The second non trivial step in the derivation of the controller is the calculation of the error term in the rotation matrix: $e_R(R^{des} - R)$

Given the current rotation, R, and the desired rotation, R^{des}, what is the rotation error?

We cannot simply subtract one matrix from another. If we do, we will get a 3x3 matrix, but that matrix is not orthogonal, and therefore is not a rotation matrix.

The correct way to calculate the error is by asking: what is the magnitude of the rotation required to go from the current rotation to the desired rotation?

$$\mathbf{R} \qquad \mathbf{R}^{\mathrm{des}}$$

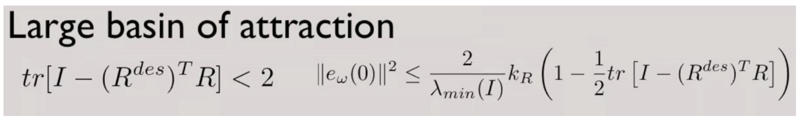
The required rotation is ΔR, given by:

$$\Lambda R = R^{T} R^{des}$$

We can multiply R by ΔR to verify that this is, in fact, true.

Once we have ΔR, then the angle of rotation which is really the magnitude of rotation, and the axis of rotation can both be determined using Rodrigues formula.

It turns out that this controller guarantees that the vehicle is stable through a wide range of angles and velocities. In fact, it's basin of attraction, which is the region from which it will converge to a desired equilibrium point, is almost all of the space of rotations, and a very large set of angular velocities and linear velocities.



Large basin of attraction

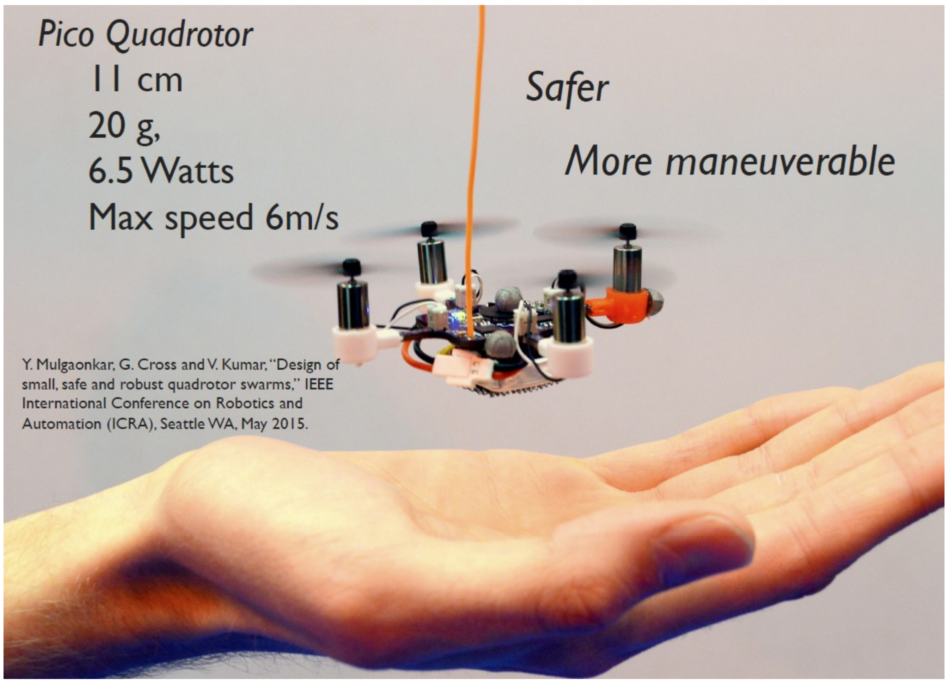
$$tr[I - (R^{des})^T R] < 2 \qquad \|e_\omega(0)\|^2 \leq \frac{2}{\lambda_{min}(I)} k_R \left(1 - \frac{1}{2} tr\left[I - (R^{des})^T R\right]\right)$$

The inequalities here characterise this basin of attraction. The first inequality specifies the set of rotations from which it can converge to the hover configuration, and the second inequality tells us the range of angular velocities from which it can converge to the hover configuration.

Notice that the second inequality has a term in the denominator which relates to the eigenvalues of the inertia tensor. λ_{min} is essentially the smallest eigenvalue of the inertia matrix. The smaller this eigenvalue, the larger the quantity on the right hand side, which tells us that as we scale down the inertia matrix, the set of angular velocities from which the robot can converge to the hover state increases. In other words, the vehicle becomes more stable.

We see this in nature. There are many examples where the advantage of small size can be seen. We saw a video showing the ability of honey bees to react to collisions and recover from them. Again, because their inertia is small, they have a controller whose base of attraction is fairly large, and they're able to exploit the advantages of small size & small inertia to recover from large perturbations. This is one of the motivations for us to build smaller quadrotors.

This picture shows prototype of the Pico Quadrotor, which is only 11cm tip- to-tip, weighs only 20g and has a maximum speed of six ms^{-1}. Vehicles like this are not only safer, but they're also more manoeuvrable.


Pico Quadrotor
11 cm
20 g,
6.5 Watts
Max speed 6m/s
Safer
More maneuverable
Y. Mulgaonkar, G. Cross and V. Kumar, "Design of small, safe and robust quadrotor swarms," IEEE International Conference on Robotics and Automation (ICRA), Seattle WA, May 2015.

We saw a video clip demonstrating the advantage of small size in which two vehicles were commanded to collide at a relative speed of 2 ms^{-1}. They were able to recover from the collision because of the large basin of attraction for the non-linear controller, and the vehicles are robust to perturbations like the ones we saw.

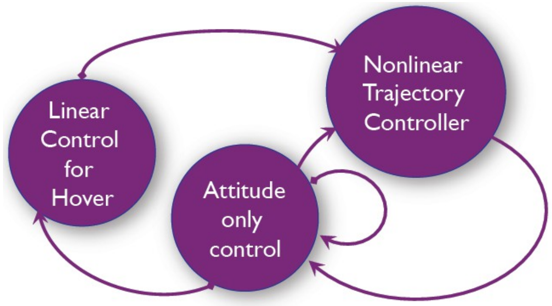
In fact, the size of the basin of attraction scales at:

$$\sim \frac{1}{L^{\frac{5}{2}}}$$

The smaller the characteristic length, L, the larger the basin of attraction.

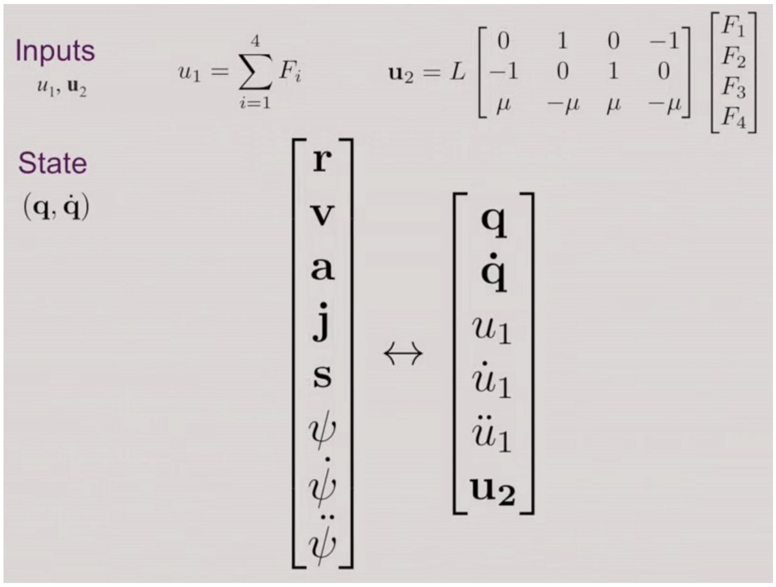
The video showed some experiments that illustrated the advantages of the non-linear controller over the linear controller. In the videos, the roll and pitch angles deviated significantly from the hover position with velocities that scaled up to 5 ms^{-1}.

In one experiment, the non-linear trajectory-controller was used along with two other controllers:



The first of these was the linearised hover-controller, and the second was a rotation-controller or an attitude-controller. By piecing these three controllers together, the robot was able to gather momentum before twisting its body to go through a window, whose width was only a few centimetres more than the height of the robot.

The use of nonlinear controllers also allows us to design more aggressive trajectories. To see that, it's useful to exploit the geometric structure that's inherent in these quadrotors.


Inputs

$u_1, \mathbf{u}_2$

$$u_1 = \sum_{i=1}^{4} F_i \qquad \mathbf{u}_2 = L \begin{bmatrix} 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ \mu & -\mu & \mu & -\mu \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

State

$(\mathbf{q}, \dot{\mathbf{q}})$

$$\begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \mathbf{a} \\ \mathbf{j} \\ \mathbf{s} \\ \psi \\ \dot{\psi} \\ \ddot{\psi} \end{bmatrix} \leftrightarrow \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \\ u_1 \\ \dot{u}_1 \\ \ddot{u}_1 \\ \mathbf{u_2} \end{bmatrix}$$

We want to construct two vectors. On the right-hand side you see the state, q, $\dot{q}$, and the inputs u_1 and u_2, but we've now augmented the state & inputs with two additional terms, $\dot{u}_1$ and $\ddot{u}_1$. The vector on the left-hand side consists of the position, velocity, and acceleration, together with the jerk, j, and snap, s. Again, the jerk is the derivative of the acceleration and the snap the derivative of jerk. φ is the yaw angle, $\dot{\varphi}$ the rate-of-change of yaw and $\ddot{\varphi}$ is the second-derivative of yaw.

It turns out that there's a one-to-one correspondence between the variables in the vector on the left and the variables in the vector on the right. And we can see this from the equations-of-motion:

$$u_1 = m(a_3 - g.b_{3)}$$

$$\dot{u}_1 = mj_3$$

$$\ddot{u}_1 = ms_3 + u_1(q^2 + rp)$$

$$p = \frac{-mj_2}{u_1}$$

$$q = \frac{mj_1}{u_1}$$

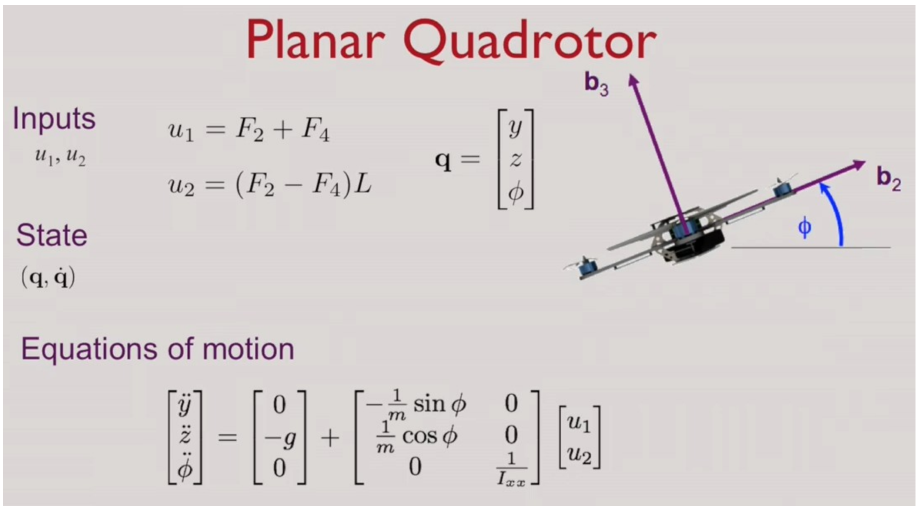
$$\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

For example, if we know the acceleration in the z-direction a_3, we can calculate u_1 and by differentiating that equation, we get an expression for $\dot{u}_1$. Differentiating it again yields an expression for $\dot{u}_1$. All the terms on the right-hand side of these equations are obtained from the vector consisting of position, velocity, acceleration, jerk, snap, the yaw angle, its derivative, and its second derivative.

This shows how we can get the terms in the vector on the right-hand side from the terms in the vector on the left-hand side. We can go through a similar exercise to go the other way around.

What are the implications of this in terms of trajectory planning? What it really means is that if we know the position, the yaw angle, & its derivatives, we can determine the state and the input.

Let's consider the equations for a planar quadrotor in which it's easier to visualize the mapping between these two vectors:



Planar Quadrotor

Inputs u_1, u_2

$$u_1 = F_2 + F_4$$

$$u_2 = (F_2 - F_4)L$$

$$\mathbf{q} = \begin{bmatrix} y \\ z \\ \phi \end{bmatrix}$$

State $(\mathbf{q}, \dot{\mathbf{q}})$

Equations of motion

$$\begin{bmatrix} \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{m}\sin\phi & 0 \\ \frac{1}{m}\cos\phi & 0 \\ 0 & \frac{1}{I_{xx}} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

The equivalence between the two vectors is due to a property called ***differential flatness***. The key idea is that all the state variables and the inputs can be written as smooth functions of the so-called flat outputs and their derivatives. The flat outputs for a quadrotor are simply the y- and z-position. This equivalence works both ways.

$$\begin{bmatrix} y \\ z \\ \dot{y} \\ \dot{z} \\ \ddot{y} \\ \ddot{z} \\ \dddot{y} \\ \dddot{z} \\ y^{(iv)} \\ z^{(iv)} \end{bmatrix} \Leftrightarrow \begin{bmatrix} y \\ z \\ \phi \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ u_1 \\ \dot{u}_1 \\ \ddot{u}_1 \\ u_2 \end{bmatrix}$$

If we know the state vector, the input u_1, its first- and second-derivative, and the input u_2, we can recover the flat outputs and their derivatives:

$$\begin{array}{ccc} \text{Flat outputs} & \text{State} & \text{Input} \\ \begin{bmatrix} y \\ z \end{bmatrix} & \begin{bmatrix} y \\ z \\ \phi \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \end{bmatrix} & \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{array}$$

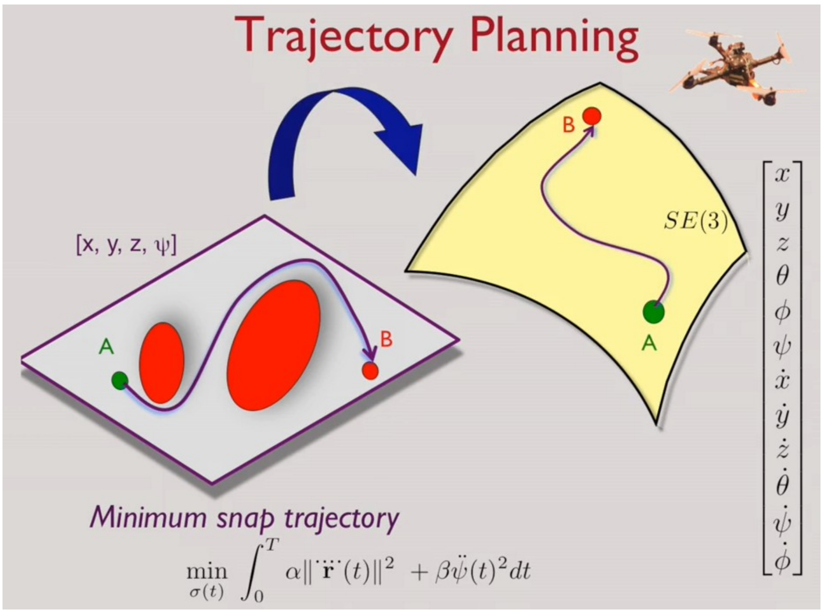
$$\begin{bmatrix} \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} -\frac{1}{m}\sin\phi \\ \frac{1}{m}\cos\phi \end{bmatrix} u_1$$

$$\begin{bmatrix} y^{(iii)} \\ z^{(iii)} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} -u_1\dot{\phi}\cos\phi - \dot{u}_1\sin\phi \\ -u_1\dot{\phi}\sin\phi + \dot{u}_1\cos\phi \end{bmatrix}$$

$$\begin{bmatrix} y^{(iv)} \\ z^{(iv)} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} -\sin\phi & -\frac{u_1}{I_{xx}}\cos\phi \\ \cos\phi & -\frac{u_1}{I_{xx}}\sin\phi \end{bmatrix} \begin{bmatrix} \ddot{u}_1 \\ u_2 \end{bmatrix} + \frac{1}{m} \begin{bmatrix} -2\dot{u}_1\dot{\phi}\cos\phi + u_1\dot{\phi}^2\sin\phi \\ -2\dot{u}_1\dot{\phi}\sin\phi - u_1\dot{\phi}^2\cos\phi \end{bmatrix}$$

This relationship between the two vectors is called a ***diffeomorphism***. The term diffeomorphism refers to a smooth map between these two vectors, and the definition can be found in most differential-geometry textbooks. Similarly, for a three-dimensional quadrotor, you have a longer vector on both sides, but once again, there's a one-to-one relationship between these vectors. If we know the flat outputs and their derivatives, you can recover the state variables and the inputs.

The three-dimensional quadrotor is differentially flat, & the implication for trajectory planning is quite simple. Rather than think about the dynamics of the system when we plan trajectories, we can focus our attention on the flat space on the left-hand side. We can think about mapping obstacles on to this flat space and planning trajectories that are safe, provided that these trajectories are smooth.


Trajectory Planning
[x, y, z, ψ]
A
B
$SE(3)$
B
A
$[x\ y\ z\ \theta\ \phi\ \psi\ \dot{x}\ \dot{y}\ \dot{z}\ \dot{\theta}\ \dot{\psi}\ \dot{\phi}]$
Minimum snap trajectory
$\min_{\sigma(t)} \int_0^T \alpha \|\ddot{\ddot{\mathbf{r}}}(t)\|^2 + \beta\ddot{\psi}(t)^2 dt$

If they are smooth, since these are the flat outputs, we're guaranteed to be able to find feasible state-vectors and inputs that correspond to these trajectories. So once we find smooth trajectories, and, as we've discussed before, we prefer to make these smooth trajectories *minimum-snap trajectories*, we can then transform them into the real space with state-vectors and inputs. Again, we're guaranteed that these state-vectors and inputs will satisfy the dynamic equations-of-motion.

The video showed an example of a minimum-snap trajectory from a starts position to a goal via an intermediate waypoint. The trajectory was generated in the flat-space, consisting of the flat-outputs, without specific knowledge of the dynamics, but because these trajectories are smooth, they could be automatically transformed into trajectories that are realistic and can be executed by a real robot.

This reduces the motion-planning problem to a problem in computational geometry. If we know how to synthesize smooth curves going through specified waypoints, a subject that we've already addressed in previous lectures, we can easily transform these curves into dynamically feasible trajectories that a non-linear controller can faithfully execute.