

*

Opt1.1 *

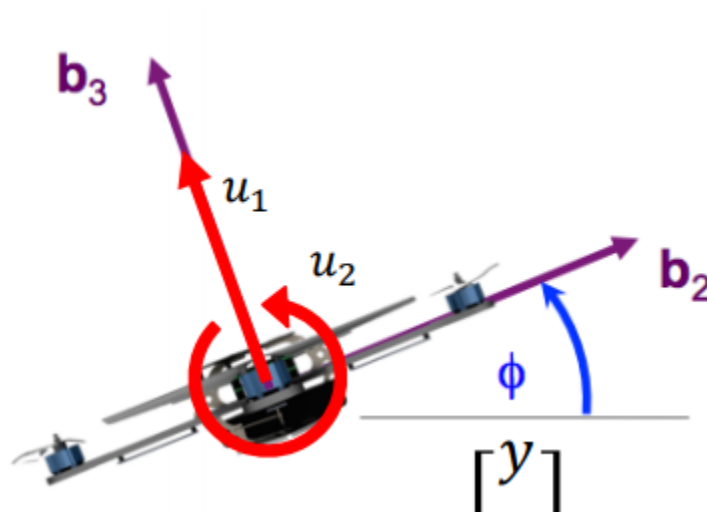
Opt1.1

Lecture 3: (Quadrotor Control)

Scribes: Ritika Shrivastava, Nanyu Zhao

3.1 Planar Quadrotor Model

Quadrotor has a central area that holds a payload and it has four arm with propellers. This is different from a rotor craft which has a front rotor and a tale rotor, where the tale rotor is used for stabilization in the yaw direction.



When each of the propellers of the quadrotor spin, they generate an upward force. For now, let's focus on the 2-dimensional model of the quadrotor. In the image below, the x-axis is coming out of the plane. We will be focusing on the two propellers. Each one of the propellers generate a force that is perpendicular to the axis of the quadrotor. In the image below, let's call this F_1 and F_2 . The sum of F_1 and F_2 is the *net upward force* also called u_1 .

$$u_1 = F_1 + F_2$$


If F_1 and F_2 are the same there is no lateral movement (*torque*) about the center of mass of the quadrotor. If there is a difference between F_1 and F_2 , there is a torque pointing in the b_2 direction. Using this information we can understand that the net torque (u_2) can be found with the following equation.

$$u_2 = (F_1 - F_2) * l$$

Where l is the length of the arm of the quadrotor. The net force (u_1) is the sum of the two forces and the torque (u_2) is proportional to the difference between the two forces. So we can see that u_1 is the thrust as it controls the two positional variables y and z . Meanwhile u_2 is the torque and controls the ϕ component. The thrust is perpendicular to the body. The y -direction is pointing to the right and the z -direction is pointing upwards. When the equations are put together we get the following formulas:

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

The equation above are Newton and Euler equations and will be derived in a later class. To get the state space, you can stack the $SE(2)$ variables and you should get the following x -vector.



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y \\ z \\ \phi \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \end{bmatrix}$$

Now let $x_1 = y$, $x_2 = z$ and so on. By doing this we get the following first derivative of x .

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -m^{-1} \sin x_3 & 0 \\ m^{-1} \cos x_3 & 0 \\ 0 & I_{xx}^{-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Notice how the the second derivative of y , z , and ϕ have been incorporated into the last three elements of this vector.

3.2 Affine Nonlinear System

We will be learning about nonlinear systems as they are important and commonly show up in robotics. The form of the equations is below:

State \mathbf{x} and input \mathbf{u}

State equations $\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}$

$$\dot{\mathbf{x}} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ 0 \\ -g \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -m^{-1} \sin x_3 & 0 \\ m^{-1} \cos x_3 & 0 \\ 0 & I_{xx}^{-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$f(x)$ which belongs to R^6 and $g(x)$ is affine. Here the control input is affine and the state is nonlinear.

3.3 Linearized Dynamics

To linearize, means to choose values for u_1 and u_2 so that y , z and ϕ stay constant. So at the equilibrium $\phi = 0$ and $u_2 = 0$. Below is an image with the equilibrium point.

Equilibrium configuration

$$\mathbf{q}_e = \begin{bmatrix} y_0 \\ z_0 \\ 0 \end{bmatrix}, \mathbf{x}_e = \begin{bmatrix} \mathbf{q}_e \\ \mathbf{0} \end{bmatrix}$$

Equilibrium hover condition

$$\begin{aligned} y_0, z_0 \\ \phi_0 &= 0 \\ u_{1,0} &= mg, u_{2,0} = 0 \end{aligned}$$

Lets try linearizing about the equilibrium point. From a heuristic understanding, this means to take smaller perturbations about ϕ , y , and z . This gives you the equations on the right.

Nonlinear dynamics

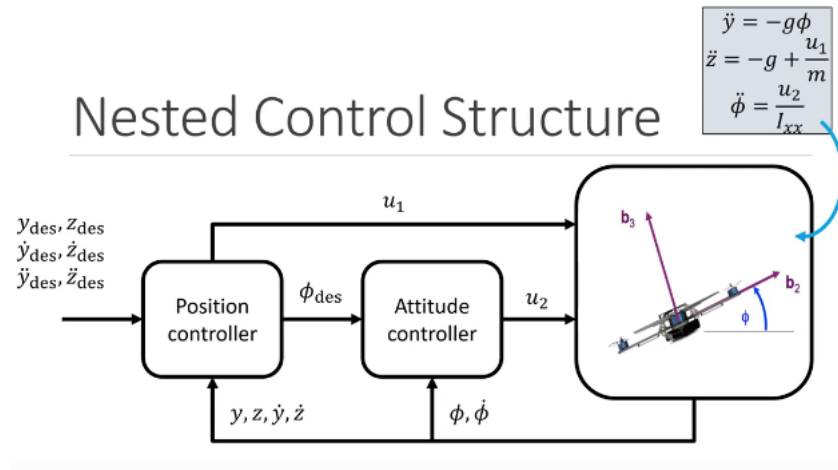
$$\begin{aligned} \ddot{y} &= -\frac{u_1}{m} \sin(\phi) \\ \ddot{z} &= -g + \frac{u_1}{m} \cos(\phi) \\ \ddot{\phi} &= \frac{u_2}{I_{xx}} \end{aligned}$$

Linearized model

$$\begin{aligned} \ddot{y} &= -g\phi \\ \ddot{z} &= \frac{u_1}{m} \\ \ddot{\phi} &= \frac{u_2}{I_{xx}} \end{aligned}$$

It is interesting to see that \ddot{z} is proportional to u_1 and that \ddot{y} is proportional to ϕ .

3.4 Nested Control Structure



As a results of the information above, the attitude of a plane is controlled first and then the position is controlled. This is because of the following control equations.

Recall for a second order system $\ddot{\mathbf{e}} + K_d \dot{\mathbf{e}} + K_p \mathbf{e} = 0$

For any configuration variable q we have

$$(\ddot{q}_{\text{des}} - \ddot{q}) + K_{d,q}(\dot{q}_{\text{des}} - \dot{q}) + K_{p,q}(q_{\text{des}} - q) = 0$$

3.4.1 Control Equations

The image below explains the seconds order system for any configuration q

Recall for a second order system $\ddot{\mathbf{e}} + K_d \dot{\mathbf{e}} + K_p \mathbf{e} = 0$

For any configuration variable q we have

$$(\ddot{q}_{\text{des}} - \ddot{q}) + K_{d,q}(\dot{q}_{\text{des}} - \dot{q}) + K_{p,q}(q_{\text{des}} - q) = 0$$

3.4.2 Control Equations for Quadrotors

Now when we take the linearized equations. We can see that we can directly control z , but this wont take into account error so we are using \ddot{z}_c . We get the following attitude and Z-position controllers.

Lateral dynamics <ul style="list-style-type: none"> $\ddot{y} = -g\phi$ $\ddot{\phi} = \frac{u_2}{I_{xx}}$ 	Vertical dynamics <ul style="list-style-type: none"> $\ddot{z} = \frac{u_1}{m}$
Desired attitude <ul style="list-style-type: none"> $\phi_{\text{des}} = -\frac{\ddot{y}_c}{g}$ $\dot{\phi}_{\text{des}} = 0$ $\ddot{\phi}_{\text{des}} = 0$ 	Z-position controller <ul style="list-style-type: none"> $u_1 = m(\ddot{z}_c)$
Attitude controller <ul style="list-style-type: none"> $u_2 = I_{xx}\ddot{\phi}_c$ 	

When we put these equations together we end up getting the following control equations.

Control equations

$$u_1 = m \left(\ddot{z}_{\text{des}} + k_{d,z}(\dot{z}_{\text{des}} - \dot{z}) + k_{p,z}(z_{\text{des}} - z) \right)$$

$$u_2 = I_{xx} \left(\ddot{\phi}_{\text{des}} + k_{d,\phi}(\dot{\phi}_{\text{des}} - \dot{\phi}) + k_{p,\phi}(\phi_{\text{des}} - \phi) \right)$$

$$\phi_{\text{des}} = -\frac{1}{g} \left(\ddot{y}_{\text{des}} + k_{d,y}(\dot{y}_{\text{des}} - \dot{y}) + k_{p,y}(y_{\text{des}} - y) \right)$$

This gives us three sets of PD gains. We can systematically tune these values using step response to get the thrust, roll, and position (which depends on the roll being well-tuned).

The advantage of using these three equations is that the information is now decoupled and can be used to tweak each of the values.

3.5 Useful math concepts for this class

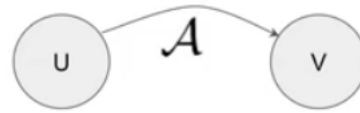
3.5.1 Linear Map==Linear Operator

Matrix is linear operator and all linear operator can be expressed by matrix multiplication

A linear operator $\mathcal{A} : U \rightarrow V$ is just a linear function*

Range Space: $\mathcal{R}(\mathcal{A}) := \{v | v = \mathcal{A}(u), u \in U\} \subset V$

Null Space: $\mathcal{N}(\mathcal{A}) := \{u | \mathcal{A}(u) = \emptyset v\} \subset U$



The null space and range space are themselves linear operators.

All linear operators can be expressed by matrix multiplication!

$$\mathcal{A} : v = Au$$

We can also examine all matrices as linear operators

Additionally, the image below provides an understanding to what a matrix is.

A matrix is a collection of vectors: $A = [v_1, v_2, \dots, v_m], v_i \in \mathbb{R}^n, u \in \mathbb{R}^m$

Thus, left-multiplying a vector by the matrix is just a weighted sum of the columns.

$$Ax = x_1v_1 + x_2v_2 \dots + x_mv_m$$

A set of vectors is *linearly dependent* iff $\exists \{\alpha_1 \dots \alpha_m\}$ s.t. $\sum_{i=1}^m \alpha_i v_i = 0$

Since $\mathcal{R}(\mathcal{A})$ and $\mathcal{N}(\mathcal{A})$ are linear operators, they're also matrices

- Rank(A) := dimension (number of columns) of $\mathcal{R}(\mathcal{A})$
- Nullity(A) := dimension of $\mathcal{N}(\mathcal{A})$

3.5.2 Rank Nullity Theorem

(We only have limited use on this)

Let $\mathcal{A} : U \rightarrow V$

$$\text{Rank}(\mathcal{A}) + \text{Nullity}(\mathcal{A}) = \dim(U)$$

This is one of the most important theorems in linear algebra!

3.5.3 Eigenvalues and Eigenvectors

Eigenvectors are the “principal axes” of a matrix. Applying the matrix to an eigenvector will only scale it, not rotate it.

$$Av = \lambda v \longrightarrow (A - I\lambda)v = \emptyset$$

The amount an eigenvector gets stretched is its corresponding *eigenvalue*.

If all the eigenvectors of a matrix span \mathbb{R}^n they're called an *eigenbasis*

λ are the eigenvalues, ν are the eigenvectors, I is a identity matrix.

3.5.4 Matrix Diagonalization

(We only have limited use on this)

This is a way to decompose matrix when eigenvectors of the matrix are linearly independent

$$Av = v\lambda \longrightarrow AV = V\Lambda \longrightarrow A = V\Lambda V^{-1}$$

Where

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \quad \Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

3.5.5 Jordan Normal Form

This is a way to decompose matrix when eigenvectors of the matrix are linearly dependent

Algebraic Multiplicity: The number of times an eigenvalue λ appears

Geometric Multiplicity: The rank of the eigenvectors corresponding to λ

If algebraic multiplicity > geometric multiplicity for any λ , an eigenbasis cannot be constructed, and thus the matrix cannot be diagonalized.

Instead we use the Jordan Normal Form.

$$A = VJV^{-1}$$

$$J = \begin{bmatrix} \boxed{\begin{matrix} \lambda_1 & 1 \\ & \lambda_1 \end{matrix}} & & & \\ & \boxed{\begin{matrix} \lambda_2 & 1 \\ & \lambda_2 \end{matrix}} & & \\ & & \ddots & \\ & & & \boxed{\begin{matrix} \lambda_n & 1 \\ & \lambda_n \end{matrix}} \end{bmatrix}$$

Figure from: Jacob Scholbach via Wikipedia. https://en.wikipedia.org/wiki/Jordan_normal_form#/media/File:Jordan_blocks.svg

3.5.6 Matrix Exponential

Matrix analogous to the ordinary exponential function.

3.5.7 Matrix Properties

Positive Definite/Semi-definite matrix: All eigenvalue of the matrix are larger than 0/greater than or equal to 0

Negative Definite/Semi-definite matrix: All eigenvalue of the matrix are smaller than 0/smaller than or equal to 0 Additional properties can be seen below.

Symmetric / Hermitian: $A = A^T$, $A = A^*$

- All hermitian matrices can be represented as $A = B^T B$

Positive Definite: $A \succ 0$

- All EVs of A are greater than zero. $x^T A x > 0, \forall x \neq \emptyset$

Positive Semidefinite: $A \succeq 0$

- All EVs of A are greater than or equal to zero. $x^T A x \geq 0, \forall x \neq \emptyset$

Negative Definite / Semidefinite: $A \prec 0$, $A \preceq 0$

- Same as above, but EVs are negative

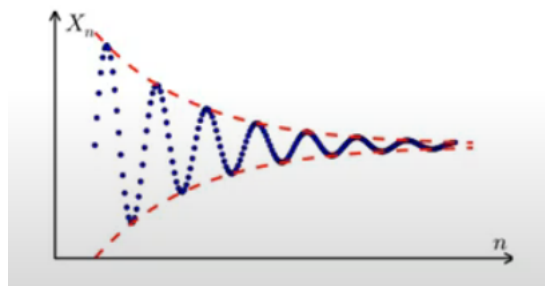
3.6 Linear Dynamical System

3.6.1 Linear Autonomous System: $\dot{x} = Ax$

we can solve for x in $\dot{x} = Ax$ by assuming $x(t) = e^{At}x(0)$ (integrating with matrix exponential).

3.6.2 Stability

The concept of stability states what the origin is stable if small perturbations are not amplified. A linear criterion for stability is called the Routh Hurwitz stability, and a non-linear criterion example is Lyapunov stability. Also note that stability never refers to a system. It only applies to an equilibrium point. Note the image below has a stable system.



Looking back at the math we covered before, we can see that when we integrate a matrix, that our goal is stability or convergence. In another sense if time goes to infinity, we do not want our system to go to infinity.

$$\| \lim_{t \rightarrow \infty} e^{At} x(0) \|_2 \neq \infty$$

If our dynamics is diagonalizable, we can pull out the change of basis. Using this we can see if $e^{\lambda t}$ is stable at time goes to infinity.

$$e^{V \Lambda V^{-1}} = V e^{\Lambda} V^{-1} = \begin{bmatrix} e^{\lambda_1} & 0 & \dots & 0 \\ 0 & e^{\lambda_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{\lambda_n} \end{bmatrix}$$

$$\begin{cases} \lambda > 0 & \lim_{t \rightarrow \infty} e^{\lambda t} \rightarrow \infty, & \text{unstable} \\ \lambda = 0 & \lim_{t \rightarrow \infty} e^{\lambda t} \rightarrow 1, & \text{marginally stable} \\ \lambda < 0 & \lim_{t \rightarrow \infty} e^{\lambda t} \rightarrow 0, & \text{stable} \end{cases}$$

In the situation where the matrix is not diagonalizable you can use the Jordan form mentioned above. When you take the exponential, you can break up all the Jordan blocks, and integrate each separately. This means that the repeated eigenvalue on the imaginary axis (real part zero) that are not diagonalizable are not stable. This means that there is some subtlety about checking the eigenvalues of linear systems to see if the system converges.

3.6.3 Lyapunov Stability

Now, let's consider the stability of non-linear systems. We also need to take into account the kinetic energy of a function. So Lyapunov looked up at the change in system energy. We can define the energy function, which must be positive semi-definite. $V(x) > 0$. From there we can look at how the energy of that function changes over time.

$$\frac{d}{dt} V(x) = \frac{\partial V}{\partial x} \dot{x} = L_f V$$

If our energy decrease at every point x , then our system is globally stable. Now to derive, the Lyapunov Equation, we can do the steps in the image below.

We choose a quadratic energy function

$$V(x) = x^T P x, \quad P \succ 0$$

And differentiate to get

$$\dot{V}(x) = x^T (A^T P + P A) x = x^T Q x$$

If Q is negative definite, then the system is stable

If Q is negative semi-definite, then the system is marginally stable, or *stable in the sense of Lyapunov (SISL)*

Note: Rather than using matrices we can use transfer functions to examine systems in the frequency domain. Control done in the frequency domain is called classical controls. This will not be covered in class, but might show up on research papers and textbooks.

3.6.4 State Feed Back Control

Given x as the input and u as the state

In state feedback control, we make the control a linear function of state

$$u = -Kx$$

Our driven system now becomes an autonomous system with

$$\dot{x} = (A - BK)x$$

By varying K, we can actually change all the eigenvalues of our new system, thus allowing us to (theoretically) determine both stability and rate of convergence for all modes of the system.

3.6.5 LQR - Linear Quadratic Regulator

LQR is a way to find K for state feed back control.

The “optimal” way to find K

Define cost matrices Q, R

$$Q \in \mathbb{R}^{n \times n}, \quad R \in \mathbb{R}^{u \times u}, \quad Q, R \succ 0$$

LQR finds a K which minimizes the cost function

$$\int_0^\infty (x^T Q x + u^T R u + 2x^T N u) dt$$

Use command “lqr” in Matlab

3.7 Citation

<https://drive.google.com/file/d/1a2bu7ezkYFQNUMVePsiIHGo614vbiCCy/view>

https://ucb-ee106.github.io/106b-sp21site/assets/lec/Lecture3_LinearControl_edit.pdf