

Attitude control and autonomous navigation of quadcopters

Pantelis Sopasakis

KU Leuven, Dept. of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analysis.

Outline

▶ Problem statement and objectives

▶ Control Theory

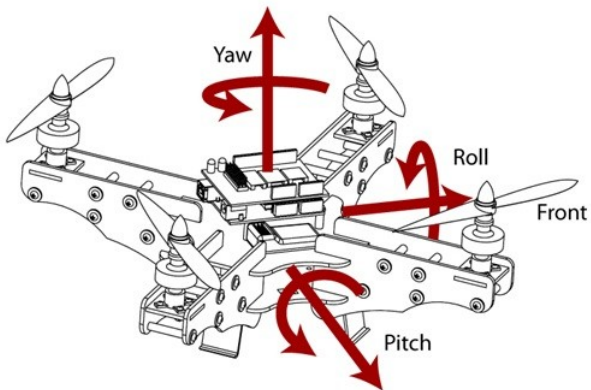
- ✓ State space systems
- ✓ Stabilisation with linear state feedback
- ✓ Linear Quadratic Regulator
- ✓ Reference tracking
- ✓ Linear State Observers

▶ Application

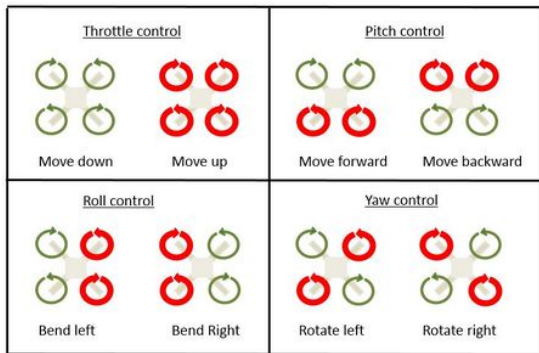
- ✓ Quaternions and rotations
- ✓ Attitude dynamics
- ✓ Implementation
- ✓ Autonomous navigation

I. Attitude control problem statement


Attitude control



Intuition



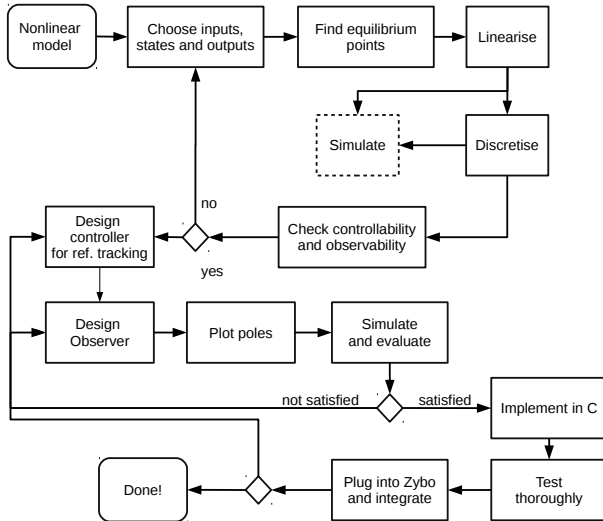
 Normal Speed

 High Speed

Attitude control: Problem statement

1. By controlling the voltages V_1, V_2, V_3 and V_4 at the four motors and obtaining measurements from the *inertial measurement unit* (IMU) make the quadcopter hover, that is $\phi = 0$ (no pitch), $\theta = 0$ (no roll), $\psi = 0$ or $\dot{\psi} = 0$ (no yaw).
2. Follow the reference signals $(\phi_r, \theta_r, \dot{\psi}_r)$ and the throttle reference signal τ_r from the RC.

Attitude control: Workflow



II. State Space Systems

States, inputs and outputs

- ▶ **States:** Those variables that provide all we would need to know about our system to fully describe its future behaviour and evolution in time,
- ▶ **Outputs:** Variables, typically transformations of the state variables, that we can measure
- ▶ **Inputs:** Variables which we have the ability to manipulate so as to control the system
- ▶ **Disturbances:** Signals which affect the system behaviour and can be thought of as input variables over which we do not have authority (e.g., weather conditions).

State space systems

State space systems are described in continuous time by

$$\dot{x}(t) = f(x(t), u(t)),$$

$$y(t) = h(x(t), u(t)),$$

where $x \in \mathbb{R}^{n_x}$ is the system *state* vector, $u \in \mathbb{R}^{n_u}$ is the *input* vector and $y \in \mathbb{R}^{n_y}$ is the vector of *outputs*.

Discrete-time state space systems

The discrete time version is

$$\begin{aligned}x_{k+1} &= f(x_k, u_k), \\ y_k &= h(x_k, u_k),\end{aligned}$$

where $k \in \mathbb{N}$.

Why state space?

- ▶ Can be directly derived from first principles of physics
- ▶ Suitable for multiple-input multiple-output systems
- ▶ Enable the study of controllability, observability and other interesting properties
- ▶ Time-domain representation: more intuitive

Equilibrium points

A pair (x^e, u^e) is called an **equilibrium point** of the continuous-time system

$$\dot{x}(t) = f(x(t), u(t)),$$

if

$$f(x^e, u^e) = 0.$$

If $u(t) = u^e$ and $x(0) = x^e$, then $x(t) = x^e$ for all $t > 0$.

Equilibrium points

A pair (x^e, u^e) is called an **equilibrium point** of the discrete time system

$$x_{k+1} = f(x_k, u_k)$$

if

$$f(x^e, u^e) = x^e.$$

Then, if $u_k = u^e$ for all k and $x_0 = x^e$, then $x_k = x^e$ for all $k \in \mathbb{N}$.

Discretisation

Excerpt from Wikipedia:

*In mathematics, **discretization** concerns the process of transferring continuous functions, models, and equations into discrete counterparts.*

Discretisation: the Euler method

Recall that

$$\dot{x}(t) = \lim_{h \rightarrow 0} \frac{x(t+h) - x(t)}{h},$$

so, for small h

$$\dot{x}(t) \approx \frac{x(t+h) - x(t)}{h}.$$

Using this approximation

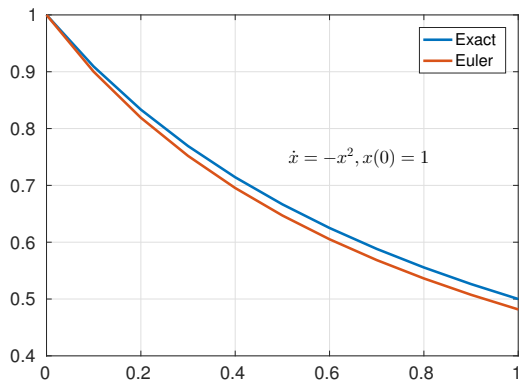
$$\dot{x}(t) = f(x(t), u(t)),$$

is approximated — with sampling period h — by the discrete system

$$x_{k+1} = x_k + hf(x_k, u_k),$$

where x_k is an approximation for $x(kh)$.

Discretisation: the Euler method



Discretisation: linear systems

When the continuous-time system has the form

$$\dot{x}(t) = Ax(t) + Bu(t),$$

then we know that its solution is

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau$$

Assume that $u(t)$ is constant on $[kh, (k+1)h)$ and equal to $u(k) = u_k$ and define $A_d = e^{Ah}$ and $B_d = \int_0^h e^{A\tau}Bd\tau$. Then

$$x_{k+1} = A_dx_k + B_du_k,$$

is an exact discretisation of the original system.

Linearisation

The best linear approximation of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ about a point x_0 is given by

$$f(x) \approx f(x_0) + Jf(x_0)(x - x_0)$$

For functions $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ this reads

$$f(x, u) \approx f(x_0, u_0) + J_x f|_{(x_0, u_0)} (x - x_0) + J_u f|_{(x_0, u_0)} (u - u_0).$$

Jacobian matrix

Let $x = (x_1, x_2, \dots, x_n)$ and

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{bmatrix}$$

Jacobian matrix

Let $x = (x_1, x_2, \dots, x_n)$ and

$$f(x) = \begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) \end{bmatrix}$$

The **Jacobian matrix** of f is a function $Jf : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ defined as

$$Jf(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{bmatrix}$$

Linearisation

Consider the continuous time system

$$\dot{x}(t) = f(x(t), u(t)),$$

choose an equilibrium point (x^e, u^e) and define

$$\Delta x(t) = x(t) - x^e,$$

$$\Delta u(t) = u(t) - u^e$$

Then, it is easy to verify that the system linearisation is written as

$$\frac{d}{dt} \Delta x(t) = A \Delta x(t) + B \Delta u(t),$$

where $A = (J_x f)(x^e, u^e)$ and $B = (J_u f)(x^e, u^e)$.

Simulations

In MATLAB we may use

1. ode23: fast and of decent accuracy
2. ode45: less fast but more reliable
3. Simulink and S-functions

In C++ we may use `odeint`.

In Python try `scipy.integrate.ode`.

III. Stabilisation of Linear Systems

Controllability

Linear time-invariant (LTI) discrete-time systems:

$$x_{k+1} = Ax_k + Bu_k$$

We say that the system is **controllable** if for every $x^1, x^2 \in \mathbb{R}^{n_x}$ there is a finite sequence of inputs (u_0, u_1, \dots, u_N) which can steer the system state from x^1 to x^2 (we can move the system to any point x^2 starting from any point x^1).

Controllability

The system is controllable iff

$$\text{rank} \underbrace{\begin{bmatrix} B & AB & A^2B & \cdots & A^{n_x-1}B \end{bmatrix}}_{\mathcal{C}(A,B)} = n_x.$$

This is the most popular criterion, but there exist a lot more. We also often say that the pair (A, B) is controllable.

Why using rank is a bad idea

Take for example the pair (A, B) with

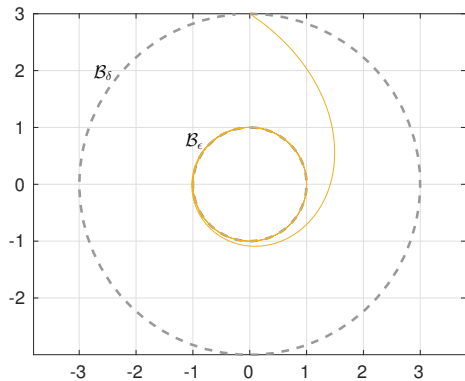
$$A = \begin{bmatrix} 1 & \epsilon \\ \epsilon & \epsilon \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Then, the controllability matrix is

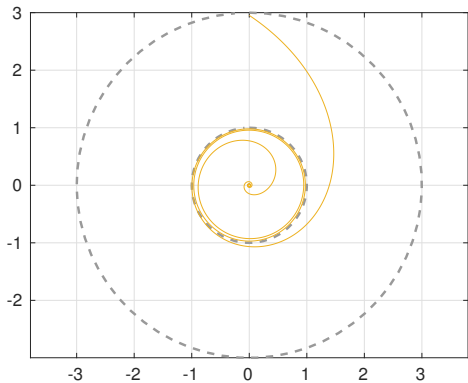
$$\begin{bmatrix} 1 & \epsilon \\ 0 & \epsilon \end{bmatrix}$$

whose rank is 2 but its singular values are 1 and ϵ .

Stability



Asymptotic Stability



Stability analysis of linear systems

Consider the linear discrete-time system

$$x_{k+1} = Ax_k.$$

The origin ($x^e = 0$) is an *asymptotically stable* equilibrium point for the above system if and only if all eigenvalues of A are strictly within the unit circle, that is

$$|\lambda_i| < 1.$$

In MATLAB one may find the eigenvalues of a matrix using `eig`. To find the largest eigenvalue use `eigs(A,1)`.

Stabilisation

Suppose the pair (A, B) is controllable and consider the system

$$x_{k+1} = Ax_k + Bu_k.$$

Problem: Find a controller $u_k = \kappa(x_k)$ so that for the controlled system

$$x_{k+1} = Ax_k + B\kappa(x_k)$$

the origin is an *asymptotically stable* equilibrium point.

Hint: Choose $\kappa(x) = Kx$ and find K so that $A + BK$ has all its eigenvalues inside the unit circle.

Such a K always exists because the system is controllable and we may choose K so as to place the eigenvalues of $A + BK$ at any desired points in the unit circle.

Stabilisation

We may choose K so that $A + BK$ has eigenvalues s_1, \dots, s_{n_x} using Ackerman's formula. In MATLAB we may use the function `place`.

But, where should we place the poles?

What about performance?

The linear quadratic regulator (LQR) gives an answer to these questions...

IV. Linear Quadratic Regulator

Motivation

We need to stabilise the system

$$x_{k+1} = Ax_k + Bu_k,$$

by a linear controller

$$u_k = Kx_k$$

so that the closed-loop system's response minimises a certain **cost index**.
We assume that x_k is exactly known at time k .

Before we proceed...

Consider the optimisation problem

$$\begin{aligned} & \underset{x \in \mathbb{R}^n}{\text{minimise}} && f(x) \\ & \text{s.t.} && g_j(x) = 0, \end{aligned}$$

Suppose the problem is *convex* and *feasible*. Let x^* be an optimiser and define the *Lagrangian* function

$$\mathcal{L}(x, \lambda) = f(x) + \sum_j \lambda_j g_j(x)$$

Then there is a vector λ^* so that

$$\nabla \mathcal{L}(x^*, \lambda^*) = 0$$

We call the problem convex if f is convex and the set $\{x : g(x) = 0\}$ is convex.

Finite horizon LQR

We need to determine a finite seq. of inputs $\pi = (u_0, u_1, \dots, u_{N-1})$ which solves the optimisation problem

$$\text{minimise } J(\pi; x) := \frac{1}{2}x_N^\top Q_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k$$

subject to

$$x_{k+1} = Ax_k + Bu_k, \text{ for } k = 0, \dots, N-1$$

$$x_0 = x$$

N is called the *horizon* of the problem.

Finite horizon LQR

Some remarks:

$$\begin{aligned} \text{minimise } J(\pi; x) &:= \frac{1}{2}x_N^\top Q_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\ x_{k+1} &= Ax_k + Bu_k, \text{ and } x_0 = x \end{aligned}$$

- ▶ We take $Q = Q^\top \succcurlyeq 0$, $R = R^\top \succ 0$ and
- ▶ the problem is convex quadratic with equality constraints,
- ▶ with decision variables u_0, u_1, \dots, u_{N-1} and x_1, \dots, x_N and
- ▶ it has a unique minimiser.

The notation $Q \succcurlyeq 0$ means that Q is pos. semidefinite. $R \succ 0$ means R is pos. def.

Finite horizon LQR: solution

The Lagrangian is

$$\mathcal{L} = J + \sum \lambda_{k+1}^\top (Ax_k + Bu_k - x_{k+1}),$$

and by setting the gradient wrt u_k ($k = 0, \dots, N-1$) equal to 0 we obtain

$$\nabla_{u_k} \mathcal{L} = Ru_k + B^\top \lambda_{k+1} = 0 \Rightarrow u_k = -R^{-1} B^\top \lambda_{k+1}$$

Similarly wrt x_k ($k = 1, \dots, N-1$) we obtain

$$\begin{aligned} \nabla_{x_k} \mathcal{L} &= Qx_k + A^\top \lambda_{k+1} - \lambda_k = 0 \\ \Rightarrow \lambda_k &= Qx_k + A^\top \lambda_{k+1} \end{aligned}$$

Taking the gradient wrt x_N we have

$$\nabla_{x_N} \mathcal{L} = Q_f x_N - \lambda_N = 0 \Rightarrow \lambda_N = Q_f x_N$$

Finite horizon LQR: solution

To sum up

$$x_{k+1} = Ax_k + Bu_k$$

$$\lambda_k = Qx_k + A^\top \lambda_{k+1}$$

$$\lambda_N = Q_f x_N$$

$$u_k = -R^{-1}B^\top \lambda_{k+1}$$

$$x_0 = x$$

Finite horizon LQR: solution

Therefore,

$$\begin{aligned}\lambda_N &= Q_f(Ax_{N-1} + Bu_{N-1}) \\ &= Q_f(Ax_{N-1} - BR^{-1}B^\top \lambda_N) \\ \Rightarrow \lambda_N &= (I + Q_fBR^{-1}B^\top)^{-1}Q_fA \cdot x_{N-1}\end{aligned}$$

and

$$\begin{aligned}\lambda_{N-1} &= [A^\top(I + Q_fBR^{-1}B^\top)^{-1}Q_fA + Q] \cdot x_{N-1} \\ &= P_{N-1} \cdot x_{N-1}\end{aligned}$$

Finite horizon LQR: solution

We may recursively show that

$$\lambda_N = Q_f x_N = P_N x_N$$

and

$$\lambda_k = P_k x_k,$$

where P_k satisfies the recursion

$$P_k = Q + A^\top (I + P_{k+1} B R^{-1} B^\top)^{-1} P_{k+1} A.$$

Finite horizon LQR: solution

The finite-horizon LQR control law has the form

$$u_k = K_k x_k,$$

where

$$K_k = -R^{-1}B^\top(I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A,$$

and

$$\begin{aligned}P_k &= Q + A^\top(I + P_{k+1}BR^{-1}B^\top)^{-1}P_{k+1}A, \\P_N &= Q_f.\end{aligned}$$

Infinite horizon LQR

We need to determine a sequence of inputs $\pi = (u_0, u_1, \dots)$ which solves the optimisation problem

$$\text{minimise } J(\pi; x) := \frac{1}{2} \sum_{k=0}^{\infty} x_k^{\top} Q x_k + u_k^{\top} R u_k$$

subject to

$$x_{k+1} = Ax_k + Bu_k, \text{ for } k \in \mathbb{N}$$

$$x_0 = x$$

Assume that (A, B) is controllable so that the problem is feasible. Why?

Infinite horizon LQR

Fact. Assume that (A, B) and (A^\top, \sqrt{Q}^\top) are controllable. Then, there is a unique symmetric pos. def. matrix P satisfying

$$\begin{aligned} P &= Q + A^\top(I + PBR^{-1}B^\top)^{-1}PA \\ &= Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA, \end{aligned}$$

and the optimal sequence of inputs is given by

$$u_k = Kx_k,$$

with $K = -(R + B^\top PB)^{-1}B^\top PA$, and K is an asym. stabilising gain. Additionally, the optimal cost will be $J^\star(x_0) = \frac{1}{2}x_0^\top Px_0$.

Infinite horizon LQR

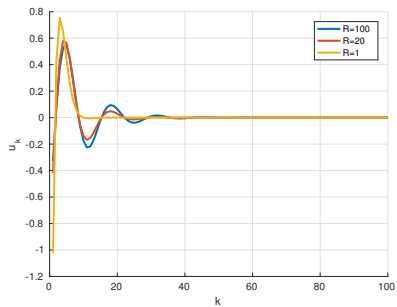
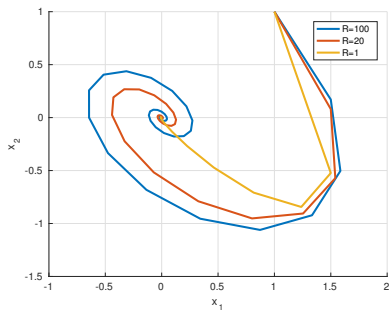
The gain K computed by the infinite horizon LQR deems $A + BK$ has all its eigenvalues strictly inside the unit circle.

In MATLAB one may compute the LQR solution using `lqr` or `dlqr`, but be wary as they return $-K$ instead of K (they make $A - BK$ stable).

Choice of Q and R

- ▶ One may choose Q and R to be diagonal matrices
- ▶ The larger Q is, the more responsive and aggressive the controlled system becomes,
- ▶ The higher R is, the smoother the response will be.

Choice of Q and R



V. Reference tracking

Problem Statement

So far we have been concerned with the stabilisation of the system towards an equilibrium point (x^e, u^e) — or $(0, 0)$ for the system with state Δx and input Δu .

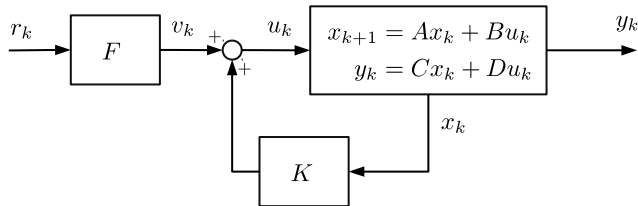
Typically in practice we need to be able to steer the system output

$$y_k = Cx_k + Du_k,$$

to a given reference r , that is $y_k \rightarrow r$ as $k \rightarrow \infty$, for any externally provided r .

Reference tracking v1

Choose $u_k = Kx_k + Fr_k$ and choose K to be a stabilising gain for (A, B)



We assume that (A, B) is controllable.

Reference tracking v1

We now have a system with input r_k , state x_k and output y_k described by*

$$\begin{aligned}x_{k+1} &= (A + BK)x_k + BFr_k \\ y_k &= (C + DK)x_k + DFr_k.\end{aligned}$$

Let (x^e, r^e) be an equilibrium point of the above system. Then

$$\begin{aligned}(A + BK - I)x^e + BFr^e &= 0 \\ (C + DK)x^e + DFr^e &= y^e.\end{aligned}$$

and observe what $A + BK - I$ is invertible (why?), so

$$x^e = -(A + BK - I)^{-1}BFr^e = EFr^e,$$

therefore

$$((C + DK)E + D)Fr^e = y^e.$$

Reference tracking v1

We require that the reference-to-output static gain is equal to I , that is

$$((C + DK)E + D)F = I,$$

Notice that $(C + DK)E + D$ is not always a square matrix.

In general, we are not always able to find such an F !

One possible solution (expensive):

$$F = ((C + DK)E + D)^+,$$

Another possible solution:

$$F \in \operatorname{argmin} \|((C + DK)E + D)F - I\|^2$$

Example

```
A = [1 1; 0 0.5]; B = [0;1]; C = [1 0.1]; D = 0.1;
nx = length(A); nu = size(B,2); ny = size(C,1);

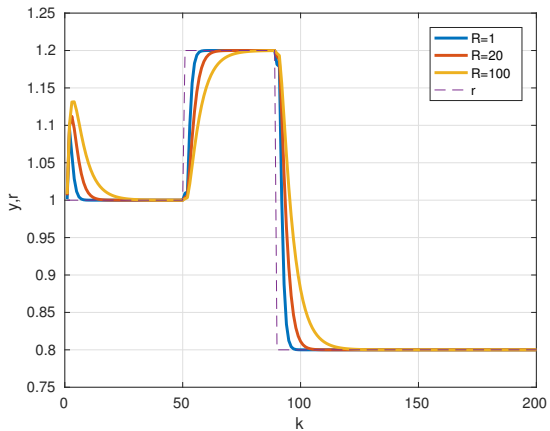
if rank(ctrb(A,B),1e-6)~=nx,
    error('System not controllable');
end

Q = eye(nx); R = 2*eye(nu);
K = -dlqr(A,B,Q,R,0);

assert( all(abs(eig(A+B*K)) <= 1-1e-7), ...
    'K is not stabilising');

E = -(A+B*K-eye(nx))\B;
X = (C+D*K)*E + D;
assert(rank(X, 1e-6) == ny, 'X not full rank');
F = X\eye(ny); % OR: F = pinv(X);
assert( norm((X*F - eye(ny)), Inf) < 1e-12 );
```

Example



Reference tracking v2

Let us consider a slightly different parametrisation for u_k

$$u_k = u^e + K(x_k - x^e),$$

where x^e and u^e are computed so that

$$\underbrace{\begin{bmatrix} A - I & B \\ C & D \end{bmatrix}}_W \begin{bmatrix} x^e \\ u^e \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix}$$

where W is not necessarily square. Now, find G so that

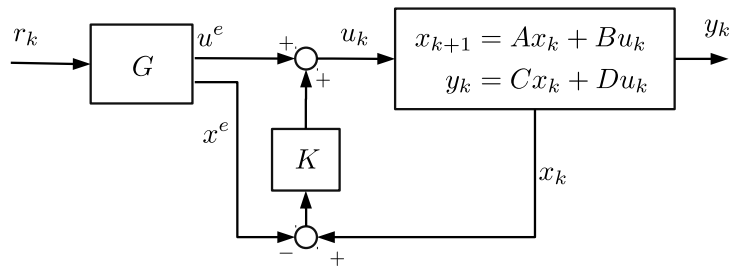
$$WG = \begin{bmatrix} 0_{n_x \times n_y} \\ I_{n_y} \end{bmatrix}$$

Then

$$\begin{bmatrix} x^e \\ u^e \end{bmatrix} = Gr.$$

This means that (x^e, u^e) is an equilibrium point of the system dynamics: $x^e = Ax^e + Bu^e$ and the corresponding output is $r = Cx^e + Du^e$.

Reference tracking v2



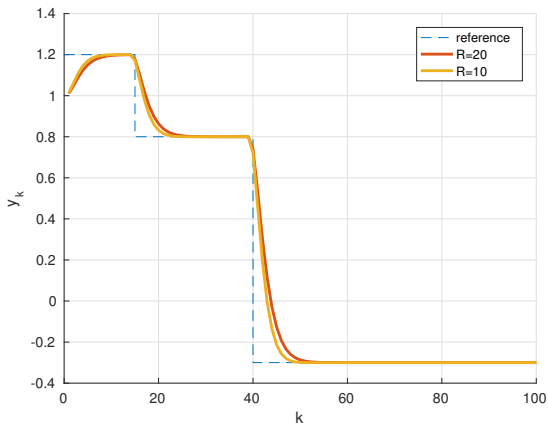
Example

```
A = [1 1; 0 0.5];  
B = [0;1];  
C = [1 0.1];  
D = 0.1;  
  
nx = length(A);           % nx = 2  
nu = size(B,2);           % nu = 1  
ny = size(C,1);           % ny = 1  
  
Q = eye(nx);  R = 20*eye(nu);  
K = -dlqr(A,B,Q,R,0);  
  
W = [A-eye(nx), B;  
      C,          D];  
G = W\[zeros(nx, ny); eye(ny)];
```

Example

```
x = [1;0];
X=x;
Y=[];
for k=1:100,
    if (k<15),
        r = 1.2;
    elseif (k>=15 && k<40),
        r = 0.8;
    else
        r = -0.3;
    end
    xue = G*r;
    u = xue(nx+1:nx+nu) + K*(x-xue(1:nx));
    x = A*x + B*u;
    X = [X;x];
    Y = [Y; C*x + D*u];
end
plot(Y','linewidth',2);
```

Example



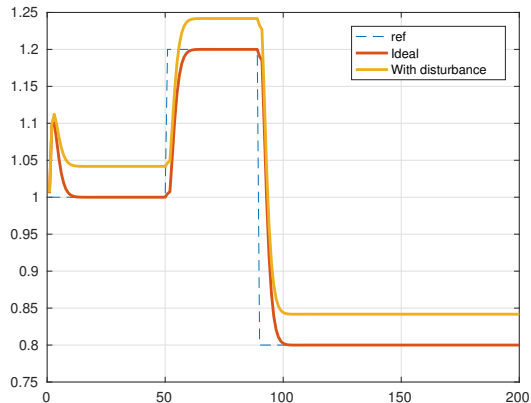
Reference tracking with disturbances

Problems. Is the controlled system really offset free? How can we guarantee that $e_k = y_k - r_k$ indeed converges to 0?

What if a constant disturbance d_k acts on the system as follows?

$$x_{k+1} = Ax_k + Bu_k + d_k$$

Reference tracking with disturbances



Here a disturbance as small as $d = 0.01$ made the system equilibrate away from the set-point.

Integral action

Solution. integral action! We introduce the error integral dynamics:

$$z_{k+1} = z_k + r_k - y_k$$

and use the control law

$$u_k = u^e + K_x(x_k - x^e) + K_z z_k.$$

In MATLAB one may compute the gains K_x and K_z using `lqi`.

The MATLAB function `lqi` returns $-K_x$ and $-K_z$. Read the documentation. The MATLAB function `ss` will also be needed.

Integral action stability

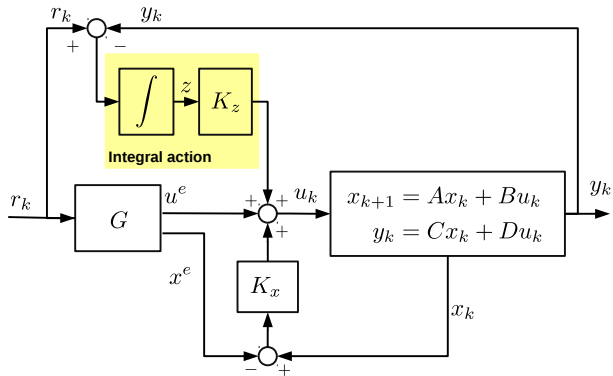
As an exercise, show that if $\begin{bmatrix} K_x & K_z \end{bmatrix}$ is a stabilising gain for the pair

$$\left(\begin{bmatrix} A & 0 \\ C & I \end{bmatrix}, \begin{bmatrix} B \\ 0 \end{bmatrix} \right),$$

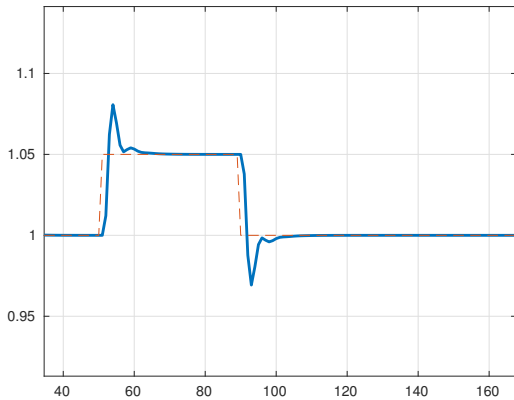
and the reference signal is constant, $r_k = r$, then $y_k \rightarrow r$ for any constant disturbance $d_k = d$.

The LQI returns K_x and K_z which satisfy this condition and, additionally, minimise an infinite-horizon cost index.

Integral action



Integral action in action



VI. Linear State Observers

Problem Statement

So far we have assumed that x_k is known at time k (measured) and free of noise. This is usually unrealistic. Instead, we obtain a measurement y_k

$$y_k = Cx_k + Du_k.$$

We then need a way to produce estimates \hat{x}_k of the current state using information that can be observed: current and past inputs and outputs.

Observability

Consider the system

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k.$$

We say that it is *observable* if there is $n \in \mathbb{N}$ so that for any x_0 and any finite sequence of inputs $\{u_k\}_{k=0}^n$, x_0 can be uniquely determined using $\{u_k\}_{k=0}^{n-1}$ and $\{y_k\}_{k=0}^n$.

Observability condition

The pair (C, A) is observable if and only if (A^\top, C^\top) is controllable, that is

$$\text{rank} \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n_x-1} \end{bmatrix} = n_x$$

In MATLAB one may use the commands `rank` and `obsv`. Use the SVD decomposition of the observability matrix to tell whether it is near-unobservable (In MATLAB use `svd`).

State observers

A linear state observer is a dynamical system

$$\begin{aligned}\hat{x}_{k+1} &= A\hat{x}_k + L(\hat{y}_k - y_k) + Bu_k \\ \hat{y}_k &= C\hat{x}_k + Du_k.\end{aligned}$$

Define the state estimation error $e_k = \hat{x}_k - x_k$. Then

$$e_{k+1} = (A + LC)e_k.$$

The error dynamics is asymptotically stable ($e_k \rightarrow 0$ as $k \rightarrow \infty$) provided that all eigenvalues of $A + LC$ are inside the unit circle.

State observer design

Straightforward solution: by pole placement.

But where should we place the poles?

Rule of thumb: the poles of $A + LC$ should be about 10 times smaller than the poles of the controlled system.

One may use MATLAB's `place` to construct L .

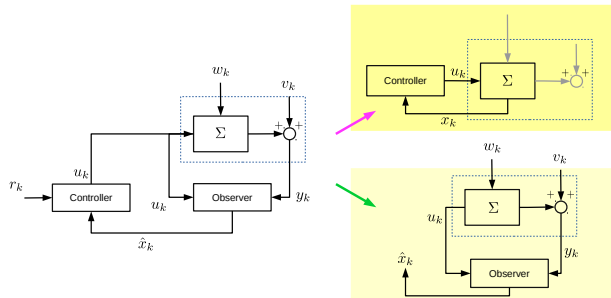
Separation principle

We may always design the controller and the observer separately and then combine them. For example, using integral action:

$$\begin{aligned}z_{k+1} &= z_k + r_k - y_k, \\u_k &= u^e + K_x(\hat{x}_k - x^e) + K_z z_k, \\\hat{x}_{k+1} &= A\hat{x}_k + L(\underbrace{C\hat{x}_k + Du_k}_{\hat{y}_k} - y_k) + Bu_k,\end{aligned}$$

where the first two equations correspond to the controller which computes u_k based on the state estimate \hat{x}_k as the state x_k is not directly available. The last equation is used to update the estimate \hat{x}_{k+1} .

Separation principle



Estimation under uncertainty

Consider the system

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + Gw_k \\y_k &= Cx_k + Du_k + Hw_k + v_k,\end{aligned}$$

where w_k and v_k are *zero-mean* random variables with

$$\mathbb{E}[w_k w_k^\top] = \Sigma_w$$

$$\mathbb{E}[v_k v_k^\top] = \Sigma_v,$$

$$\mathbb{E}[w_k v_k^\top] = 0,$$

with $\mathbb{E}[w_k w_j^\top] = 0$ and $\mathbb{E}[v_k v_j^\top] = 0$ for $k \neq j$.

Minimum energy estimation

We need to produce an estimate \hat{x}_k out of observations $\{y_j, u_j\}_{j \leq k}$ so as to minimise

$$J_{\text{mee}} = \sum_{t=-\infty}^k v_t^\top Q v_t + w_t^\top R w_t,$$

where

- ▶ large Q : we trust our measurements
- ▶ large R : we trust our system model
- ▶ Choose: $Q = \Sigma_v^{-1}$, $R = \Sigma_w^{-1}$

This is equivalent to minimising the steady-state covariance of the state estimation error

$$P = \lim_k \mathbb{E}[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^\top].$$

Steady-state Kalman Filter

The best estimator is then known to be

$$\hat{x}_{k+1} = A\hat{x}_k + L(C\hat{x}_k + Du_k - y_k) + Bu_k,$$

where L is given by

$$L = -(APC^\top + GQH^\top)(CPC^\top + R + HQH^\top)^{-1},$$

where P solves an algebraic Ricatti equation. In MATLAB, L and P are computed by `kalman` (see also `lqg`).

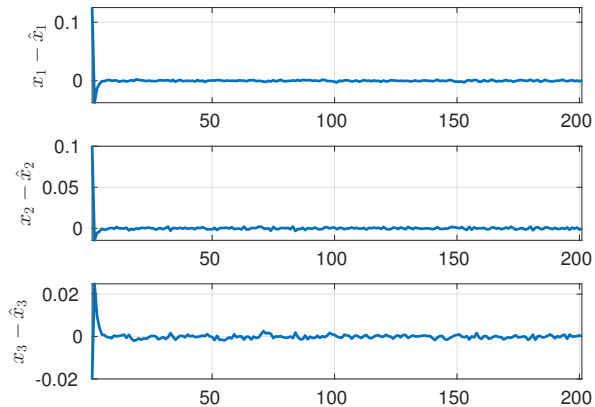
Example

System dynamics:

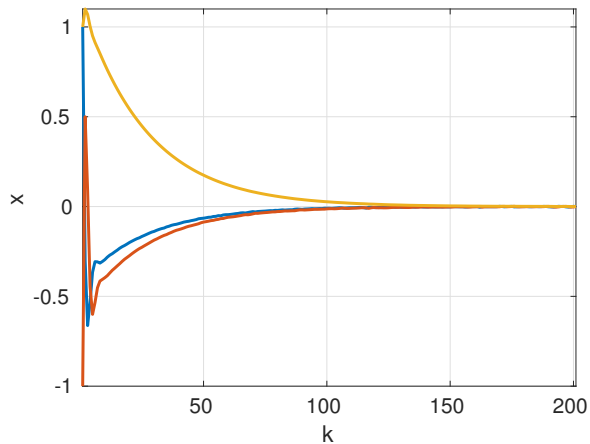
$$x_{k+1} = \begin{bmatrix} 0.5 & -0.9 & -0.1 \\ 0.9 & 0.5 & 0.1 \\ 0.1 & 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u_k + w_k$$
$$y_k = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -0.5 \end{bmatrix} x_k + v_k$$

The system is controllable and observable. We choose the LQR parameters $Q_{\text{lqr}} = 5I$, $R_{\text{lqr}} = 4$, and the Kalman filter parameters $Q_{\text{kal}} = 10 \cdot I$, $R_{\text{kal}} = 0.01 \cdot I$.

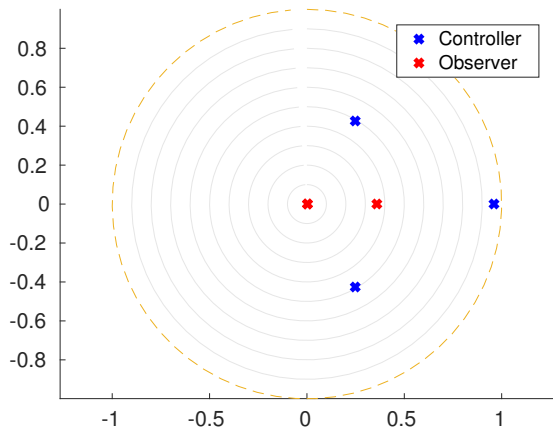
Example — estimation error



Example — response



Example — poles



Measurement bias

What will happen if our measurements have a constant bias?

$$\begin{aligned}x_{k+1} &= Ax_k + Bu_k + w_k \\ y_k &= Cx_k + Du_k + v_k + d_k^y,\end{aligned}$$

where $v_k \sim \mathcal{N}(0, \Sigma_v)$, $w_k \sim \mathcal{N}(0, \Sigma_w)$ and d_k^y is **constant**.

We will see that by applying a standard KF, $\hat{x}_k - x_k$ does not converge to a zero-mean distribution.

Measurement bias — Example

```
% System data
A = [1 1.2; 0.1 0.5];
B = [0;1];
C = [1 0.1];
D = 0.1;

nx = size(A,1); nu = size(B,2); ny = size(C,1);

% LQR design
Q = eye(2);
R = 80;
K = -dlqr(A, B, Q, R, 0);

dy = 1.0; % measurement bias
```

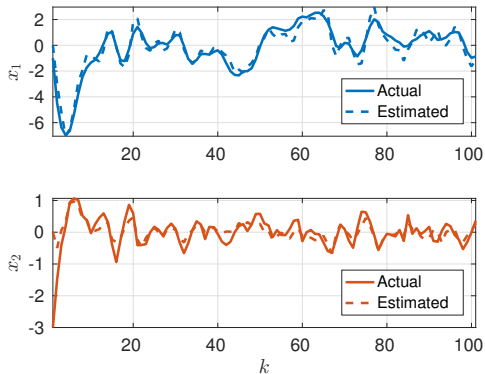
Measurement bias — Example

```
% Kalman filter design
Qo = diag([20, 20]); Ro = 100;
ss_kalman = ss(A, [B eye(2)], C, [D 0 0], -1);
[~, L, P] = kalman(ss_kalman, Qo, Ro, 0);

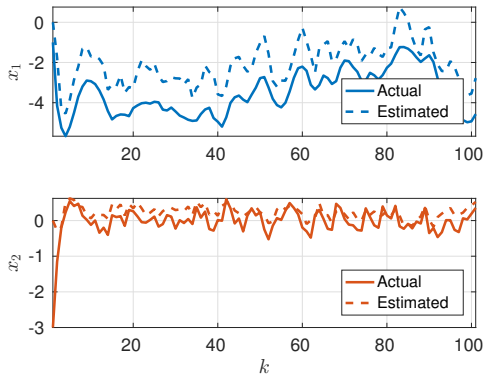
x  = [-1;10]; % initial values (x_0, z_0)
x_ = [-1;10]; % initial state estimate

T = 200; % simulation time
X = zeros(2,T); X(:,1) = x; % cache of states
X_ = zeros(2,T); X_(:,1) = x_; % cache of estimates
for k=1:T,
    x = A * x + B * K * x_ + randn * 0.02;
    y = C * x + D * K * x_ + randn * 0.05 + dy ;
    x_ = (A - L*C)*x_ + (B - L*D) * K * x_ + L * y;
    X(:,k+1) = x_; X(:,k+1) = x;
end
```

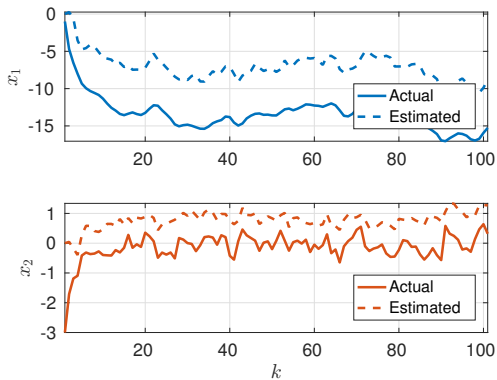
Measurement bias — State estimation, $dy=0$



Measurement bias — State estimation, $dy=1$



Measurement bias — State estimation, $dy=5$



State estimation & Measurement bias

Some remarks:

- ▶ If $d^y = 0$ everything works like clockwork
- ▶ A KF fails to reconstruct the system of the state
- ▶ A biased output tracks the desired disturbance

Measurement bias — a remedy

One possible remedy is to perfectly calibrate our sensors. But often this is not enough, let alone, they often wear out with time and use.

The system dynamics (omitting all noise terms) is written as

$$\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k$$
$$y_k = \begin{bmatrix} C & I \end{bmatrix} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + D u_k$$

Exercise 1: Show that the above system is not controllable. Exercise 2: Construct a matrix $A \in \mathbb{R}^{2 \times 2}$ and a $C \in \mathbb{R}^{1 \times 2}$ so that (C, A) is observable, but the above system is unobservable.

Measurement bias — a remedy

By defining $\tilde{x}_k = (x_k, d_k)^\top$ we may rewrite the system as

$$\begin{aligned}\tilde{x}_{k+1} &= \tilde{A}\tilde{x}_k + \tilde{B}u_k \\ y_k &= \tilde{C}\tilde{x}_k + Du_k,\end{aligned}$$

and if (\tilde{C}, \tilde{A}) is observable (see slides 60 and 61), we may design a state observer to estimate \tilde{x}_k , so the observer will produce two estimates \hat{x}_k and \hat{d}_k and as $k \rightarrow \infty$, $\hat{d}_k - d_k \rightarrow 0$.

Measurement bias — a remedy

```
% Data of the augmented system
```

```
A_ = blkdiag(A, 1);
```

```
C_ = [C 1];
```

```
B_ = [B;0];
```

```
% Design augmented state observer
```

```
sys_ = ss(A_, [B_ eye(3)], C_, [D zeros(1,3)], -1);
```

```
[~, L_] = kalman(sys_ Qo, Ro, 0);
```

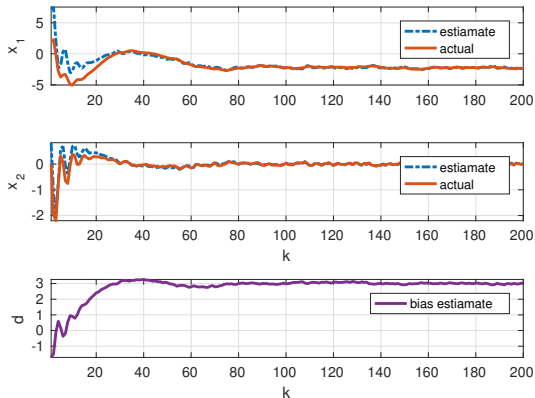
```
% State estimation
```

```
x_ = (A_ - L*C_)*x_ + (B_ - L*D_) * u + L * y;
```

```
xest = x_(1:2);
```

```
dest = x_(3);
```

Measurement bias — a remedy



Here we used $d^y = 3$. Indeed the observer produces an estimate \hat{d}_k which converges to d^y (plus-minus zero-mean noise).

Measurement bias — conclusions

Some notes:

- ▶ The variable $y_k - \hat{d}_k$ is an estimate of the system's unbiased output
- ▶ The controller's goal should be to drive $y_k - \hat{d}_k$ to r , so asymptotically $Cx_k + Du_k \rightarrow r$.

Extensions

There exist more elaborate state estimators such as

- ▶ Linear Quadratic Gaussian (LQG): LQR and KF
- ▶ The Extended Kalman Filter
- ▶ The Unscented Kalman Filter
- ▶ Particle filters

and many another.

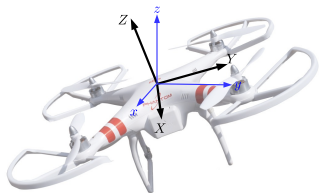
Designer's arsenal

We have the following tools in our toolbox:

- ▶ Rejection of measurement bias by augmenting the state space system with the disturbance
- ▶ Integral action to attenuate the effect of constant disturbances acting on the system state
- ▶ Use none, both or either depending on the problem, but choose the simplest formulation that leads to good closed-loop behaviour. In other words, verify your assumptions with simple experiments.

VII. Quaternions and Rotations

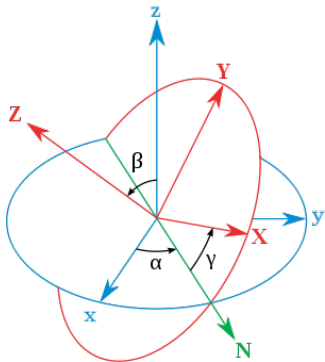
Reference frame



The orientation of a solid body in space can only be defined with respect to a reference frame.

Here we use a **body-fixed frame**, that is, roughly speaking, the orientation is described with what the body itself defines as upright and forward.

Euler angles



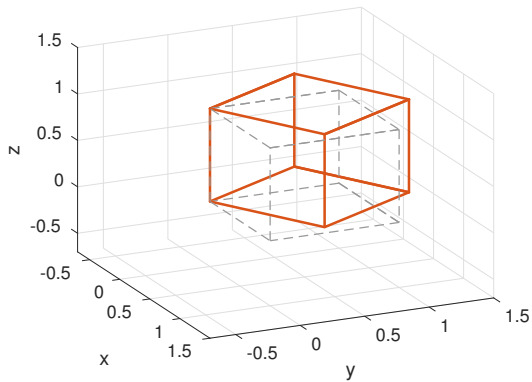
Euler angles: (α, β, γ)

Reference frame: (x, y, z)

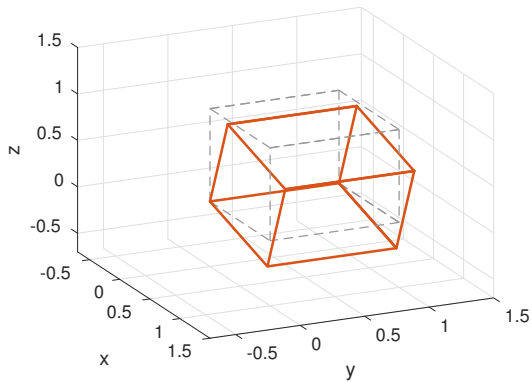
Rotated frame: (X, Y, Z)

Important: the order of the three elementary rotations matters! The standard convention is: yaw, pitch, roll.

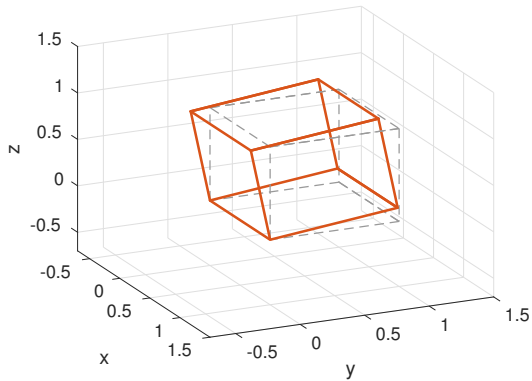
Yaw (heading)



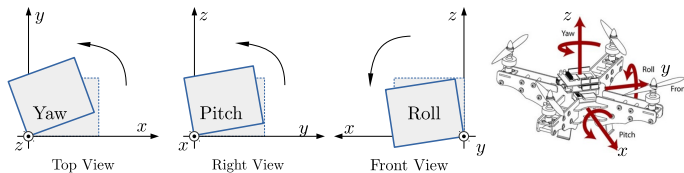
Pitch (elevation)



Roll (bank)



Positive rotation convention



Use the right hand rule. Mnemonic trick: In alphabetical order: $x \rightarrow y \rightarrow z \rightarrow x \rightarrow \dots$. Positive yaw: face left; Positive pitch: tilt upward as when an aeroplane taking off; Positive roll: roll to the right side.

Quaternions

A quaternion is a representation of a rotation; it is a 4-dimensional vector

$$q = (q_0, q_1, q_2, q_3).$$

Often, it is represented as a hyper-complex number

$$q = q_0 + iq_1 + jq_2 + kq_3,$$

where $i^2 = j^2 = k^2 = ijk = -1$. We call q_0 its **scalar part** and (q_1, q_2, q_3) its **vector part**.

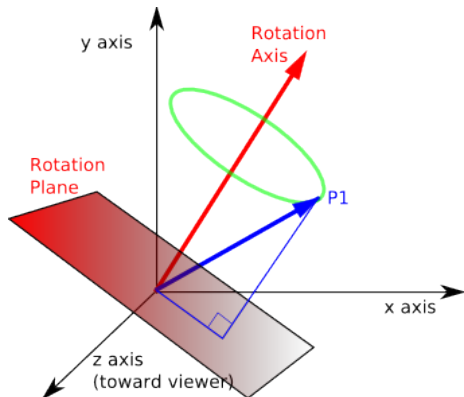
The space of quaternions is denoted by \mathbb{H} .

Quaternion representations of rotations

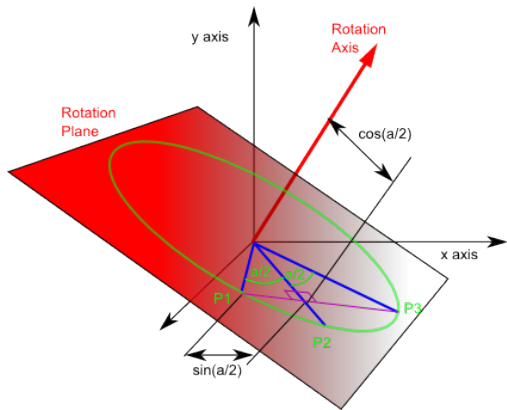
A rotation/orientation can be described by a quaternion by determining (i) a direction of rotation $u \in \mathbb{R}^3$ and (ii) the angle of rotation about u . The corresponding quaternion is

$$q = \cos \frac{\alpha}{2} + (iu_x + ju_y + ku_z) \sin \frac{\alpha}{2}.$$

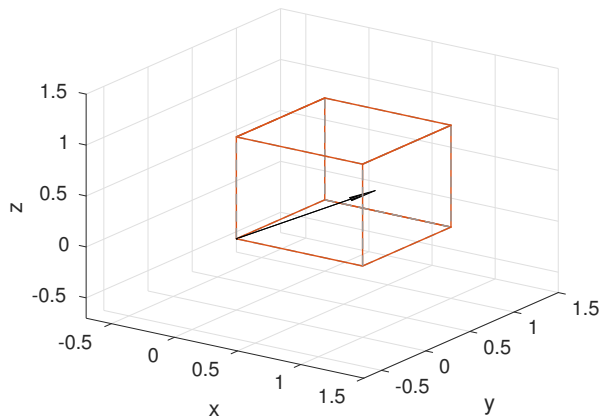
Quaternions



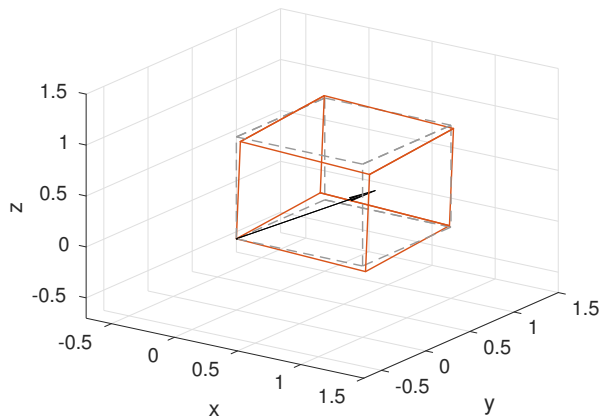
Quaternions



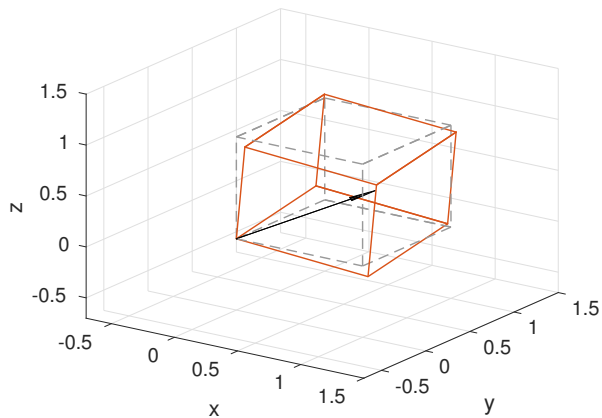
Rotations using quaternions



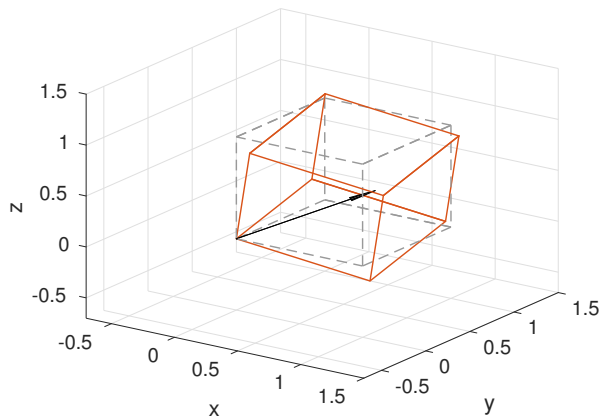
Rotations using quaternions



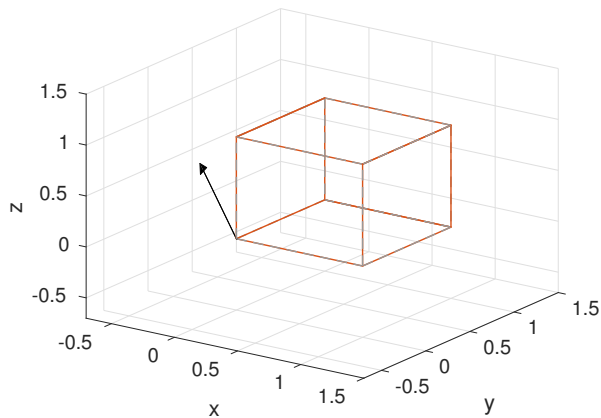
Rotations using quaternions



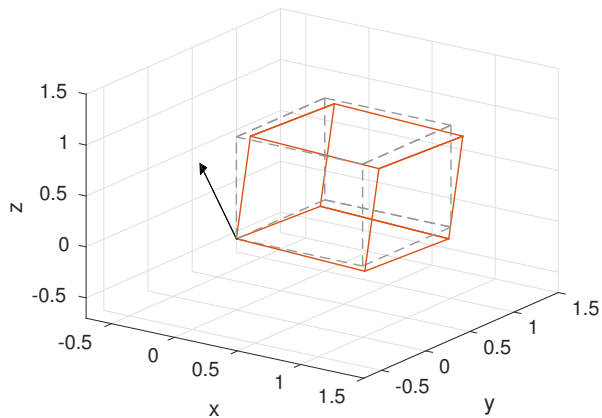
Rotations using quaternions



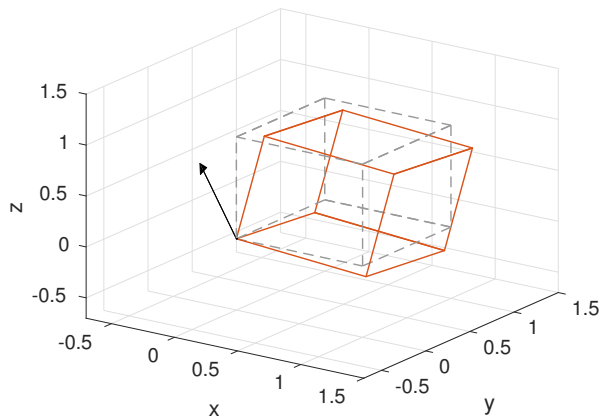
Rotations using quaternions



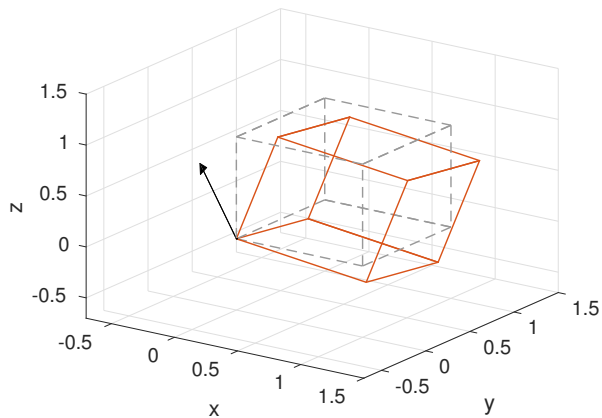
Rotations using quaternions



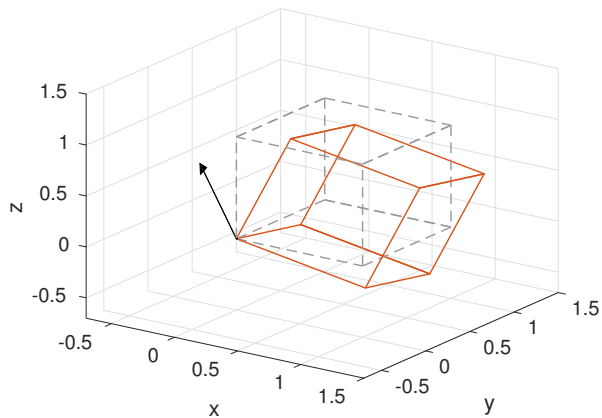
Rotations using quaternions



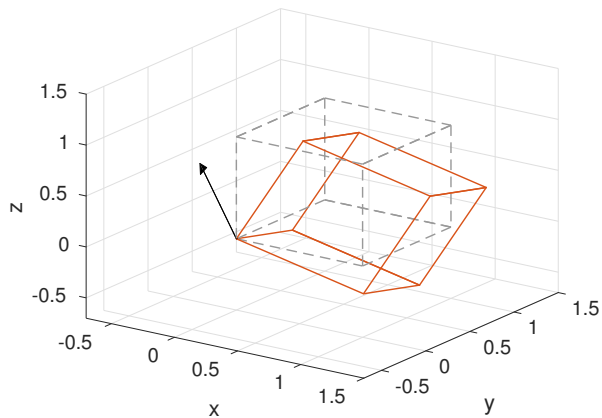
Rotations using quaternions



Rotations using quaternions



Rotations using quaternions



Quaternions: some simple calculus

Let $p = (p_0, p_1, p_2, p_3)$, $q = (q_0, q_1, q_2, q_3)$ and $\lambda \in \mathbb{R}$. Then:

- ▶ **Addition:** $p + q = (p_0 + q_0, p_1 + q_1, p_2 + q_2, p_3 + q_3)$
- ▶ **Multiplication:**
 - ✓ $ijk = -1$
 - ✓ $ij = k, jk = i, ki = j$
 - ✓ $ji = -k, kj = -i, ik = -j$
 - ✓ $p \cdot q = (p_0 + ip_1 + jp_2 + kp_3) \cdot (q_0 + iq_1 + jq_2 + kq_3) = \dots$
 - ✓ But, $p \cdot q$ is not the same as $q \cdot p$
 - ✓ Let $1_{\mathbb{H}} = (1, 0, 0, 0)$; then $p \cdot 1_{\mathbb{H}} = 1_{\mathbb{H}} \cdot p = p$.
- ▶ **Division:** $p/q = p \cdot q^{-1}$ (later)

Quaternions — some definitions

The **norm** of a quaternion $q \in \mathbb{H}$ is

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

Quaternions with $\|q\| = 1$ are called **unit quaternions** (\mathbb{H}_1), are used to represent rotations in \mathbb{R}^3 and have the representation

$$q = \cos \frac{\alpha}{2} + \sin \frac{\alpha}{2} (iu_x + ju_y + ku_z)$$

The *complex conjugate* of q is

$$q^* = (q_0, -q_1, -q_2, -q_3).$$

Hereafter, we will only be dealing with unit quaternions. In MATLAB the conjugate of a quaternion is computed by `quatconj`. The 2-norm of a quaternion is computed by `quatnorm`.

Quaternions — combined rotation

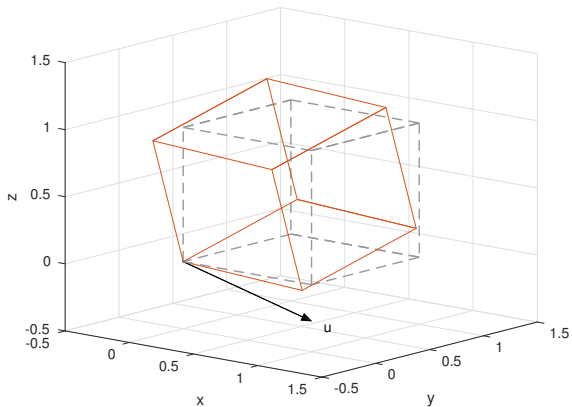
We define the (Hamilton) product of two quaternions $p, q \in \mathbb{H}$

$$p \otimes q \equiv p \cdot q = Q(p) \cdot q = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

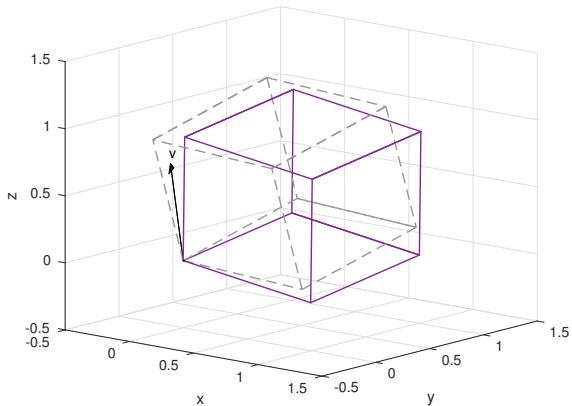
The operation \otimes is not commutative. The product $q \otimes p$ describes the combined rotation of p followed by q (in the absolute frame of reference).

Useful properties: $(p \otimes q)^* = q^* \otimes p^*$ and $(p^*)^* = p$. Therefore, $p \otimes q = (q^* \otimes p^*)^*$. Source: Fresk & Nikolakopoulos, 2013. In MATLAB use `quatmultiply`.

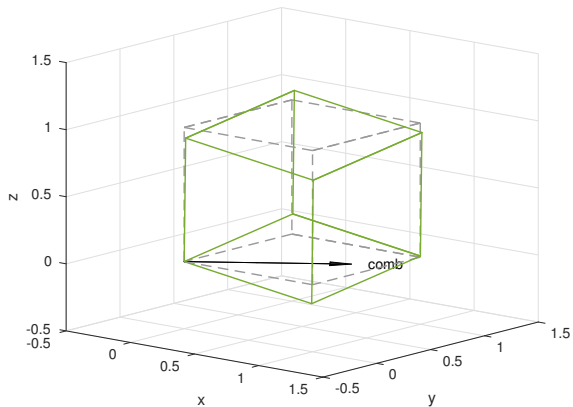
Quaternions — combined rotation



Quaternions — combined rotation



Quaternions — combined rotation



Hamilton product

We defined the Hamilton product $p \otimes q = Q(p) \cdot q$. We may represent this product in a dual way as

$$p \otimes q = \bar{Q}(q)p = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Interesting observations:

- ▶ $\bar{Q}(q^*) = \bar{Q}(q)^\top$.
- ▶ For $p, q \in \mathbb{H}_1$, $p \otimes q \in \mathbb{H}_1$.

Quaternions — inversion

Let p be a unit quaternion. Then

$$p \otimes p^* = p^* \otimes p = 1_{\mathbb{H}},$$

where $1_{\mathbb{H}} = (1, 0, 0, 0)$ is a no-rotation quaternion. In this sense, p^* is the *inverse* of p . In general

$$p^{-1} = \frac{p^*}{\|p\|}.$$

The inverse of a quaternion p^{-1} undoes the rotation p . It is

$$p \otimes p^{-1} = p^{-1} \otimes p = 1_{\mathbb{H}}.$$

Rotating objects using quaternions

Let an object be defined by some points $P_i \in \mathbb{R}^3$ and let $q \in \mathbb{H}_1$ be a unit quaternion describing a rotation. Then, the application of q on P_i rotates it to a new point \tilde{P}_i . This rotation is described by

$$\begin{aligned}\begin{bmatrix} 0 \\ \tilde{P}_i \end{bmatrix} &= p \otimes \begin{bmatrix} 0 \\ P_i \end{bmatrix} \otimes p^{-1} \\ &= p \otimes \begin{bmatrix} 0 \\ P_i \end{bmatrix} \otimes p^* \\ &= Q(p)\bar{Q}(p^*) \begin{bmatrix} 0 \\ P_i \end{bmatrix} \\ &= Q(p)\bar{Q}(p)^\top \begin{bmatrix} 0 \\ P_i \end{bmatrix}\end{aligned}$$

Rotations in MATLAB

```
% Define a box
A = [0 0 0]; B = [1 0 0]; C = [0 1 0]; D = [0 0 1];
E = [0 1 1]; F = [1 0 1]; G = [1 1 0]; H = [1 1 1];
P = [A;B;F;H;G;C;A;D;E;H;F;D;E;C;G;B];

% Define a quaternion
alpha = 0.1;
p = [cos(alpha/2), sin(alpha/2)*[2; 0.2; -0.6]'];
p = quatnormalize(p);

% Rotate the box P
N = size(P,1);
Q = quatmultiply(quatmultiply(p,[zeros(N,1) P]), ...
    quatconj(p));
Q = Q(:,2:4);

% Plot
plot3(Q(:,1),Q(:,2),Q(:,3),'LineWidth',2);
```

Rotations in MATLAB

Another alternative is to use the **direction cosines matrix**, a matrix $D \in \mathbb{R}^{3 \times 3}$ so that for $P \in \mathbb{R}^3$, $\tilde{P} = DP$ is the rotated point

```
% Using DCM  
Q = P*quat2dcm(p);
```

Moreover, MATLAB provides the function `quatrotate` where, however, one needs to provide p^* and not p

```
% Using quatrotate  
Q = quatrotate(quatconj(p), P);
```

Difference of quaternions — the wrong way

Fact. All quaternions αq , for $\alpha \neq 0$, represent the same exact rotation!

Let us define

$$d(p, q) = p - q.$$

Then

$$d(p, \alpha p) = (1 - \alpha)p,$$

but p and αp are equivalent rotations.

If $d(p, q)$ is a no-rotation quaternion then p and q are identical.

Difference of quaternions — the wrong way

Assume p and q are unit quaternions and take $p = -q$. Then

$$d(p, q) = 2p.$$

Although p and q correspond to the same rotation, there is no way to infer that by $d(p, q)$.

Difference of quaternions — the right way

A useful difference mapping is a $\delta : \mathbb{H} \times \mathbb{H} \rightarrow \mathbb{H}$ given by

$$\delta(p, q) = p \otimes q^*.$$

Take a unit quaternion $p \in \mathbb{H}$:

$$\delta(p, p) = p \otimes p^* = (1, 0, 0, 0).$$

Euler angles to quaternions

The rotation described by the Euler angles (ϕ, θ, ψ) is given by the quaternion

$$q = \begin{pmatrix} \cos \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \\ \sin \frac{\phi}{2} \cos \frac{\theta}{2} \cos \frac{\psi}{2} - \cos \frac{\phi}{2} \sin \frac{\theta}{2} \sin \frac{\psi}{2} \\ \cos \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} + \sin \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} \\ \cos \frac{\phi}{2} \cos \frac{\theta}{2} \sin \frac{\psi}{2} - \sin \frac{\phi}{2} \sin \frac{\theta}{2} \cos \frac{\psi}{2} \end{pmatrix}$$

Euler angles to quaternions

A quaterion $q = (q_0, q_1, q_2, q_3)$ corresponds to the following Euler angles

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \arctan \frac{2(q_0 q_1 + q_2 q_3)}{1 - 2(q_1^2 + q_2^2)} \\ \arcsin(2(q_0 q_2 - q_3 q_1)) \\ \arctan \frac{2(q_0 q_3 + q_1 q_2)}{1 - 2(q_2^2 + q_3^2)} \end{bmatrix}$$

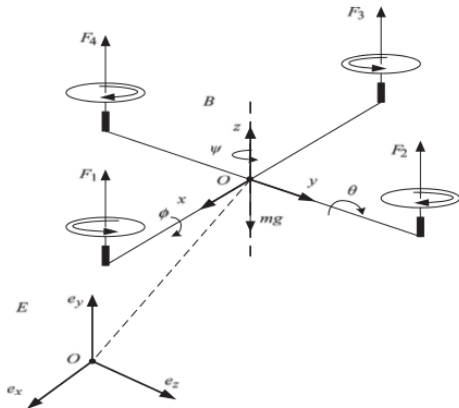
In MATLAB, do not use `atan` — use `atan2` instead.

Why quaternions?

- ▶ With Euler angles we always need to specify the sequence of rotations
- ▶ Gimbal lock — a singularity
- ▶ Rotations with simple algebraic operations
- ▶ Quaternions: numerically robust rotations.

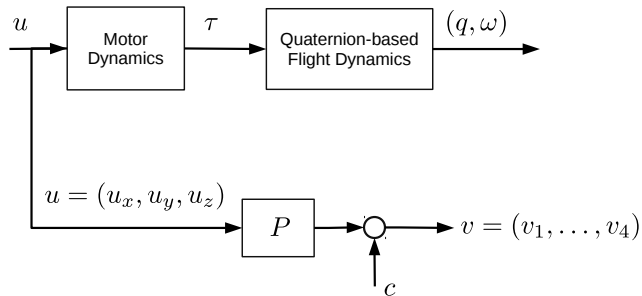
VIII. Attitude Dynamics & Control

Reference frames

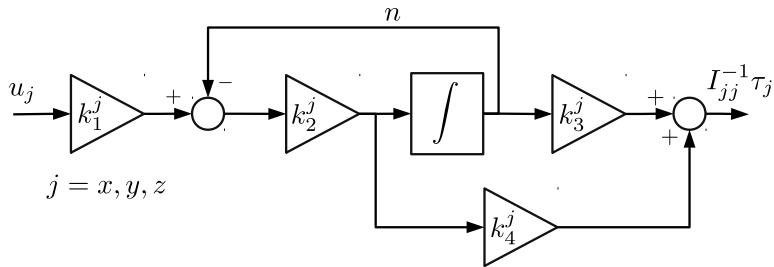


Source: Xiong and Zheng, 2013.

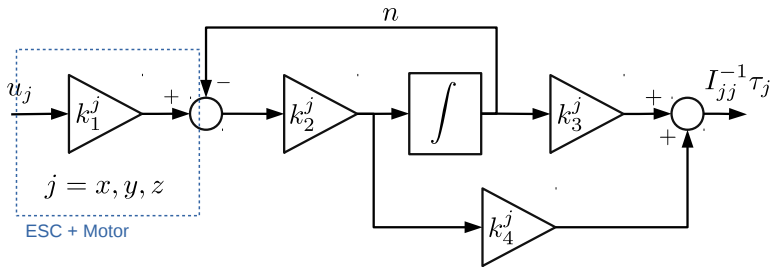
Quadcopter dynamics



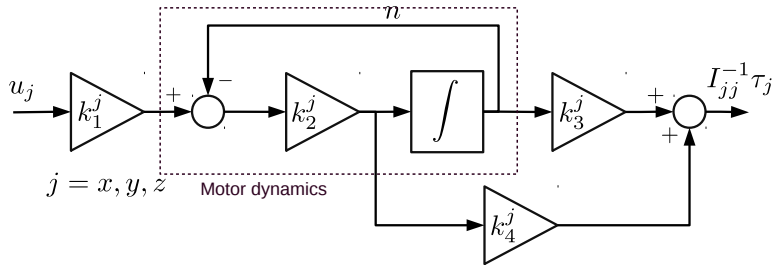
Motor & propeller dynamics



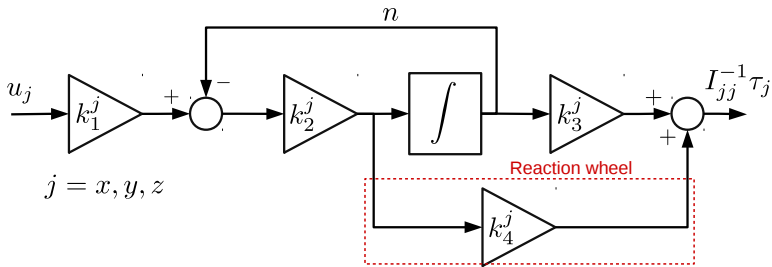
Motor & propeller dynamics



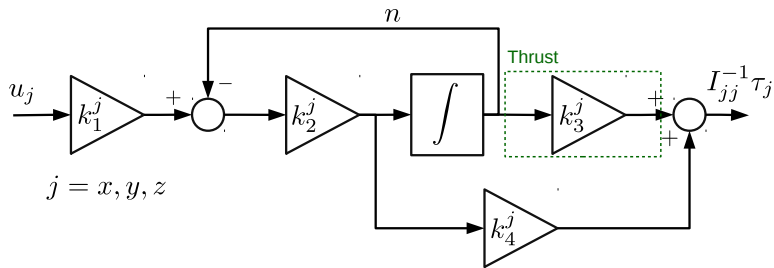
Motor & propeller dynamics



Motor & propeller dynamics



Motor & propeller dynamics



Parameters of motor & propeller dynamics

Parameters k_1 and k_2 are given by

$$k_1^{x,y,z} = \frac{V_{\max} - V_{\min}}{60} K_v$$
$$k_2^{x,y,z} = \frac{1}{\tau_m}$$

where

- ▶ V_{\max} is the max. voltage (around 11.1V),
- ▶ $V_{\min} = 10\%V_{\max}$,
- ▶ K_v is the *motor speed constant* (in *rpm/V*),
- ▶ τ_m is the time constant of the motors ($\tau_m = 35ms$)

Parameters of motor & propeller dynamics

Parameters k_3^x and k_3^y are given by (for $j = x, y$)

$$k_3^j = \frac{d}{dn} C_T \rho n^2 D^4 \Big|_{n_h} \frac{N_m L}{\sqrt{2} I_{jj}}$$

where

- ▶ C_T is the thrust coeff. the propellers,
- ▶ ρ is the density of the air,
- ▶ N_m is the number of motor,
- ▶ D is the diameter of each propeller,
- ▶ L is the arm length of the quadcopter,
- ▶ I_{prop} is the moment of inertia of each propeller,
- ▶ I_{xx}, I_{yy} are the m.o.i about the x and y axes and
- ▶ n_h is the hovering frequency of rotation (rps)
- ▶ $C_T \rho n^2 D^4$ is the thrust applied by each propeller.

Parameters of motor & propeller dynamics

Parameter k_3^z is given by

$$k_3^z = \frac{d}{dn} \frac{C_P \rho n^2 D^5}{2\pi} \bigg|_{n_h} \frac{N_m}{I_{zz}}$$

where

- ▶ C_P is the power coefficient of the propellers,
- ▶ I_{zz} is the m.o.i about the z axis and
- ▶ n_h is the hovering frequency of rotation (rps)

Parameters of motor & propeller dynamics

Parameters $k_4^{x,y}$ and k_4^z are given by

$$k_4^{x,y} = 0$$
$$k_4^z = 2\pi N_m \frac{I_{\text{prop}} + I_m}{I_{zz}},$$

where

- ▶ I_{prop} is the moment of inertia of each propeller
- ▶ I_m is each motor's moment of inertia

Flight dynamics

The attitude dynamics is described by

$$\begin{aligned}\dot{q} &= \frac{1}{2} \cdot q \otimes \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \\ \dot{\omega} &= I_{cm}^{-1}(\tau - \omega \times (I_{cm}\omega)),\end{aligned}$$

where I_{cm} is the inertia matrix

$$I_{cm} = \begin{bmatrix} I_{xx} & & \\ & I_{yy} & \\ & & I_{zz} \end{bmatrix}$$

Signals to motors

The signals v_1, \dots, v_4 to the four motors are computed in terms of the torque signals u_x, u_y, u_z and the a *common* signal c as follows

$$v_1 = c + u_x + u_y - u_z$$

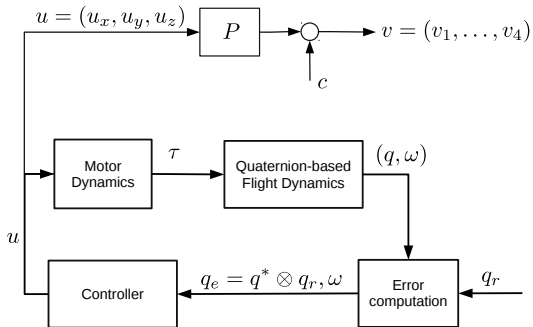
$$v_2 = c + u_x - u_y + u_z$$

$$v_3 = c - u_x + u_y + u_z$$

$$v_4 = c - u_x - u_y - u_z$$

where v_1, \dots, v_4 are *relative voltage* signals and will create the torques τ_x , τ_y and τ_z .

Closed-loop system



Details

- ▶ Consider only the dynamics of (q_1, q_2, q_3) , excluding q_0
- ▶ Use the fact that rotation quaternions are unit quaternions
- ▶ Write down the $(q_1, q_2, q_3), \omega$ -dynamics and linearise the system
- ▶ Verify that it is controllable
- ▶ Use the scalar part of q_e to “regularise” its vector part
- ▶ You will find bibliographic references at the end.

Details

Given that rotation quaternions $q \in \mathbb{H}$ are unitary

$$\|q\| = 1,$$

we have that

$$q_0 = \sqrt{1 - q_1^2 - q_2^2 - q_3^2}.$$

This means that the vector part of q suffices to determine q_0 (using the convention that $q_0 > 0$). We may use this fact to eliminate q_0 from the model.

Details

A rotation of -5° (about any axis) has the same effect as a rotation of 355° . In order to avoid going around the long way we need to canonicalise q_r as follows

$$\bar{q}_r = \begin{cases} q_r, & \text{if } q_{r0} > 0, \\ -q_r, & \text{otherwise} \end{cases}$$

Unless we need to perform a full rotation about any axis (which, unless we want to perform acrobatic manoeuvres, is the case only for the z axis).

Details

To compute the (hovering) equilibrium point $(q^e, \omega^e, n^e, u^e)$ we need to take into account the following

- ▶ The quadrotor should stay upright, that is $q^e = (1, 0, 0, 0)$
- ▶ It should not rotate or wobble, so $\omega^e = 0$
- ▶ It should lift its own weight, so the total thrust should be equal to its weight, that is

$$N_m C_T \rho n_h^2 D^4 = mg$$

- ▶ We then use n_h to compute k_3 at equilibrium,
- ▶ We linearise about $(n_x, n_y, n_z) = (0, 0, 0)$.

RC signals

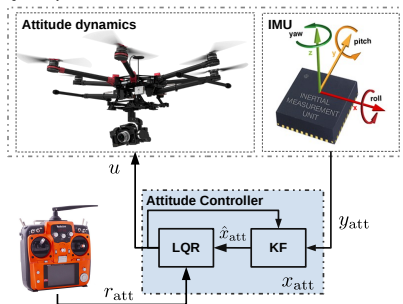


Note. The references from the RC are sent out as Euler angles. First, we need to compute the corresponding quaternion q_r .

IX. Implementation of Attitude Control System

The EAGLE MATLAB simulator

Quadcopter



This is the schematic overview of what we need to build.

The EAGLE MATLAB simulator

Main functions:

- ▶ `quat_params`: model parameters (adjust them!)
- ▶ `quat_dynamics`: continuous-time attitude dynamics
- ▶ `quat_linear_dyn`: linearised continuous-time dynamics
- ▶ `eagle_simulator`: attitude controller/observer designer (lots of options; read the documentation).

Helper functions:

- ▶ `deg2quat`: degrees to quaternion
- ▶ `print_mv`: generates C code for mat-vec operations. This function may assist in generating C code.

There is detailed documentation. Do study it.

Useful MATLAB functions (i)

- ▶ For C code generation we need:
 - ✓ `fopen`, `fclose`: open and close files
 - ✓ `fprintf`: write to file
- ▶ Make nice figures using:
 - ✓ `plot`, `plot3`: make figure; use the option `linewidth` to make lines adequately visible
 - ✓ `hold on`: make plots with multiple data
 - ✓ `xlabel`, `ylabel`: always add axis labels
 - ✓ `grid on`: show grid
- ▶ Control
 - ✓ `dlqr`, `kalman`: LQR and Kalman design
 - ✓ `eig`: eigenvalues of a matrix
 - ✓ `svd`: singular value decomposition

Useful MATLAB functions (ii)

► Quaternions:

- ✓ `quatnorm`: returns $\|p\|$
- ✓ `quatnormalize(p)`: returns $p/\|p\|$
- ✓ `deg2quat`: angle (in degrees) to normalised quaternion
- ✓ `quatmultiply(p,q)`: performs $p \otimes q$
- ✓ `quatdivide`: multiply by inverse quaternion
- ✓ `quatconj`: conjugate quaternion
- ✓ `quatinv`: inverse quaternion
- ✓ `quatrotate`: rotates a vector by (the conjugate of) a quaternion
- ✓ `atan2`: four quadrant arctangent

Step 1

- ▶ Peruse the course material
- ▶ Write down the system dynamics in state space form
- ▶ Determine the equilibrium points of the (CT) system
- ▶ Linearise the system & simulate
- ▶ Discretise & simulate again — compare

Step 2

- ▶ Decide the system structure (inputs, states, outputs)
- ▶ Is the system controllable? Are you sure?
- ▶ In MATLAB use `ctrb` and `rank`.

The SVD decomposition is also useful for finding the rank of a matrix and determining whether (A, B) is near-uncontrollable. In MATLAB use `svd` or `rank(·, tol)`.

Step 3

- ▶ Design a controller that steers the system state to a fixed equilibrium point (as in Section IV; no reference tracking)
- ▶ Choose some initial values for Q and R and find an LQR gain K
- ▶ Compute the eigenvalues of $A + BK$. Is it stable?
- ▶ Fine tune Q and R until you get satisfactory closed-loop behaviour

Step 4

- ▶ Implement an LQR controller with integral action
- ▶ Tweak the LQR weights Q and R — choose $Q = \text{blkdiag}(Q_x, Q_z)$
- ▶ Apply a constant disturbance — is it attenuated adequately?
- ▶ Find the poles of the overall system (with state (x_k, z_k))

Step 5

- ▶ Design a controller using LQR for reference tracking
- ▶ Design a steady-state Kalman filter
- ▶ Plot the poles of the observer and the controller
- ▶ Simulate the system and assess the closed-loop performance
- ▶ Assume we have inexact estimates of the model parameters and measurement noise

Make reasonable assumptions regarding the level of noise and the error in the determination of the system parameters.

Use experimental data. For example, how accurate are the IMU measurements? How accurate is the model?

Step 6

- ▶ Implement a C code generator that will produce an ANSI C implementation of your controller/observer
- ▶ Test the C code (on a computer)
- ▶ Do not hard-code the controller observer yourselves! You will certainly need to modify/re-tune your controller/observer and test it on the quadcopter several times. Make a good code generator.
- ▶ Test your implementation on the quadcopter:
 - ✓ Print the signals to the motors (with the motors disconnected)
 - ✓ Test without the propellers attached to the motors
 - ✓ Test on the gimbal vise
 - ✓ First test flight

Step 7

How to get rid of the drift...

- ▶ Calibrate the IMU (every time before taking off)
- ▶ Balance the quadcopter
 - ✓ Make sure the arms are straight and perpendicular to one another
 - ✓ Balance the centre of mass
 - ✓ Straighten up all motors and level the propellers
- ▶ Re-calibrate the ESCs
- ▶ Reduce vibrations
- ▶ As a last resort, use the trimmers on the RC to correct

Regarding the C code

- ▶ Pure ANSI C,
- ▶ `/* comments are like that */`
- ▶ You cannot mix declarations with initialisations, e.g., you cannot write `float a = 0.0;`. Use initialisers instead.
- ▶ Use structures (`struct`).
- ▶ Avoid allocating memory dynamically, e.g., using `malloc`.
- ▶ Document your functions with `/** ... */`
- ▶ Perform unit tests.
- ▶ Use git or some other code versioning system.

X. Navigation

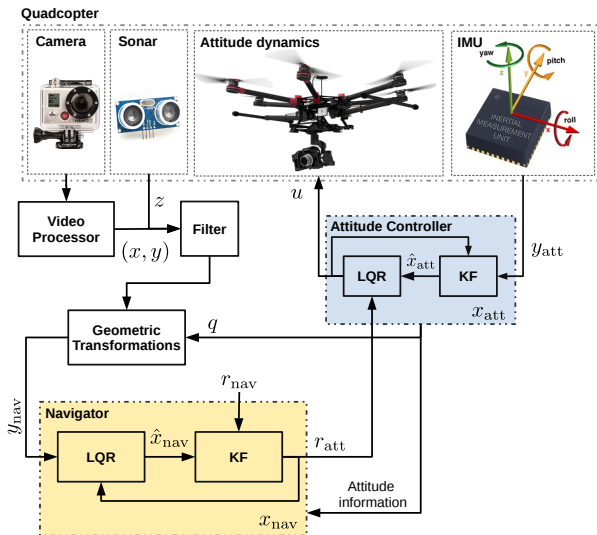
Navigation

Goal: Design a controller which will make the quadcopter fly autonomously, hover at specified positions, maintain a given heading, follow prescribed paths or traverse given points in space.

The variables we want to control are (x, y, z, ψ) .

Approach: We first derive the navigation dynamics based on first principles and then we use data to identify a dynamical model which we use to auto-pilot the quadcopter.

Navigation — the closed-loop system



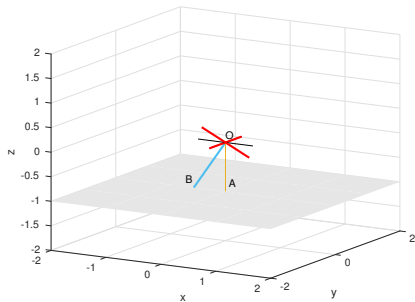
The navigation system need to collaborate with the image processing module and the attitude control module.

High-tilt corrections of measurements

- ▶ At low-tilt mode, we may assume that the camera and the sonar face (approximately) downwards and that the (x, y, z) measurements are reliable
- ▶ The tilt of the quadcopter may significantly alter these measurements
- ▶ We shall first see how the tilt affects the measured altitude
- ▶ And then we will use a quaternion-based correction technique.

Have a look at slide 140: the block “Geometric transformations” uses the current quaternion q to correct the (x, y, z) measurements.

High-tilt correction of sonar measurements

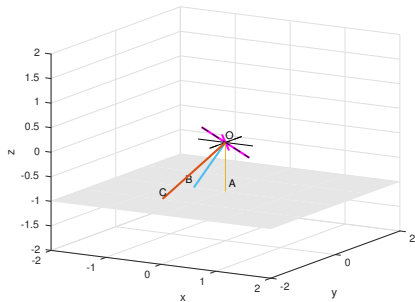


Tilted about y -axis by $\theta = \widehat{BOA}$. $\triangle BAO$ is a right-angled triangle. We measure $\bar{z} = |OB|$, but need

$$z = |OA| = \frac{\bar{z}}{\cos \theta}.$$

This example is only to gain some intuition. In practice, use the methodology on slides 144 to 147.

High-tilt correction of sonar measurements



Tilted about x and y -axes by
 $\theta = \widehat{BOA}$ and $\phi = \widehat{COB}$.

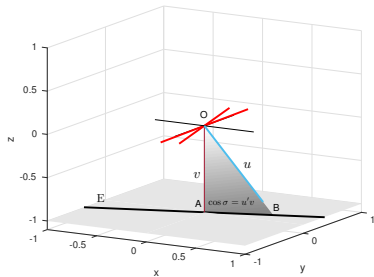
$\triangle CAO$ is a right-angled triangle

We measure $\tilde{z} = |OC|$, but need

$$z = |OA| = \frac{\tilde{z}}{\cos \theta \cos \phi}.$$

This example is only to gain some intuition. In practice, use the methodology on slides 144 to 147.

Quaternion-based correction



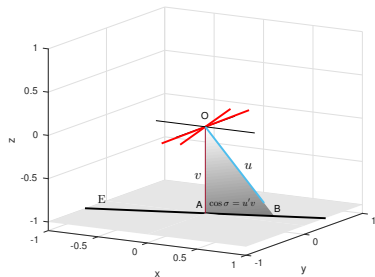
Define a unit vector $v = (0, 0, -1)$ facing downwards and rotate it using q to obtain a vector u , that is

$$\begin{bmatrix} 0 \\ u \end{bmatrix} = q \otimes \begin{bmatrix} 0 \\ v \end{bmatrix} \otimes q^{-1}.$$

Note that $\|u\| = 1$. The angle between u and v is $\cos \sigma = u'v$.

Exercise: show that $u_x = -2(q_0 q_2 + q_1 q_3)$, $u_y = 2(q_0 q_1 - q_2 q_3)$ and $q_z = -q_0^2 + q_1^2 + q_2^2 - q_3^2$.

Quaternion-based correction

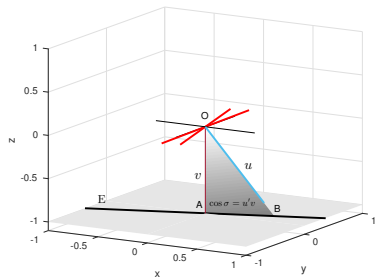


Let $\bar{z} = |OB|$ be the sonar measurement.
Then, the altitude $z = |OA|$ is

$$z = \cos \sigma \cdot \bar{z} = u'v \cdot \bar{z}.$$

If the quadcopter is upright, $u = v$ and $u'v = \|u\|^2 = 1$, so $z = \bar{z}$.

Quaternion-based correction

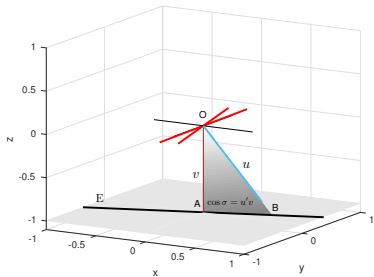


By virtue of the Pythagorean Theorem on $\triangle BAO$ (\hat{A} is a right angle), the length $|AB|$ is

$$|AB| = \sqrt{\bar{z}^2 - z^2}$$

Both A and B lie on a line E whose direction is defined by the first two components of $u = (u_x, u_y, u_z)$.

Quaternion-based correction



On the (x, y) -plane, define the vector $\hat{u} = (u_x, u_y)$. Then, the x and y errors, namely $e = (e_x, e_y)$, are given by

$$e = |AB| \cdot \frac{\hat{u}}{\|\hat{u}\|}.$$

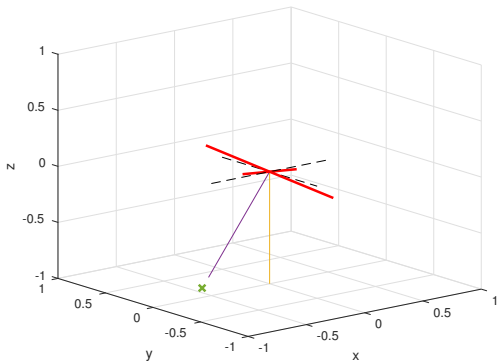
Example (i)

```
A = [-.5 0 0; .5 0 0];           % arm #1
B = [0 .5 0; 0 -.5 0];          % arm #2
v = [0; 0; -1];                 % vector facing down
z_ = 1.101718;                  % measured altitude
theta = 15; phi = 20; psi = 30; % rotation
q = quatconj(deg2quat(psi, theta, phi));
u = quatrotate(q, v')';
A_ = quatrotate(q, A);          % rotated arm #1
B_ = quatrotate(q, B);          % rotated arm #2
cos_sigma = u'*v;
z = z_*cos_sigma;               % actual altitude
AB = sqrt(z_^2 - z^2);          % length |AB|
e = AB*u(1:2)/norm(u(1:2));     % (x,y)-corrections
```


Example (ii)

```
figure; hold on;
% Quadcopter, original upright position:
plot3(A(:,1), A(:,2), A(:,3), '--', 'Color', 'k');
plot3(B(:,1), B(:,2), B(:,3), '--', 'Color', 'k');
plot3([0 v(1)], [0 v(2)], [0 v(3)]);
plot3([0 u(1)], [0 u(2)], [0 u(3)]);
% Quadcopter, rotated by p:
plot3(A_(:,1), A_(:,2), A_(:,3), 'r', 'linewidth', 2);
plot3(B_(:,1), B_(:,2), B_(:,3), 'r', 'linewidth', 2);
% Position correction:
plot3(e(1), e(2), -1, 'x');
% Figure configuration (camera, grid, labels):
campos([-5.3 -6.5 2.8]); axis([-1 1 -1 1 -1 1]);
grid on; xlabel('x'); ylabel('y'); zlabel('z');
```

Example (iii)



(Dashed black lines): original position, (Red lines): rotated quadcopter, (Orange vertical line): vector v , (Purple tilted line): Vector u , (Point marked with green \times): correction e .

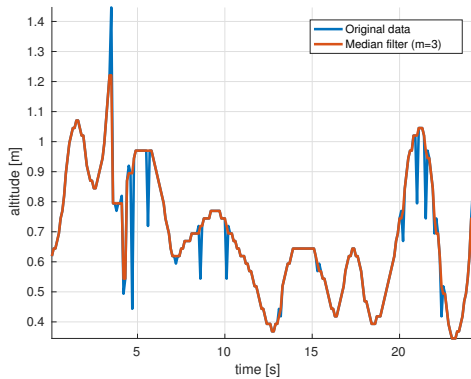
Median filter

Problem: Raw measurements, such as sonar measurements, may give measurements which are outright outliers (e.g., altitude measurements of 0 or $5m$ while hovering around $1m$). Any kind of moving average, complementary or Kalman-type filters will be affected by these outliers.

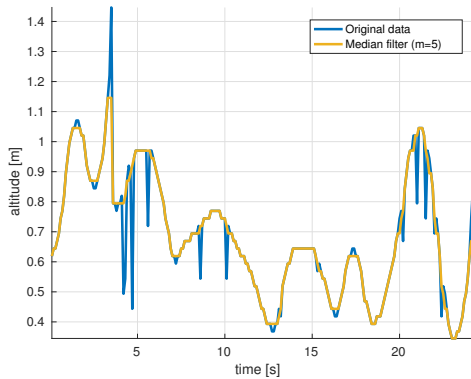
Solution: Store the past m measurements (m should preferably be an odd number), that is $\{y_{k-m+1}, \dots, y_k\}$ and take the median value; this is the filtered value \hat{y}_k .

To compute the median of m numbers $\{a_1, \dots, a_m\}$ with $m \geq 3$ an odd integer sort them into a sequence $\{a_{(1)}, \dots, a_{(m)}\}$ and take $a_{(\lceil m/2 \rceil)}$.

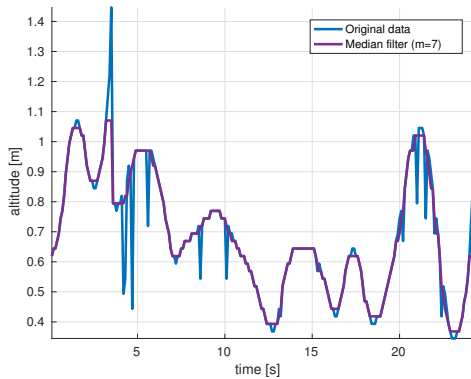
Median filter



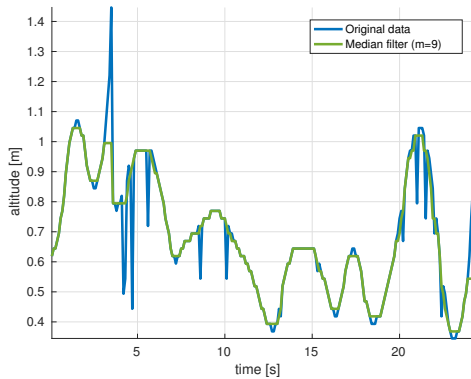
Median filter



Median filter



Median filter



Median filter — Implementation

In MATLAB, use `medfilt1(data, m)`. In C one may use the standard library function `qsort` to sort an array of numbers in ascending order.

Measurement fusion for localisation

Measurements:

- ▶ Sonar (z)
- ▶ Camera (x, y)
- ▶ Accelerometer (a_x, a_y, a_z)
- ▶ Total acceleration estimated from \hat{n}_i

Models:

- ▶ Kinematics: $\ddot{\mathbf{x}} = \mathbf{a} - (0, 0, g)^\top$
- ▶ Total acceleration: $\mathbf{a} = \mathbf{F}/m$, where $\mathbf{F} = \mathbf{F}(q, n_i)$

Methods:

- ▶ Kalman filter with bias rejection
- ▶ LQR with integral action

Modelling

The total force which is applied by the 4 propellers is

$$F = C_T \rho D^4 \sum_{i=1, \dots, 4} n_i^2,$$

and the (norm of the) total acceleration which is due to the propellers is F/m , where n_i is the frequency of rotation of propeller i .

Modelling

To find the direction of the total acceleration we need to use the quaternion which described the orientation of the quadcopter, that is

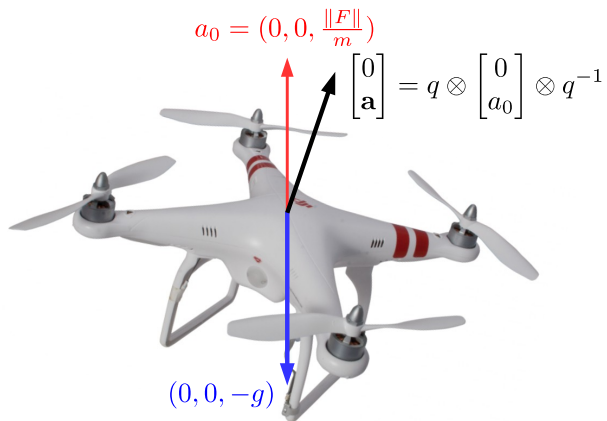
$$\begin{bmatrix} 0 \\ a_{\text{prop}} \end{bmatrix} = q \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{F}{m} \end{bmatrix} \otimes q^{-1}.$$

In other words, we rotate the vertical upward vector $(0, 0, \frac{F}{m})$ by the quaternion q . Then the total acceleration is

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = a_{\text{prop}} + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix},$$

where we simply subtracted the gravitational acceleration from a_z .

Modelling



Modelling

Then, the (x, y, z) -dynamics of the quadcopter's motion is

$$\ddot{x} = a_x,$$

$$\ddot{y} = a_y,$$

$$\ddot{z} = a_z.$$

Where (a_x, a_y, a_z) is computed as on slide 159. The above equations lead to the dynamical system

$$\dot{v} = a,$$

$$\dot{r} = v,$$

where $v = (v_x, v_y, v_z)$ is the vector of linear velocities and $r = (x, y, z)$ is the position of the quadcopter in space.

Auto-pilot

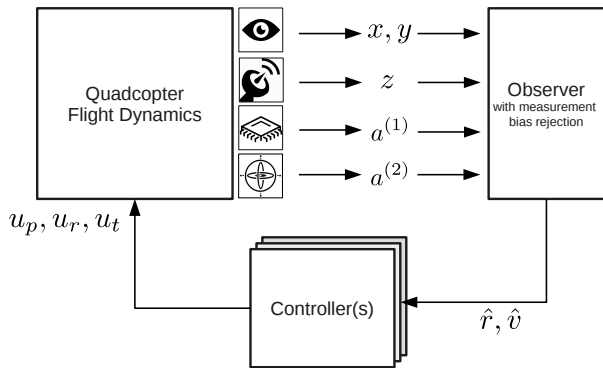
Think of auto-piloting as a replacement of the remote control. The controlled system has manipulated variables

- ▶ the *pitch* command u_p
- ▶ the *roll* command u_r
- ▶ the *throttle* command u_t and
- ▶ the *yaw* command which, however, will be disregarded here.

The system has the following measurable outputs

- ▶ the position (x, y) of the quadcopter (from the vision module)
- ▶ the altitude z (from the sonar)
- ▶ the acceleration $a^{(1)}$ from the attitude module
- ▶ the acceleration $a^{(2)}$ from the accelerometer of the IMU

Auto-pilot



Example: Kinematics in continuous time

Define $x_{\text{nav}} = (r, v) \in \mathbb{R}^6$. The observer on slide 163 is based on the very simple dynamical system (in continuous time)

$$\frac{d}{dt}x_{\text{nav}} = \underbrace{\begin{bmatrix} 0_{3 \times 3} & I_3 \\ 0_{3 \times 3} & 0_{3 \times 3} \end{bmatrix}}_{A_{\text{nav}}} x_{\text{nav}} + \underbrace{\begin{bmatrix} 0_{3 \times 3} \\ I_3 \end{bmatrix}}_{B_{\text{nav}}} \mathbf{a},$$

and $y_{\text{nav}} = (r, \mathbf{a}^{(1)}, \mathbf{a}^{(2)}) \in \mathbb{R}^{12}$, that is

$$y_{\text{nav}} = \underbrace{\begin{bmatrix} I_3 & 0_{3 \times 3} \\ 0_{6 \times 3} & 0_{6 \times 3} \end{bmatrix}}_{C_{\text{nav}}} x_{\text{nav}} + \underbrace{\begin{bmatrix} 0_{3 \times 3} \\ I_3 \\ I_3 \end{bmatrix}}_{D_{\text{nav}}} \mathbf{a}$$

We have omitted the disturbance terms.

Example: Kinematics in discrete time

The kinematics in discrete time is described by a system

$$\begin{aligned}x_{\text{nav},k+1} &= A_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + B_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k \\ y_{\text{nav},k} &= C_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + D_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k\end{aligned}$$

with matrices

$$A_{\text{nav}}^{\text{d}} = \begin{bmatrix} 1 & & & & h \\ & 1 & & & h \\ & & 1 & & h \\ & & & 1 & h \\ & & & & 1 \end{bmatrix}, \quad B_{\text{nav}}^{\text{d}} = \begin{bmatrix} h^2/2 \cdot I_3 \\ hI_3 \end{bmatrix},$$
$$C_{\text{nav}}^{\text{d}} = C_{\text{nav}}, \quad D_{\text{nav}}^{\text{d}} = D_{\text{nav}}$$

Example: Modelling noise (i)

We assume that external disturbances act on the system

$$\begin{aligned}x_{\text{nav},k+1} &= A_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + B_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k + \textcolor{red}{w}_k \\y_{\text{nav},k} &= C_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + D_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k + \textcolor{blue}{v}_k + \textcolor{green}{G}_d d_k,\end{aligned}$$

where

- ▶ $w_k \in \mathbb{R}^6$ is a zero-mean process noise
- ▶ $v_k \in \mathbb{R}^9$ is a zero-mean measurement noise
- ▶ $d_k \in \mathbb{R}^?$ is a persistent disturbance ($d_{k+1} = d_k$)
- ▶ $G_d \in \mathbb{R}^{9 \times ?}$ is a matrix that maps the effect on d_k on $y_{\text{nav},k}$

Example: Modelling noise (ii)

Kinematic model:

$$\begin{aligned}x_{\text{nav},k+1} &= A_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + B_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k + \mathbf{w}_k \\y_{\text{nav},k} &= C_{\text{nav}}^{\text{d}} \cdot x_{\text{nav},k} + D_{\text{nav}}^{\text{d}} \cdot \mathbf{a}_k + \mathbf{v}_k + G_d d_k,\end{aligned}$$

Tasks:

- ▶ Perform experiments
- ▶ Use the logs to determine the covariance matrices of \mathbf{w}_k and \mathbf{v}_k
- ▶ Choose a dimension for d_k and a matrix G_d
 - ✓ Is there a bias in r and/or $\mathbf{a}^{(1)}$ and/or $\mathbf{a}^{(2)}$?
 - ✓ Is the resulting augmented model observable?
- ▶ Implement KF with bias rejection

Example: Controller design

A very simple navigation controller is one of the form

$$\begin{aligned}u_p &= u_p^0 + K_p(\hat{x} - x^{\text{ref}}), \\u_r &= u_r^0 + K_r(\hat{y} - y^{\text{ref}}), \\u_t &= u_t^0 + K_t(\hat{z} - z^{\text{ref}}).\end{aligned}$$

This can be modified by adding integrators, use the velocity estimates and a lot more.

This part of the desing is left to you.

The equilibrium values u_p^0 , u_r^0 and u_t^0 need to be determined experimentally.

Flight modes

- ▶ **Manual mode:** The quadcopter receives yaw, pitch, roll and thrust commands from the RC (done)
- ▶ **Altitude hold mode:** The quadcopter hovers at a constant altitude while we may use the rudder on the RC to make it move around
- ▶ **Loiter mode:** The quadcopter stays at a fixed position (x, y, z) and with fixed yaw ψ
- ▶ **Reference tracking mode:** The quadcopter follows a prescribed trajectory of (x, y, z, ψ)

References

1. J.P. Hespanha, *Lecture notes on LQR/LQG controller design*, 2005. Available online at <http://goo.gl/pZzguU>.
2. E. Fresk and G. Nikolakopoulos, *Full Quaternion Based Attitude Control for a Quadrotor*, 2013 European Control Conference, Zürich, Switzerland, 2013.
3. B.L. Stevens, L.L. Frank, *Aircraft Control and Simulation*, Wiley-Interscience, 2nd Edition.
4. A. Chovancová et al., *Mathematical Modelling and Parameter Identification of Quadrotor (a survey)*, Procedia Eng. 96, 2014, pp. 172–181.
5. J.-J. Xiong and E.-H. Zheng, *Position and attitude tracking control for a quadrotor UAV*, ISA Transactions 53(3), 2014, pp. 725–731.
6. Y. Yang, *Analytic LQR Design for Spacecraft Control System Based on Quaternion Model*, J. Aerosp. Eng. 25(3), 2012.
7. Y. Yang, *Quaternion based model for momentum biased nadir pointing spacecraft*, Aerospace Sci. & Tech. 14, 2010.
8. Y. Yang, *Spacecraft attitude determination and control: Quaternion based method*, Annual Reviews in Control 36(2), 2012.