Lecture 13

- State Estimation

- **Open-loop state estimation**
- **Closed-loop state estimation**
  - **Error dynamics**
  - **Guidelines for setting estimator poles**
- **Separation Principle**
  - **Using the estimation in feedback:** $u = -K\hat{x}$

Two reasons for state estimation:

1. Not all states are measured directly (e.g., encoder tracks cart position; velocity not measured)
2. Measurements may have noise (e.g., sensor noise, or round-off error due to quantization, etc.)

We want the state for our feedback, $u = -Kx$.

Can we use "estimate" of state, via: $u = -K\hat{x}$?

$Y = CX + DU$ (if no noise in measurements)

AKA: Observer design.

State Estimation – Problem Statement

Sometimes, feedback control requires we "estimate" some of the states because, for example:

1. We might not sense all states in the output vector y, e.g.:

$$X = \begin{bmatrix} \dot{x} \\ x \end{bmatrix} \qquad y = [0 \quad 1]X + [0]u$$

y = position (velocity not measured)

2. Our measurements might be noisy, e.g.:

$$Y = CX + DU + V$$

V: measurement noise

PROBLEM STATEMENT: If we start with some "initial value" as an ESTIMATE of the state vector (for k=0), how can ESTIMATE the states for k>=0 (over time)?

Before: "state feedback": $u = -Kx$

Now: x is not perfectly known, $u = -K\hat{x}$

$\hat{x}$: est. of x

use an estimate

Approach 1: Mimic the known dynamics

- Make an initial guess, xhat = $\hat{x}$, of the (nx1) initial state, x.
  - *There will likely be some error in this guess…*
- Run a "digital twin", simulating linearized dynamics.
- Assume u is known perfectly: you are setting it.

Below, there is no noise.
- We will revisit effects of noise later…

State Estimation – "Open loop" approach

Start with some "initial value" as an ESTIMATE of the states…
…and assume this vector of value evolves via "Ax+Bu" dynamics.

State: $x(k+1) = A_d x(k) + B_d u(k)$

Estimate of state: $\hat{x}(k+1) = A_d \hat{x}(k) + B_d u(k)$ ← common alternate notation for "error"

Error in the esimate: $e_x(k+1) = x(k+1) - \hat{x}(k+1)$ or: $\tilde{x}(k+1)$

e_x: error →

Error dynamics?

notation for error in our lectures

$e_x = x - \hat{x}$, so…

$x_{k+1} = A_d x_k + B_d u_k$

$-(\hat{x}_{k+1} = A_d \hat{x}_k + B_d u_k)$

$(x_{k+1} - \hat{x}_{k+1}) = A_d(x_k - \hat{x}_k) + B_d(u_k - u_k)$ ($u_k - u_k \to 0$)

$e_{k+1} = A_d e_k$ ← same poles as open-loop system.

What happens?

The vector of errors, $e_x = x - \hat{x}$, has dynamics identical to the actual state.

i.e., whatever the initial conditions are for the error vector at k=0, they evolve over time as: $ze_x = A_d e_x$

Approach 1: What happens?

Instead of n states, we now have 2n states:

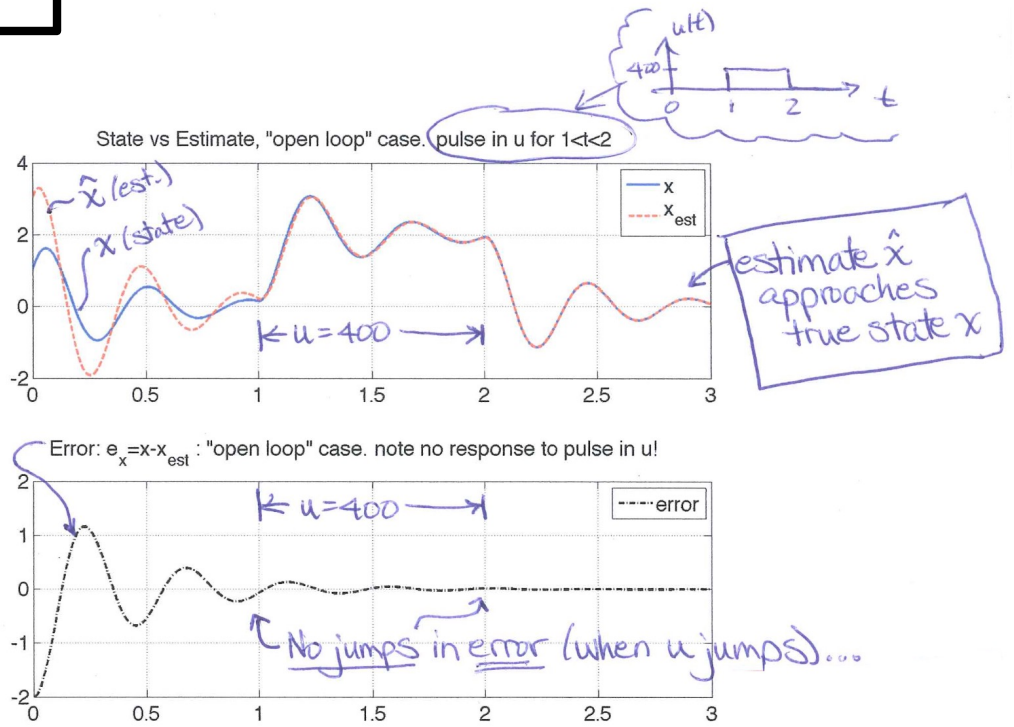
- The original (nx1) vector x
- Plus, an (nx1) vector, $e_x(k)$, of errors

Example (below, at right):

- State x has some initial condition
- Error, $e_x = x - \hat{x}$, has its own initial condition

$$(x_{k+1} - \hat{x}_{k+1}) = A_d(x_k - \hat{x}_k) + B_d(u_k - u_k)^{0}$$

$$e_{k+1} = A_d e_k$$ ← same poles as open-loop system.



In top subplot, the position variable of a 2nd-order system decays via stable, underdamped poles of matrix A.

- The state, x, is perturbed by the "pulse" in input, u

In bottom subplot, the error has the same dynamics, given by eig(A).

- The error IS NOT PERTURBED by the input u, because we know what u is.

Recall from last page: $ze_x = A_d e_x$

Approach 2: Use a feedback signal

For feedback, we only have access to output y via: $y = Cx + Du$.

- So, we might not have full access to states in x.
- But: can't we reason about **what output we *expect* to measure**, if our estimate is correct… ??

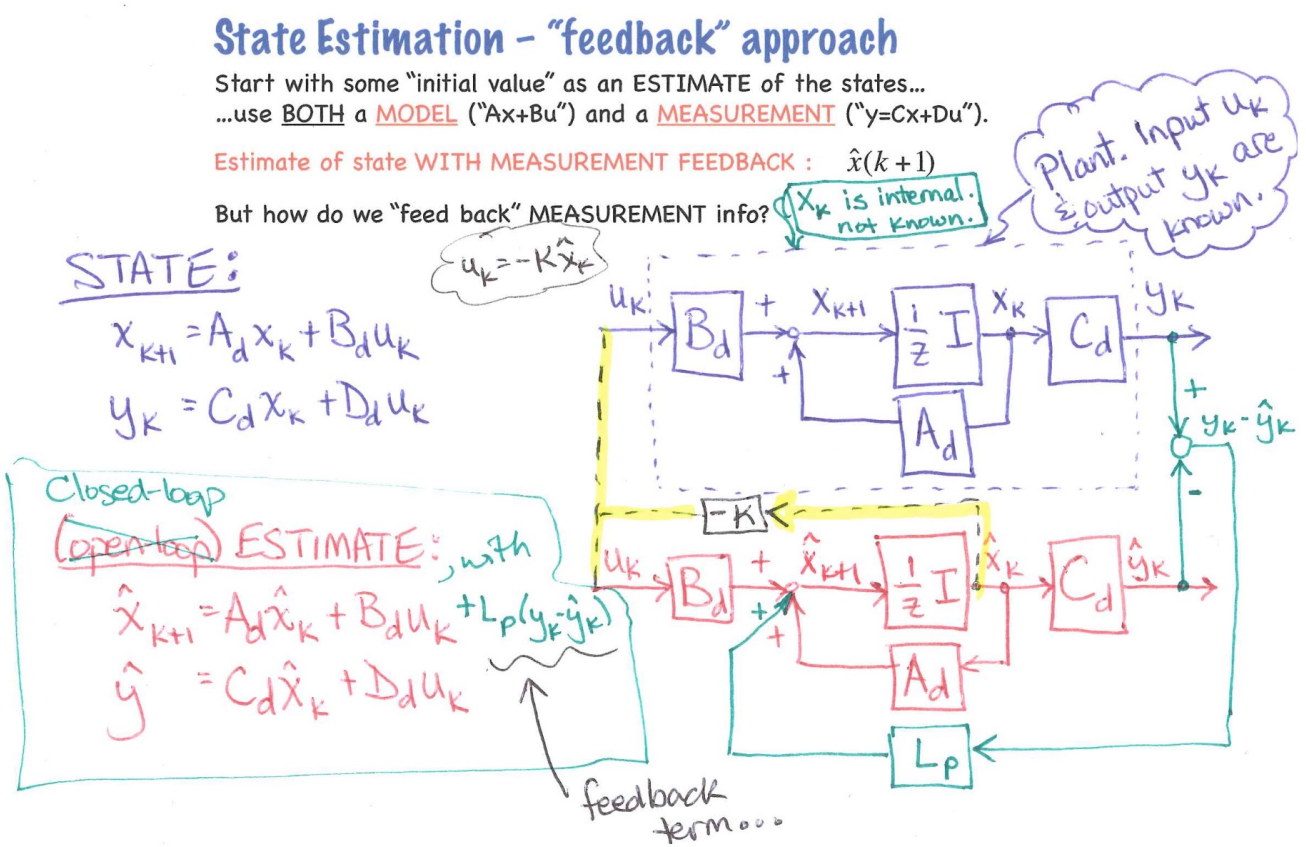
What to do :

- Compare what we have access to, y, to what we expect that output to be, based on the estimate: $\hat{y} = C_d\hat{x} + D_d u$
- Scale this, to add in a feedback "correction factor" that is: $L_p(y_k - \hat{y}_k)$

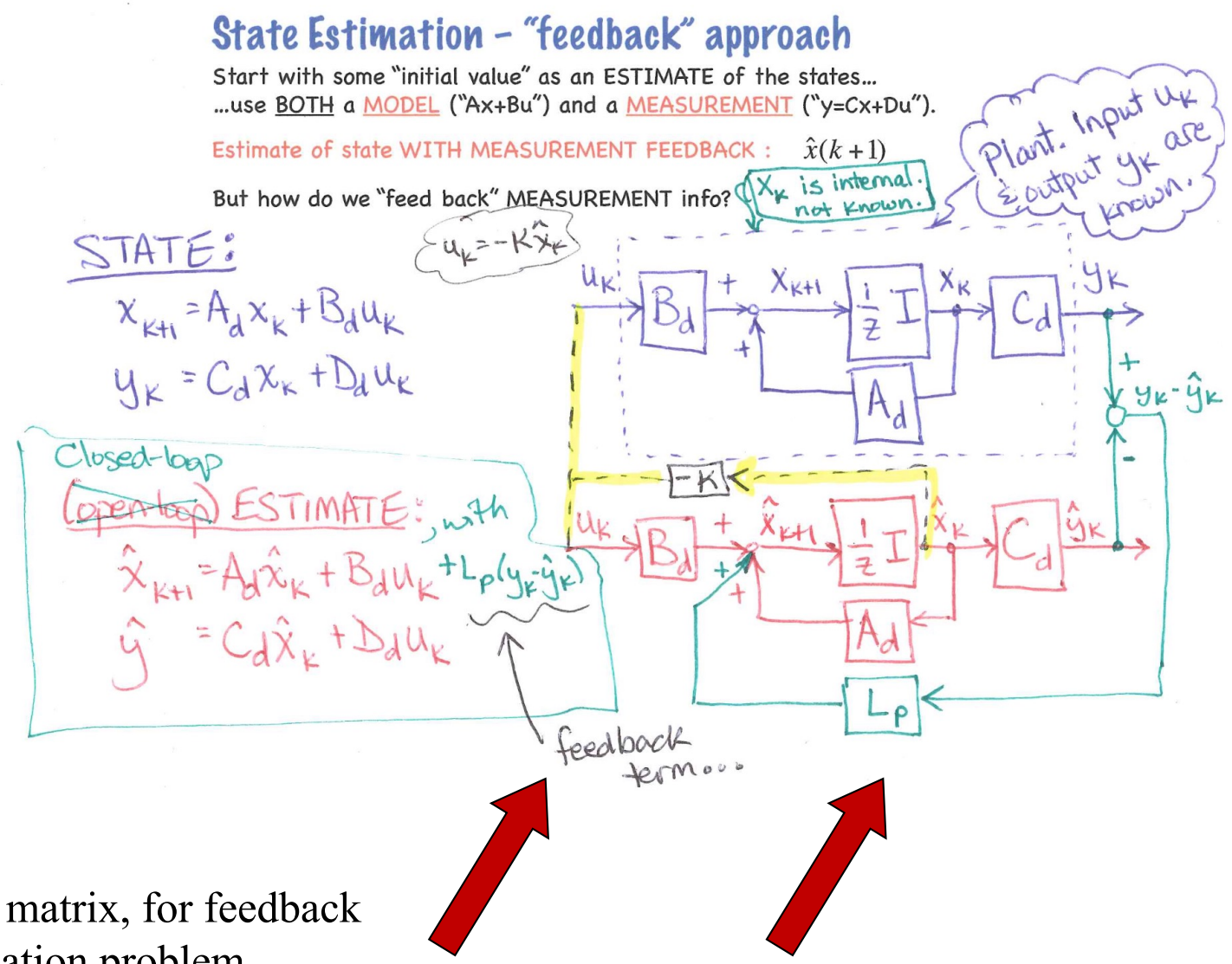
<u>At top</u>: The linear system for the actual state dynamics.

<u>Below</u>: The "digital twin" tracks an estimate of the state.

L_p times the difference between measured and expected outputs provides a feedback signal, to drive the error in estimate to zero.



Approach 2: Use a feedback signal



L_p is a gain matrix, for feedback in the estimation problem.

Approach 2: What happens, mathematically?

Now with feedback:

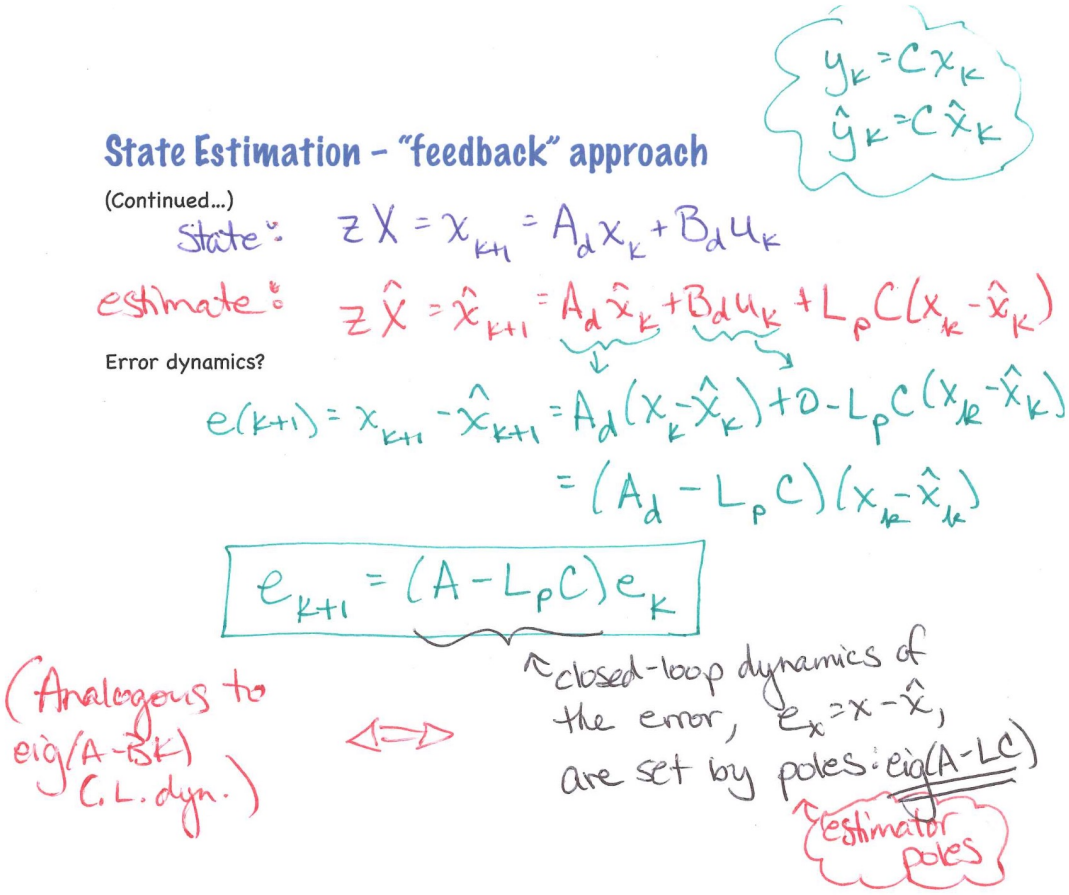
The additional **error states** have their own dynamics over time, given by: $\text{eig}(A - L_p C)$.

So, the initial error decays (recall, assuming no noise so far) via those poles: $\text{eig}(A - L_p C)$,
…so long as those poles are stable.

This is it, for the error dynamics.

However, the error acts as a signal input to affect the actual states.

(To be discussed more later…)



Approach 2: Predictor vs Current Estimator

On a computer, it takes finite time to computer the estimate.

So, we're usually solving for what is technically called the "predictor" estimator gains, in L_p.

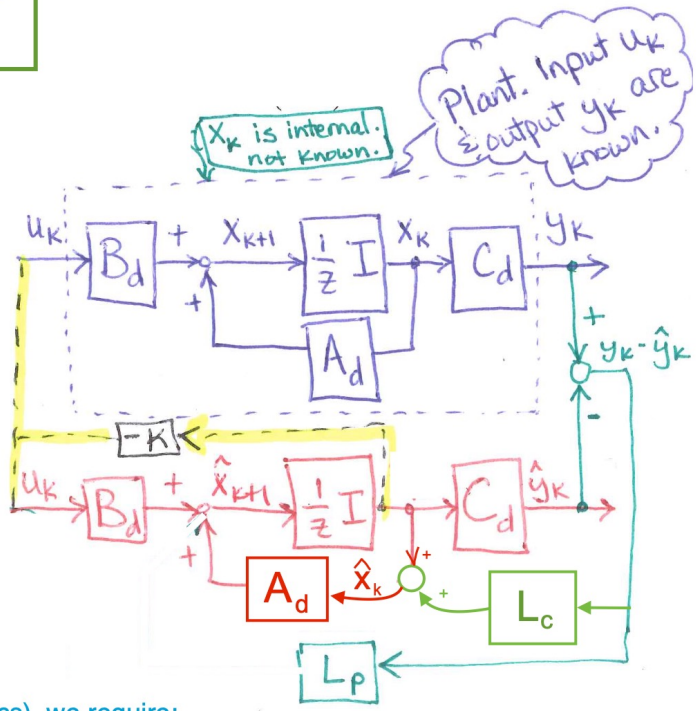
(i.e., are we predicting x_k or are we predicting x_{k+1}?)

Lp is the "predictor" estimator gain matrix.
It is used in feedback to predict x(k+1).

Lc is the "current" estimator gain matrix.
It is used to predict x(k).

Let us "cut" the signal from Lp.

At right is where the feedback from Lc enters the block diagram.

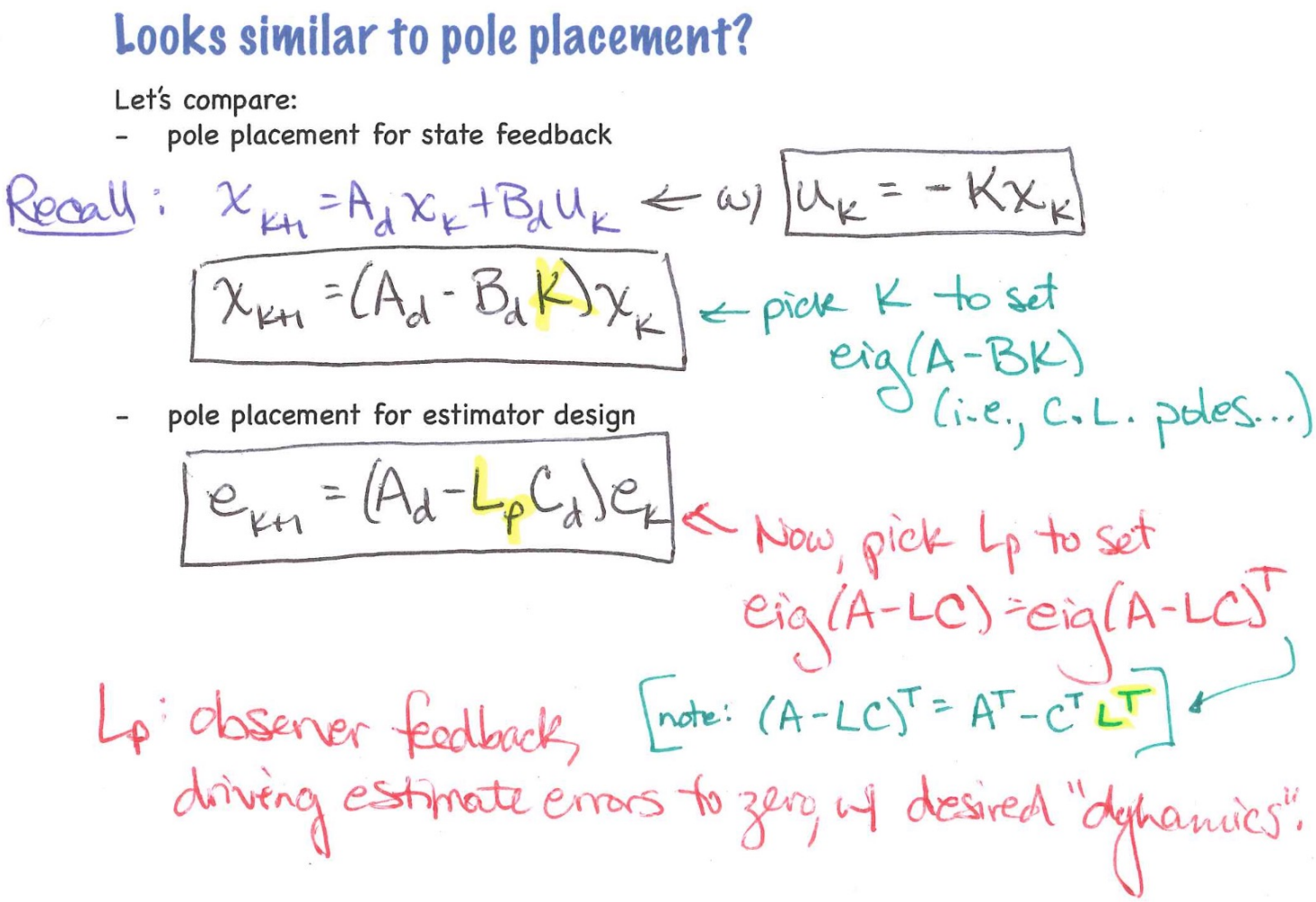


Note that xhat(k) has now moved, since it is defined only AFTER the feedback contribution (from Lc) is added in.

For the same closed loop poles (i.e., dynamics), we require:

Lp = Ad * Lc , so Lc = inv(Ad) * Lp

Approach 2: Pole placement (applied with new details…)

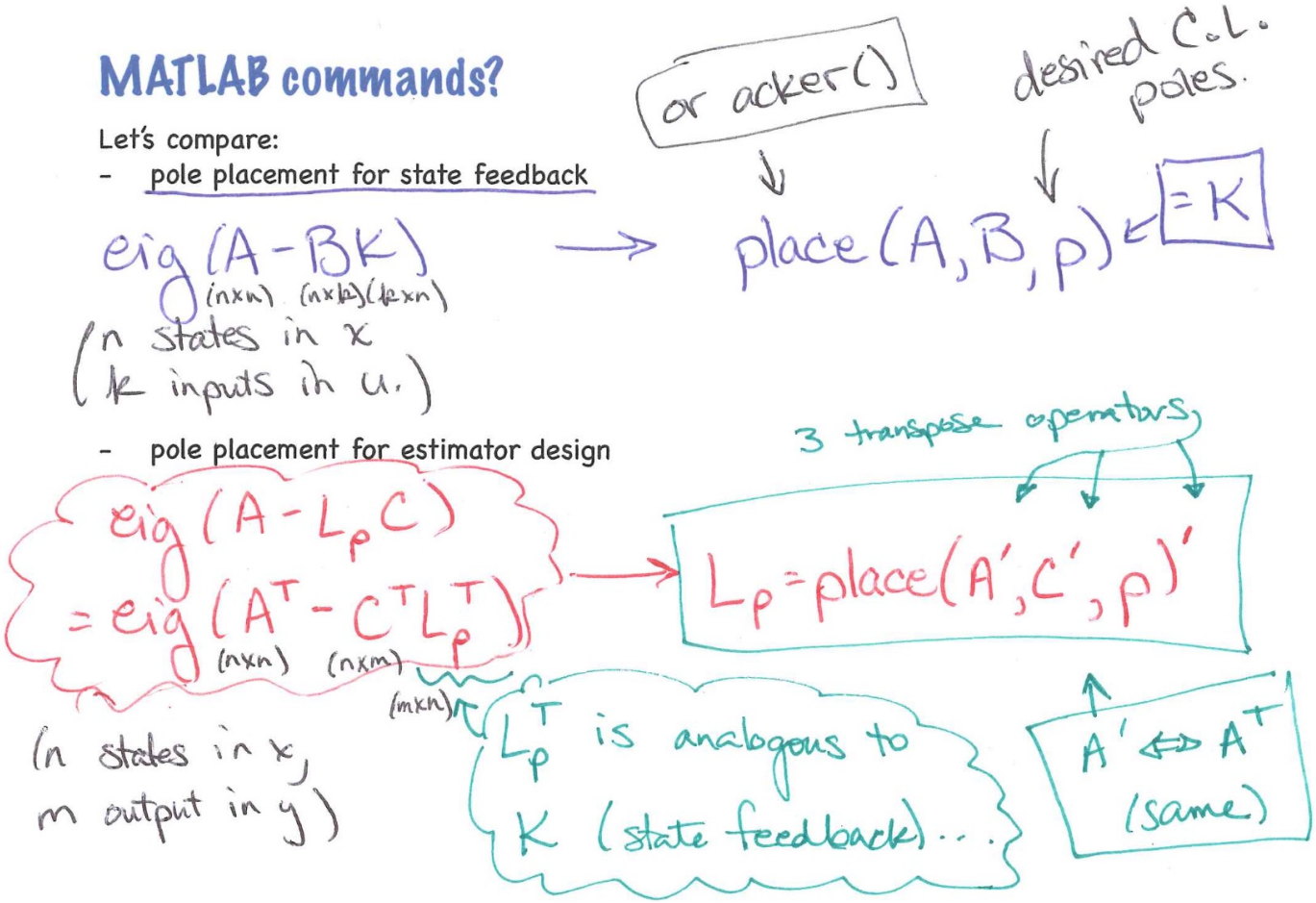


Approach 2: Matlab commands

For control, `K = place(A, B, pc_des)`

For estimation, `Lp = place(A', C', pe_des)'`

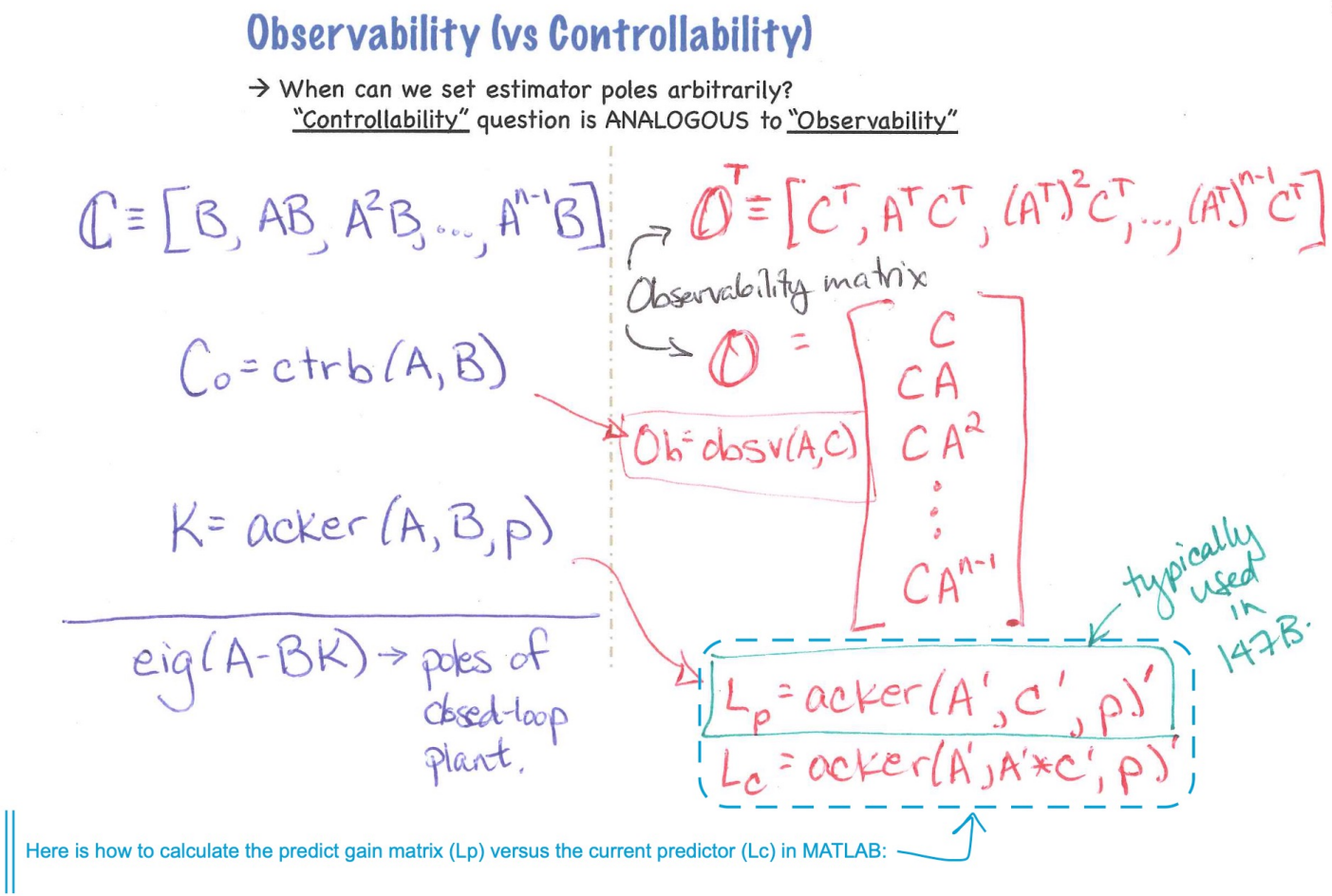
Note there are three (3) transpose characters, for the command above.



Approach 2: **Observability** is similar to Controllability

Basically, we replace: A and B
with: A^T and C^T

Then, make sure the corresponding matrix is "full rank" (to set n poles, via n free variables).



Approach 2: How to pick estimator poles..

...but how do you choose estimator poles?

---

1. Want "error dynamics" to be "faster" than controlled state dynamics.
   - i.e., fast decay; minimal overshoot

---

2. Do not want large gains to amplify sensor noise.
   - Y = CX + n, where "n" is sensor noise

Approach 2: Guidelines, to set estimator poles

Gain matrix L_p sets poles (i.e., dynamics) of the estimator error.

Here is the "feedback correction" to the estimate:

$$z\hat{x} = A\hat{x} + Bu + L_p(y - \hat{y})$$

Now, assume:

$$y = Cx + Du + n$$

Add noise to y.

where **n is sensor noise.**

Larger gains in L_p will generally result in:

- FASTER poles, so error dies away more rapidly,
- But also AMPLIFICATION of the sensor noise!

→ **Trade-off!**

Typically, aim for estimator poles
2x-4x "faster" than eig(A-BK),
without too much overshoot.

This issue is addressed more formally by "Kalman filtering", in which Lp is set in an optimal way to minimize noise in the estimate. (This topic is scheduled to be presented in Lecture 16...)

Separation Principle

> Question:
> If we do not use the true state but instead use the estimate for feedback via $u = -K\hat{x}$, what happens? *(What are the poles? Is it stable?)*

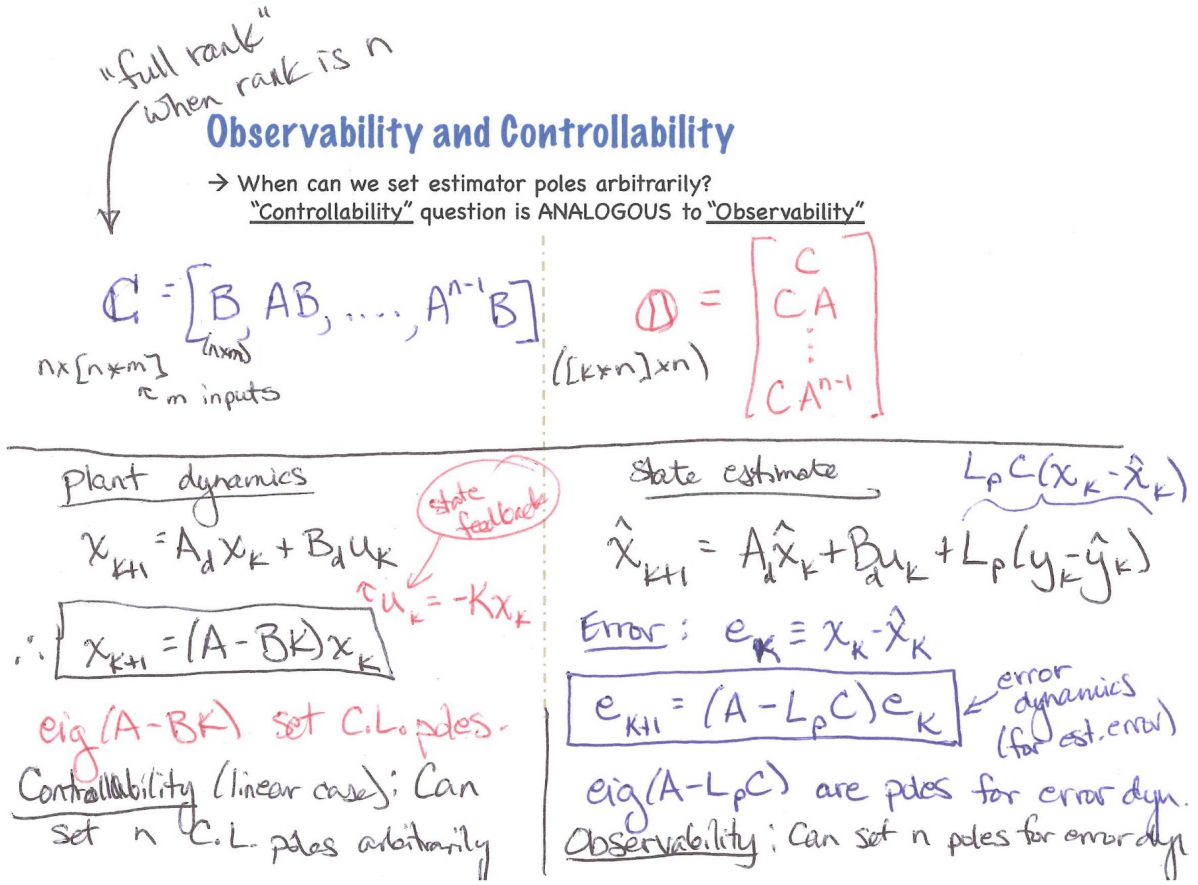
Brief Answer:

The error dynamics are given only by eig(A-L_pC).

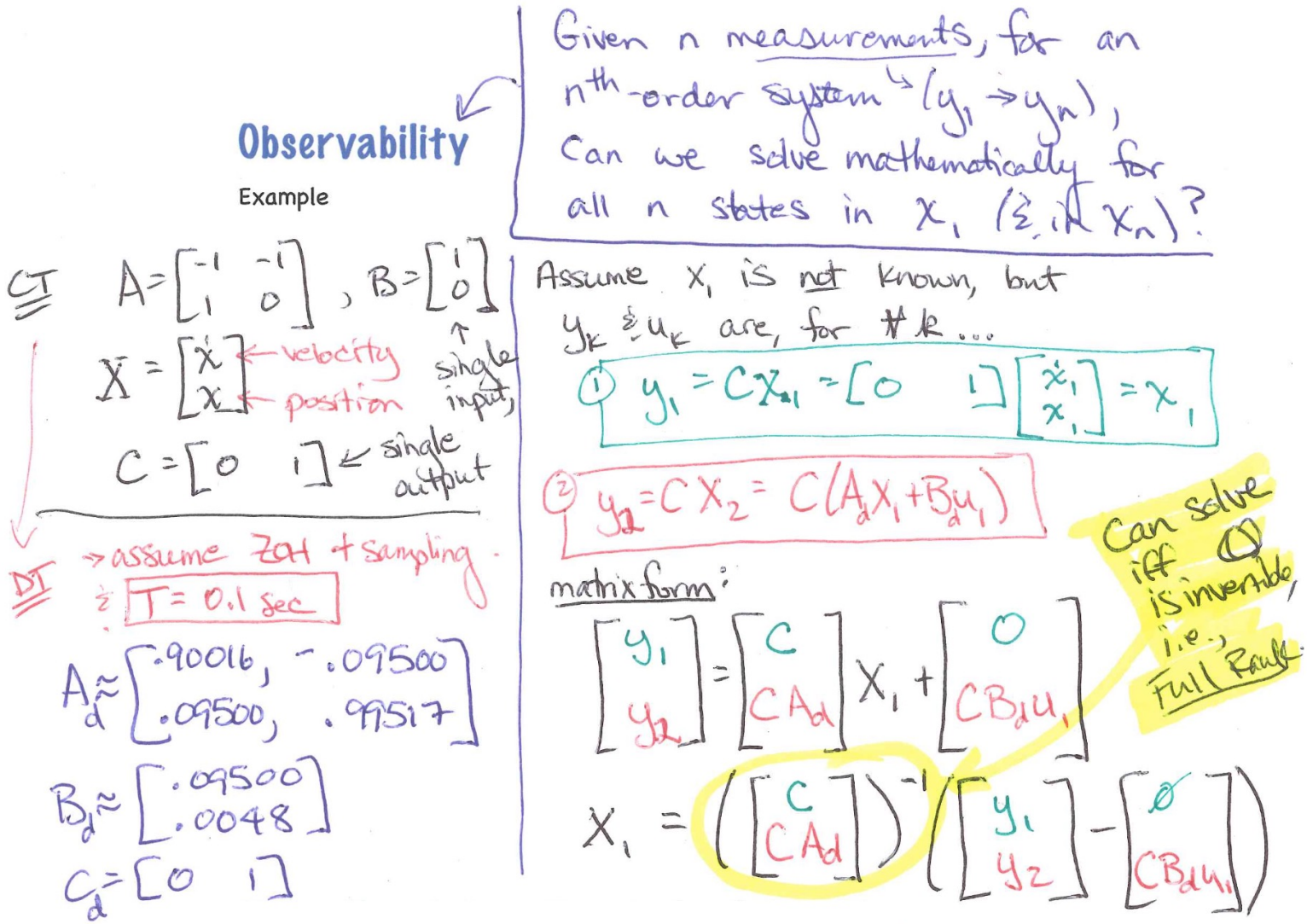
The state dynamics end up being "driven" by the error (as a signal), in addition to having their own closed-loop poles given by eig(A-BK).

So, the entire system now has twice as many poles (i.e., 2n poles), which are the combination of both:

eig(A-L_pC)
eig(A-BK)



Separation Principle



Separation Principle

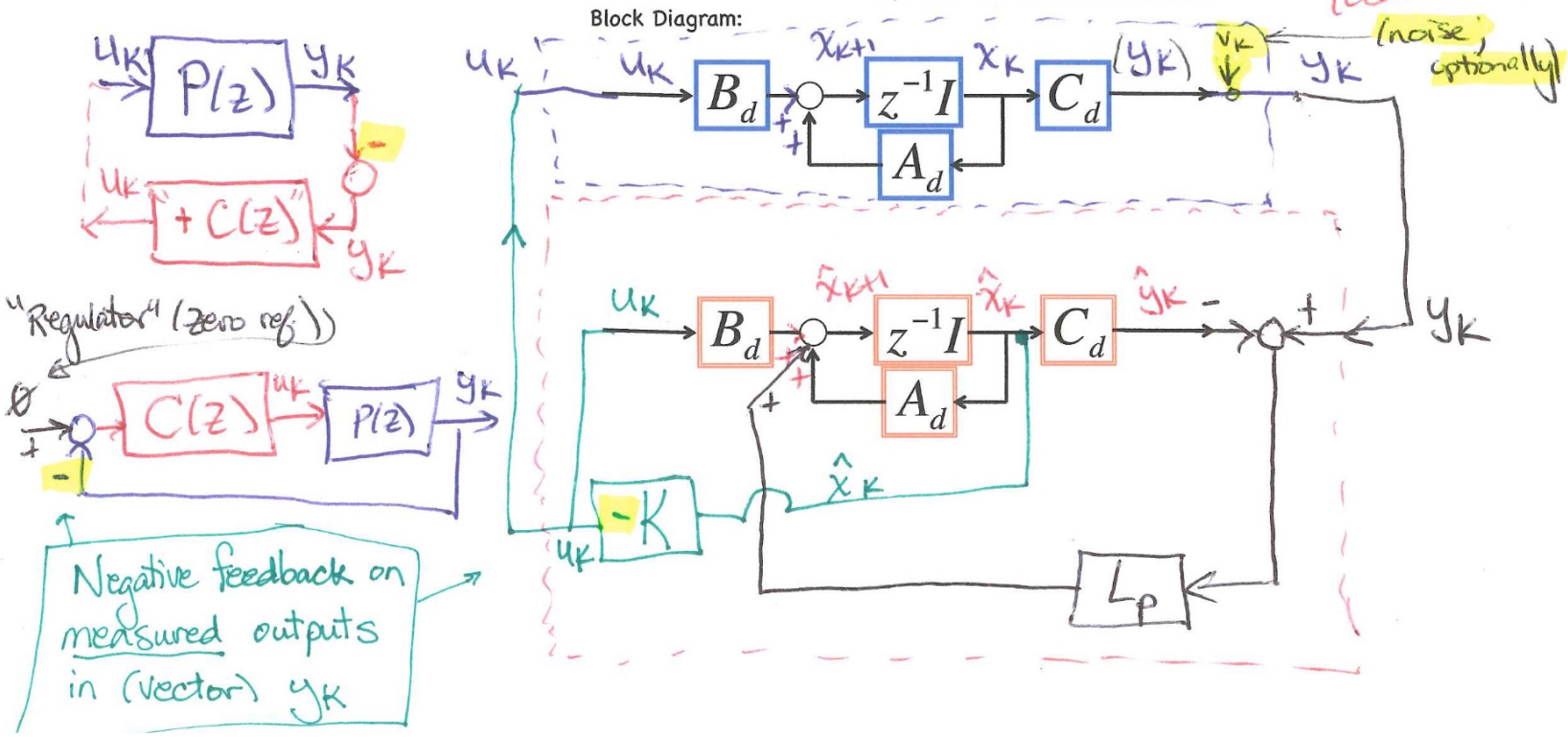
Recall STATE FEEDBACK
Control uses the true state:

$$u(k) = -Kx(k)$$

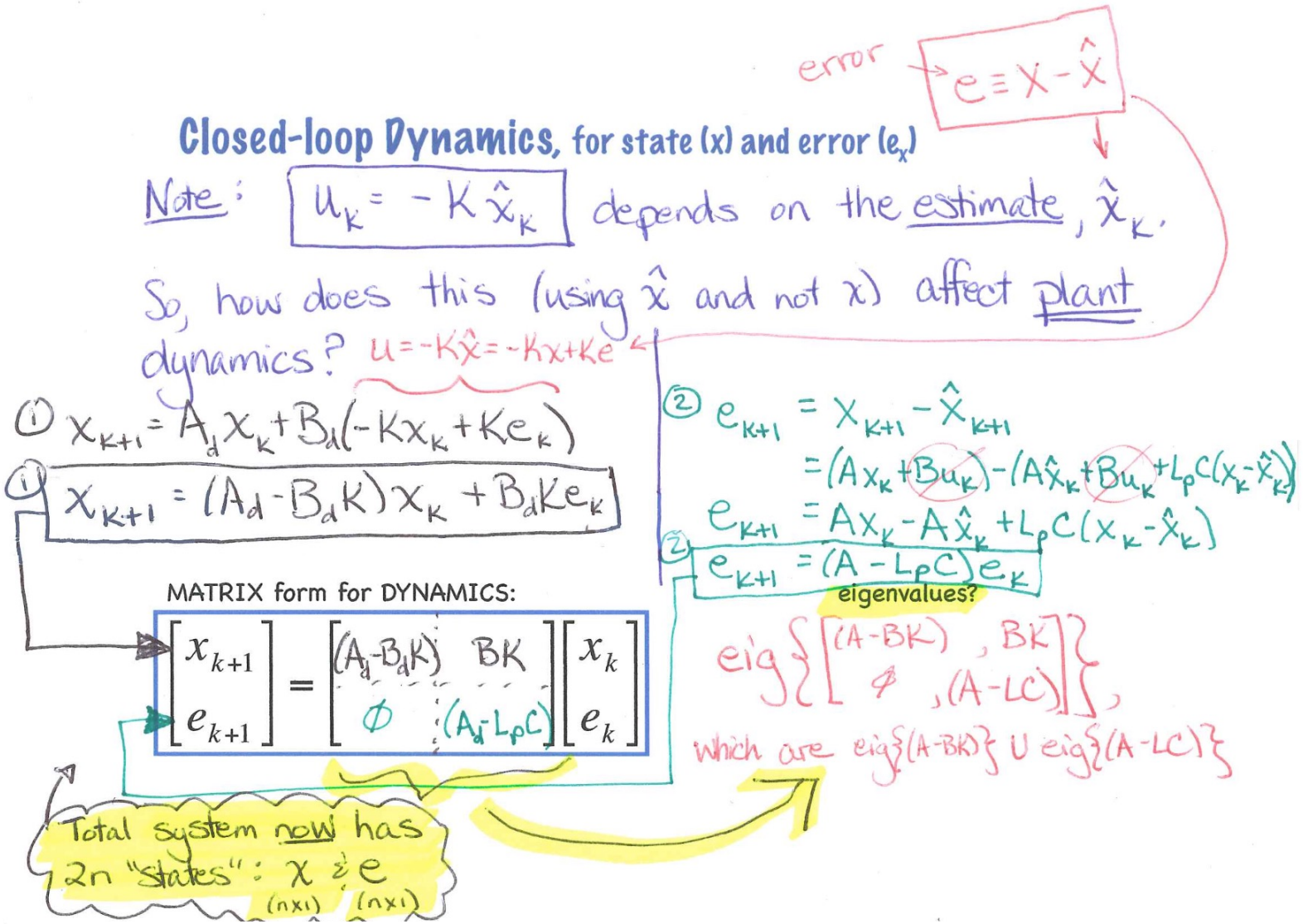
What if we use the state ESTIMATE for feedback control, instead?

$$u(k) = -K\hat{x}(k)$$

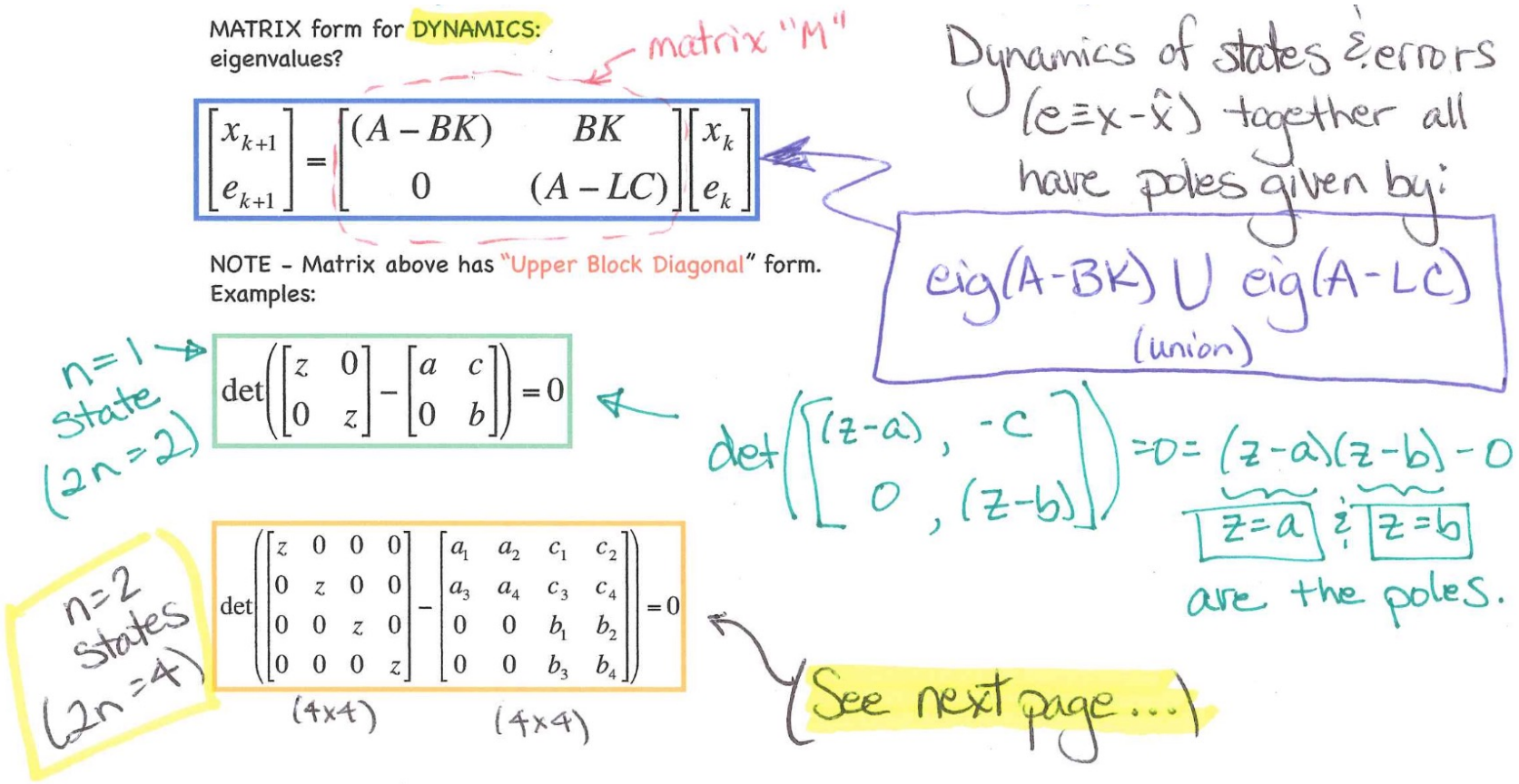
← uses $\hat{x}$ (estimate)



Separation Principle: Closed-loop dynamics



Separation Principle : Closed-loop dynamics



Separation Principle : Examples

Eigenvalue examples, via MATLAB

Below is code to "see" some of the ideas for yourself – at least for our toy examples:

```
syms a1 a2 a3 a4 b1 b2 b3 b4 c1 c2 c3 c4 z
A = [a1 a2; a3 a4];
B = [b1 b2; b3 b4];
C = [c1 c2; c3 c4];
M = [A C; zeros(2,2) B]
M =
[ a1, a2, c1, c2]
[ a3, a4, c3, c4]
[  0,  0, b1, b2]
[  0,  0, b3, b4]
zI = z*eye(4)
zI =
[ z, 0, 0, 0]
[ 0, z, 0, 0]
[ 0, 0, z, 0]
[ 0, 0, 0, z]

det(zI-M)
ans =
z^4 - a4*z^3 - b1*z^3 - b4*z^3 - a1*z^3 + a1*a4*z^2 - a2*a3*z^2 + a1*b1*z^2 + a1*b4*z^2
+ a4*b1*z^2 + a4*b4*z^2 + b1*b4*z^2 - b2*b3*z^2 + a1*a4*b1*b4 - a1*a4*b2*b3 -
a2*a3*b1*b4 + a2*a3*b2*b3 - a1*a4*b1*z + a2*a3*b1*z - a1*a4*b4*z + a2*a3*b4*z -
a1*b1*b4*z + a1*b2*b3*z - a4*b1*b4*z + a4*b2*b3*z

simplify(det(zI-M))
ans =
(a2*a3 - a1*a4 + a1*z + a4*z - z^2)*(b2*b3 - b1*b4 + b1*z + b4*z - z^2)
```

$$\det\left(\begin{bmatrix} z & 0 & 0 & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & z \end{bmatrix} - \begin{bmatrix} a_1 & a_2 & c_1 & c_2 \\ a_3 & a_4 & c_3 & c_4 \\ 0 & 0 & b_1 & b_2 \\ 0 & 0 & b_3 & b_4 \end{bmatrix}\right) = \det(zI - M) = 0$$

A

B

"Upper block diagonal"

"Brute force" solution, via Matlab...

...matches union of eig(A) U eig(B)

$eig(A) \rightarrow \det(zI - A) = (z-a_1)(z-a_4) - (-a_3)(-a_2) = 0$

$z^2 - (a_1+a_4)z + (a_1a_4 - a_2a_3) = 0$

$eig(B) \rightarrow z^2 - (b_1+b_4)z + (b_1b_4 - b_2b_3) = 0$

Separation Principle : The Big Take-Away…

Separation Principle

For feedback with an ESTIMATOR,

$u_k = -K\hat{x}_k$, when $\hat{x}_{k+1} = A\hat{x}_k + Bu_k + L_p(y - C\hat{x}_k)$

($y = Cx$)

1) CONTROL for nth-order system has n poles, determined by eig(A-BK).

First: Pick (dominant) closed-loop poles for eig(A-BK), (e.g., via pole placement).

2) ESTIMATION of n states results in another n poles, determined by eig(A-LC)

Then: Pick estimator poles to be 2x-4x "faster" than eig(A-BK), so they do not dominate dynamics.

3) TOTAL system now has 2*n poles:

& they are the union of eig(A-BK) & eig(A-LC).

i.e., we can separate design of all 2n poles via two uncoupled (sets of) equations.

"separation principle"

Separation Principle : **Practical Strategy** (to place poles)

How to place poles, to set "K" and Lp"?

Controller poles:

Use classic control design specifications, (from 147a)

CT (s-plane) ⟹ DT (z-plane)

ω_d, t_{rise}, M_p, etc

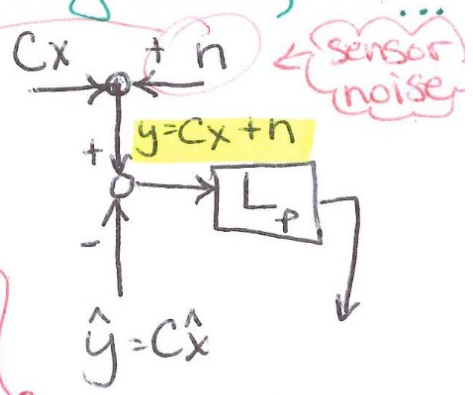
$$z = e^{sT}$$

Estimator poles:

Why 2x-4x faster than eig(A-BK)? Ans: eig(A-BK) should dominate (meaning have "slowest" decay rates), AND...

Effects of Noise?

$$\begin{bmatrix} x_{k+1} \\ e_{k+1} \end{bmatrix} = \begin{bmatrix} (A-BK) & BK \\ 0 & (A-LC) \end{bmatrix} \begin{bmatrix} x_k \\ e_k \end{bmatrix} + \begin{bmatrix} 0 \\ -L \end{bmatrix} n_k$$



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

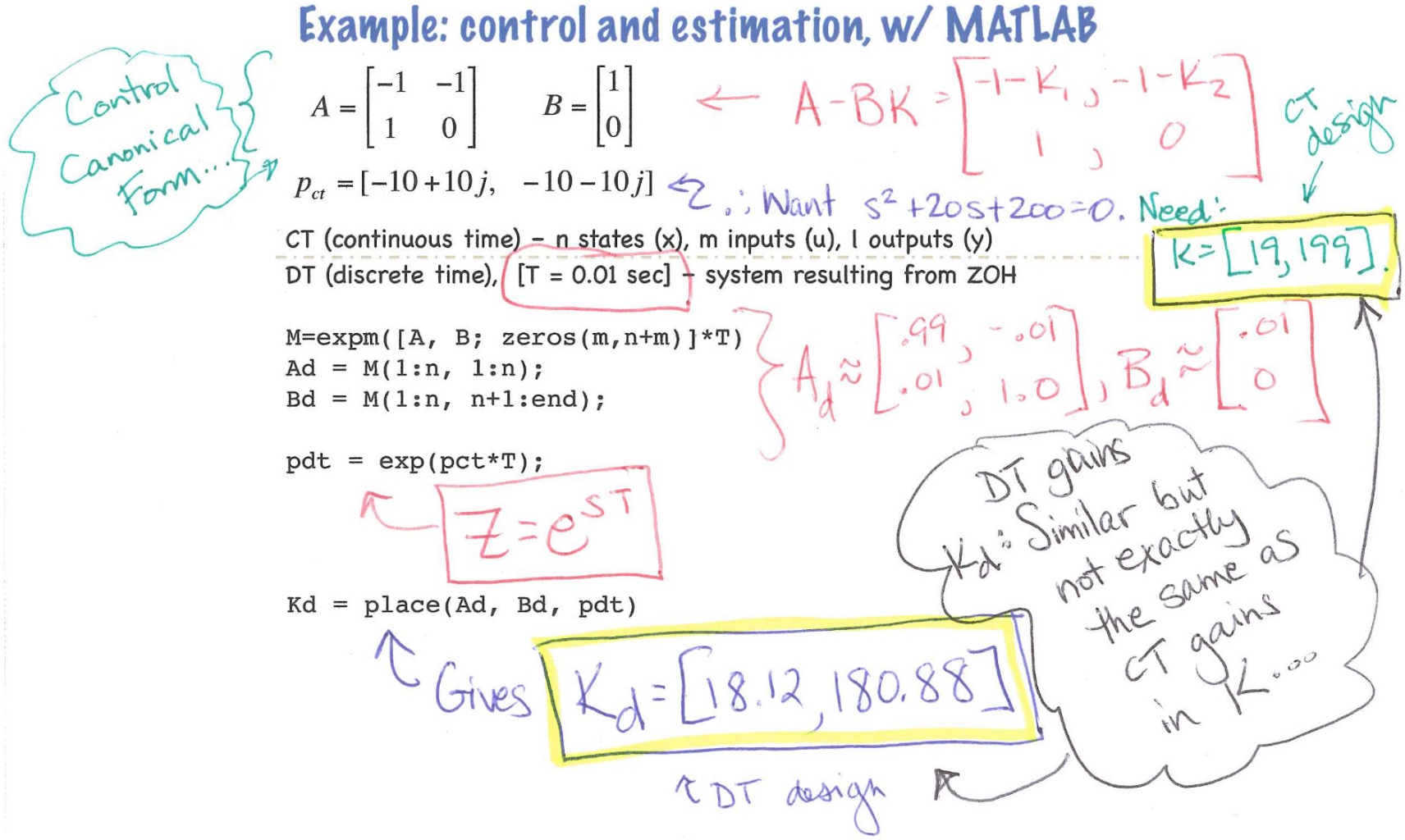
$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + L_p(Cx_k + n_k - C\hat{x}_k)$$

$$e_{k+1} = x_{k+1} - \hat{x}_{k+1} = A(x_k - \hat{x}_k) - L_p(Cx_k - C\hat{x}_k + n_k)$$

$$\therefore \boxed{e_{k+1} = (A - L_pC)e_k - L_p n_k}$$

Derivation...

Separation Principle : First, set DT controller poles…



Separation Principle : Set estimator poles 2x to 4x faster

Below, pct are the closed-loop poles of eig(A-BK).
pe_ct are the CT closed-loop poles of eig(A-LC). (i.e., estimator poles)
pe_dt are the DT estimator (closed-loop) poles.

Example: continued

Estimator poles, 2x - 4x faster.

```
C = [0 1];
pe_ct = 3*pct;
pe_dt = exp(pe_ct*T);
Lpc = place(A',C',pe_ct)'
Lpd = place(Ad',Cd',pe_dt)'
Kd = place(Ad, Bd, pdt)

Xinit = [10;10];
Xest = [0;0];
N = 100
X = zeros(2,N); X(:,1) = Xinit;
Xe = X; Xe(:,1) = Xest;
u = zeros(1,N);
for n=1:N-1
    u(n) = -Kd*Xe(:,n);
    Xe(:,n+1) = (Ad-Lpd*C)*Xe(:,n) + Bd*u(n) + Lpd*C*X(:,n);
    X(:,n+1) = Ad*X(:,n) + Bd*u(n);
end
t = T*[0:N-1];
figure(1); clf; subplot(211)
plot(t,X,'.-'); subplot(212)
plot(t,X-Xe,'.-');
```

X_0

$\hat{X}_0$

mismatch between states (x) & estimate ($\hat{x}$) initially.

$u_k = -K\hat{x}_k$ ← uses estimate.

$\hat{x}_{k+1}$

← estimate update

x_{k+1}

← true state

Separation Principle : Set estimator poles 2x to 4x faster

Example:
Below, the x axis for the STATES is scaled to be 2x (on the top subplot) what the x axis for the ERROR is, on the lower subplot.

The lower plot show a response due to ONLY the (faster) estimator poles… eig(A-LC) while the upper plot state responses are due to all 4 poles: eig(A-BK) and eig(A-LC)

Because eig(A-LC) poles are FASTER (by design), the upper subplot is (by design) dominated by the slower (state feedback controller) poles, eig(A-BK).