Chapter 6

Autopilot Design
Architecture

- Path planner
- Path manager
- Path following
- Autopilot
- Unmanned Vehicle

Inputs:
- Destination, obstacles
- Waypoints
- Path Definition
- Airspeed, Altitude, Heading, commands
- Servo commands
- Wind

Outputs:
- Status
- Tracking error
- Position error
- On-board sensors

Intermediate:
- On-board sensors
- Destination, obstacles
- Waypoints
- Path Definition
- Airspeed, Altitude, Heading, commands
- Servo commands
- Wind

Reference:
Outline

• Different Options for Autopilot Design
  – Successive Loop Closure
  – Total Energy Control
  – LQR Control
Successive Loop Closure

Open-loop system

Closed-loop system
SLC: Inner Loop Closed

At frequencies below inner-loop bandwidth, approximate CLTF as 1, then design middle loop.
SLC: Two Loops Closed

At frequencies below middle-loop bandwidth, approximate CLTF as 1, then design outer loop.

Key idea: Each successive loop must be lower in bandwidth --- typically by a factor of 5 to 10.

---

Lateral-directional Autopilot

Course Control

Roll Control

\[ F(s) = \frac{k_{p\phi}}{s + \alpha} \]

\[ \delta_a = \frac{a_{\phi 2}}{s + a_{\phi 1}} \]

\[ \frac{1}{s} \]

\[ \phi = \frac{g}{V_g} \]

\[ \chi \]

\[ \chi^c + \]

\[ k_{p\chi} \]

\[ e_{\chi} \]

\[ \phi^c \]

\[ k_{i\chi} \]

\[ s \]

\[ \frac{1}{s} \]

\[ \frac{g}{V_g} \]

\[ \chi \]

\[ rc = 0 \]

\[ \delta_r \]

\[ -1 \]

\[ -C_r (s^2 + 2\zeta_n s + \omega_n^2) \]

\[ (s + p_{sp})(s^2 + 2\zeta_{dr} \omega_{n_{dr}} s + \omega_{n_{dr}}^2) \]

\[ r \]

\[ r^c = 0 \]

\[ k_r \]

\[ \frac{\tau_r s}{\tau_r s + 1} \]

\[ \frac{-C_r (s^2 + 2\zeta_n s + \omega_n^2)}{(s + p_{sp})(s^2 + 2\zeta_{dr} \omega_{n_{dr}} s + \omega_{n_{dr}}^2)} \]

\[ \frac{-C_r (s^2 + 2\zeta_n s + \omega_n^2)}{(s + p_{sp})(s^2 + 2\zeta_{dr} \omega_{n_{dr}} s + \omega_{n_{dr}}^2)} \]

Roll Autopilot

\[
H_\phi/\phi^c(s) = \frac{k_{p\phi} a_\phi}{s^2 + (a_\phi + a_\phi k_{d\phi})s + k_{p\phi} a_\phi} = \frac{\omega_{n\phi}^2}{s^2 + 2\zeta_\phi \omega_{n\phi} s + \omega_{n\phi}^2}
\]

Closed Loop TF

Canonical 2\textsuperscript{nd}-order TF

Design parameters are \( e_\phi^{\text{max}} \) and \( \zeta_\phi \)

Gains are given by

\[
k_{p\phi} = \frac{\delta_a^{\text{max}}}{e_\phi^{\text{max}}}
\]

\[
k_{d\phi} = \frac{2\zeta_\phi \omega_{n\phi} - a_\phi}{a_\phi}
\]

Roll Autopilot

\[ H_{\phi/\phi^c}(s) = \frac{k_{p_\phi} a_{\phi_2}}{s^2 + (a_{\phi_1} + a_{\phi_2} k_{d_\phi}) s + k_{p_\phi} a_{\phi_2}} = \frac{\omega_{n_\phi}^2}{s^2 + 2\zeta_\phi \omega_{n_\phi} s + \omega_{n_\phi}^2} \]

Closed Loop TF

Canonical 2\textsuperscript{nd}-order TF

Design parameters are $\omega_{n_\phi}$ and $\zeta_\phi$

Gains are given by

\[
\begin{align*}
    k_{p_\phi} &= \frac{\omega_{n_\phi}^2}{a_{\phi_2}} \\
    k_{d_\phi} &= \frac{2\zeta_\phi \omega_{n_\phi} - a_{\phi_1}}{a_{\phi_2}}
\end{align*}
\]

Implementation:

\[
\delta_a(t) = k_{p_\phi} (\phi^c(t) - \phi(t)) - k_{d_\phi} p(t).
\]

Roll Autopilot

• The book suggests using an integrator on roll in the roll loop to correct for steady state error.

• Our current suggestion is to not have an integrator on inner loops including the roll loop.
  
  • Integrators add delay and instability -> not a good idea for the inner-most loops.

  • An integrator will be used on the course loop to correct for steady state values.
Course Hold Loop

For the course loop, note the presence of the input disturbance. Using a PI controller for course, the response to the course command and disturbance is given by

\[
\chi = \frac{k_{p_x} g/V_a s + k_{i_x} g/V_a}{s^2 + k_{p_x} g/V_a s + k_{i_x} g/V_a} \chi^c + \frac{g/V_a s}{s^2 + k_{p_x} g/V_a s + k_{i_x} g/V_a} d_{\chi}
\]

Note:

- There is a zero in the response to the course command \( \chi^c \).

- The presence of the zero at the origin ensures rejection of low frequency disturbances.
TF Zero Affects Response

With a zero, the canonical $2^{nd}$-order TF is given by

$$H = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

Note that $\zeta$ has a different effect when the zero is present.
Course Hold Loop

\[
\chi = \frac{(k_{p_x}g/V_g)s + (k_{i_x}g/V_g)}{s^2 + (k_{p_x}g/V_g)s + (k_{i_x}g/V_g)} \chi_c + \frac{(g/V_g)s}{s^2 + (k_{p_x}g/V_g)s + (k_{i_x}g/V_g)} d_\chi
\]

Equating coefficients to canonical TF gives:

\[
\omega_{n_x}^2 = k_{i_x}g/V_g \quad \text{and} \quad 2\zeta_x \omega_{n_x} = k_{p_x}g/V_g
\]

or

\[
\omega_{n_x} = \frac{1}{W_x} \omega_{n_{\phi}} \quad \quad \quad \quad k_{p_x} = 2\zeta_x \omega_{n_x} V_g/g \quad \quad \quad \quad k_{i_x} = \omega_{n_x}^2 V_g/g
\]

Design parameters are bandwidth separation \( W_x \) and damping ratio \( \zeta_x \)

Implementation:

\[
\phi^c(t) = k_{p_x}(\chi^c(t) - \chi(t)) + k_{i_x} \int_{-\infty}^{t} (\chi^c(\tau) - \chi(\tau))d\tau.
\]

- The rudder is used to counter act the yaw rate.

- The washout filter makes it so that the yaw damper only counteracts high frequency yaw rate.

- The washout filter is similar to a dirty derivative with gain $\tau_r$
Yaw Damper Implementation

Yaw damper:
\[ \delta_r(s) = k_r \left( \frac{s}{s + \frac{1}{\tau_r}} \right) r(s) \]

In time domain:
\[ \dot{\delta}_r = -\frac{1}{\tau_r} \delta_r + k_r \ddot{r} \]

Integrating:
\[ \delta_r(t) = -\frac{1}{\tau_r} \int_{-\infty}^{t} \delta_r(\sigma) d\sigma + k_r r(t) \]

\[
\begin{align*}
\dot{\xi} &= -\frac{1}{\tau_r} \xi + k_r r \\
\delta_r &= -\frac{1}{\tau_r} \xi + k_r r.
\end{align*}
\]

State-space form:

Discrete-time:
\[
\begin{align*}
\xi_k &= -\frac{1}{\tau_r} \xi_{k-1} + k_r r_{k-1} \\
\delta_r &= -\frac{1}{\tau_r} \xi_k + k_r r_k.
\end{align*}
\]
**Yaw Damper Implementation**

Discrete-time:

\[
\xi_k = -\frac{1}{\tau_r} \xi_{k-1} + k_r r_{k-1}
\]

\[
\delta_{r_k} = -\frac{1}{\tau_r} \xi_k + k_r r_k.
\]

```python
class yawDamper:
    def __init__(self, k_r, tau_r, Ts):
        # set initial conditions
        self.xi = 0.
        self.Ts = Ts
        self.k_r = k_r
        self.tau_r = tau_r

    def update(self, u):
        self.xi = self.xi
        + self.Ts * (-1/self.tau_r * self.xi + self.k_r * r)
        delta_r = -1/self.tau_r * self.xi + self.k_r * r
        return delta_r
```
Lateral Autopilot - Summary

If model is known, the the design parameters are

Inner Loop (roll attitude hold)

- $\omega_{n\phi}$ - Error in roll when aileron just saturates.

- $\zeta_\phi$ - Damping ratio for roll attitude loop.

Outer Loop (course hold)

- $W_\chi > 1$ - Bandwidth separation between roll and course loops.

- $\zeta_\chi$ - Damping ratio for course hold loop.

Yaw damper (if rudder is available)

- $\tau_\tau$ - cut off frequency for wash-out filter.

- $k_\tau$ - gain for yaw damper.
Lateral Autopilot – In Flight Tuning

If model is not known, and autopilot must be tuned in flight, then the following gains are tuned one at a time, in this specific order:

**Inner Loop (roll attitude hold)**

- $k_{d\phi}$ - Increase $k_{d\phi}$ until onset of instability, and then back off by 20%.
- $k_{p\phi}$ - Tune $k_{p\phi}$ to get acceptable step response.

**Outer Loop (course hold)**

- $k_{p\chi}$ - Tune $k_{p\chi}$ to get acceptable step response.
- $k_{i\chi}$ - Tune $k_{i\chi}$ to remove steady state error.

**Sideslip hold (if rudder is available)**

- $k_{p\beta}$ - Tune $k_{p\beta}$ to get acceptable step response.
- $k_{i\beta}$ - Tune $k_{i\beta}$ to remove steady state error.
Longitudinal Flight Regimes

- $h^c$
  - \( \text{sat}(h^c, h + h_{\text{hold}}) \)
- $h$
  - Regulate altitude by commanding pitch attitude
  - \( \text{sat}(h^c, h - h_{\text{hold}}) \)
  - Regulate airspeed by commanding throttle
- $h_{\text{take off}}$
  - Take-off zone
    - Full throttle
    - Regulate pitch to a fixed $\theta^c$
Pitch Attitude Hold

\[ H_{\theta/\theta^c}(s) = \frac{k_{p\theta} a_{\theta_3}}{s^2 + (a_{\theta_1} + k_{d\theta} a_{\theta_3})s + (a_{\theta_2} + k_{p\theta} a_{\theta_3})} = \frac{K_{\theta DC} \omega_{n\theta}^2}{s^2 + 2\zeta_{\theta} \omega_{n\theta} s + \omega_{n\theta}^2} \]

Closed Loop TF

Equating coefficients, the gains are given by

\[ k_{p\theta} = \frac{\omega_{n\theta}^2 - a_{\theta_2}}{a_{\theta_3}} \quad k_{d\theta} = \frac{2\zeta_{\theta} \omega_{n\theta} - a_{\theta_1}}{a_{\theta_3}} \]

Design parameters are \( \omega_{n\theta} \) and \( \zeta_{\theta} \)

The DC gain is

\[ K_{\theta DC} = \frac{k_{p\theta} a_{\theta_3}}{a_{\theta_2} + k_{p\theta} a_{\theta_3}} \]

Altitude Hold Using Commanded Pitch

Provided pitch loop functions as intended, we can simplify the inner-loop dynamics to $\theta^c/\theta \approx K_{\theta_{DC}}$. 

Altitude from Pitch – Simplified

\[
h(s) = \left( \frac{K_{\theta_{DC}} V_a k_{ph} s + K_{\theta_{DC}} V_a k_{ih}}{s^2 + K_{\theta_{DC}} V_a k_{ph} s + K_{\theta_{DC}} V_a k_{ih}} \right) h^c(s) + \left( \frac{s}{s^2 + K_{\theta_{DC}} V_a k_{ph} s + K_{\theta_{DC}} V_a k_{ih}} \right) d_h(s)
\]

A PI control on altitude ensures that \( h \) tracks constant \( h^c \) with zero steady state error, and rejects low frequency disturbances.

Altitude from Pitch Gain Calculations

Equating the transfer functions

\[ H_{h/h^c} = \left( \frac{K_{\theta_{DC}} V_a k_{ph} \left( s + \frac{k_{ih}}{k_{ph}} \right)}{s^2 + K_{\theta_{DC}} V_a k_{ph} s + K_{\theta_{DC}} V_a k_{ih}} \right) h^c(s) = \frac{2\zeta \omega_n s + \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} \]

gives the coefficients

\[ k_{ih} = \frac{\omega_n^2}{K_{\theta_{DC}} V_a} \quad \quad k_{ph} = \frac{2\zeta \omega_n}{K_{\theta_{DC}} V_a} \]

where bandwidth separation is achieved by selecting

\[ \omega_{n_h} = \frac{1}{W_h} \omega_{n_{\theta}}. \]

Design parameters are bandwidth separation \( W_h \) and damping ratio \( \zeta_\theta \)

Airspeed Hold Using Throttle

\[
V_a = \left( \frac{a_{V_2} (k_{pV} s + k_{iV})}{s^2 + (a_{V_1} + a_{V_2} k_{pV})s + a_{V_2} k_{iV}} \right) V_a^c + \left( \frac{s}{s^2 + (a_{V_1} + a_{V_2} k_{pV})s + a_{V_2} k_{iV}} \right) d_V
\]

A PI control on the throttle to airspeed loop ensures that \( V_a \) tracks a constant \( V_a^c \) with zero steady state error, and rejects low frequency disturbances.

Airspeed Hold Using Throttle

Equating the transfer functions

\[ H_{V_a/V_a^c}(s) = \left( \frac{aV_2 k_{pV} s + aV_2 k_{iV}}{s^2 + (aV_1 + aV_2 k_{pV})s + aV_2 k_{iV}} \right) = \frac{2\zeta \omega_n s + \omega_n^2}{s^2 + 2\zeta \omega_n s + \omega_n^2} \]

gives the coefficients

\[ k_{iV} = \frac{\omega_{nV}^2}{aV_2} \quad k_{pV} = \frac{2\zeta V \omega_{nV} - aV_1}{aV_2} \]

Design parameters are natural frequency \( \omega_{nV} \) and damping ratio \( \zeta_V \).
Longitudinal Flight Regimes

- \( h^c \)
  - \( \text{sat}(h^c, h + h_{\text{hold}}) \)
  - \( \text{sat}(h^c, h - h_{\text{hold}}) \)
  - Regulate altitude by commanding pitch attitude
  - Regulate airspeed by commanding throttle

- \( h \)
  - \( \text{sat}(h^c, h + h_{\text{hold}}) \)
  - \( \text{sat}(h^c, h - h_{\text{hold}}) \)

- \( h_{\text{take off}} \)
  - Take-off zone
  - Full throttle
  - Regulate pitch to a fixed \( \theta^c \)
Alternative to State Machine

• An alternative is to eliminate the climb and descend modes, and to saturate the altitude command as

\[
\text{h\_c\_filtered} = \text{sat}(\text{h\_c}, \text{h} + \text{P.\_altitude\_hold\_zone}, \text{h} - \text{P.\_altitude\_hold\_zone}); \\
\text{theta\_c} = \text{altitude\_hold}(\text{h\_c\_filtered}, \text{h}, 0, \text{P}); \\
\text{delta\_t} = \text{airspeed\_with\_throttle\_hold}(\text{Va\_c}, \text{Va}, 0, \text{P});
\]

• This scheme seems to eliminate much of the adverse coupling between altitude an airspeed.
```python
class autopilot:
    def __init__(self, ts_control):
        self.roll_from_aileron = pdControlWithRate(kp, kd, limit)
        self.course_from_roll = piControl(kp, ki, Ts, limit)
        self.yaw_damper = transferFunction(num, den, Ts)
        self.pitch_from_elevator = pdControlWithRate(kp, kd, limit)
        self.altitude_from_pitch = piControl(kp, ki, Ts, limit)
        self.airspeed_from_throttle = piControl(kp, ki, Ts, limit)

    def update(self, cmd, state):
        # lateral autopilot
        chi_c = wrap(cmd.course_command, state.chi)
        phi_c = self.saturate(
            cmd.phi_feedforward + self.course_from_roll.update(chi_c, state.chi) - np.radians(30), np.radians(30))
        delta_a = self.roll_from_aileron.update(phi_c, state.phi, state.p)
        delta_r = self.yaw_damper.update(state.r)

        # longitudinal autopilot
        # saturate the altitude command
        h_c = self.saturate(cmd.altitude_command, state.h - AP.altitude_zone, st
        theta_c = self.altitude_from_pitch.update(h_c, state.h)
        delta_e = self.pitch_from_elevator.update(theta_c, state.theta, state.q)
        delta_t = self.airspeed_from_throttle.update(cmd.airspeed_command, state
        delta_t = self.saturate(delta_t, 0.0, 1.0)

        return delta, self.commanded_state
```
Longitudinal Autopilot – In Flight Tuning

If model is not known, and autopilot must be tuned in flight, then the following gains are tuned one at a time, in this specific order:

**Inner Loop (pitch attitude hold)**

- $k_{d\theta}$ - Increase $k_{d\theta}$ until onset of instability, and then back off by 20%.
- $k_{p\theta}$ - Tune $k_{p\theta}$ to get acceptable step response.

**Altitude Hold Outer Loop**

- $k_{p_h}$ - Tune $k_{p_h}$ to get acceptable step response.
- $k_{i_h}$ - Tune $k_{i_h}$ to remove steady state error.

**Airspeed Hold Outer Loop**

- $k_{p_{V_2}}$ - Tune $k_{p_{V_2}}$ to get acceptable step response.
- $k_{i_{V_2}}$ - Tune $k_{i_{V_2}}$ to remove steady state error.

**Throttle hold (inner loop)**

- $k_{p_V}$ - Tune $k_{p_V}$ to get acceptable step response.
- $k_{i_V}$ - Tune $k_{i_V}$ to remove steady state error.
PID Loop Implementation

PID control in continuous time is given by

\[ u(t) = k_p e(t) + k_i \int_{-\infty}^{t} e(\tau) d\tau + k_d \frac{de}{dt}(t) \]

where

\[ e(t) = y^c(t) - y(t). \]

Taking the Laplace transform gives

\[ U(s) = k_p E(s) + k_i \frac{E(s)}{s} + k_d s E(s). \]

Use a dirty derivative for causality and to reduce noise:

\[ U(s) = k_p E(s) + k_i \frac{E(s)}{s} + k_d \frac{s}{\tau s + 1} E(s),\]

where \(1/\tau\) is the bandwidth of the differentiator.

PID Loop Implementation

To convert to discrete time implementation, use the Tustin (or trapezoidal) rule

\[ s \mapsto \frac{2}{T_s} \left( \frac{1 - z^{-1}}{1 + z^{-1}} \right). \]

The integrator \( I(s) = \frac{1}{s} E(s) \) becomes

\[ I(z) = \frac{T_s}{2} \left( \frac{1 + z^{-1}}{1 - z^{-1}} \right) E(z). \]

Taking the inverse z-transform gives

\[ I[n] = I[n - 1] + \frac{T_s}{2} (E[n] + E[n - 1]). \]

The differentiator \( D(s) = \frac{s}{\tau s + 1} E(s) \) becomes

\[ D(z) = \frac{2}{T_s} \left( \frac{1 - z^{-1}}{1 + z^{-1}} \right) E(z) = \left( \frac{2}{2\tau + T_s} \right) (1 - z^{-1}) \]

\[ 1 - \left( \frac{2\tau - T_s}{2\tau + T_s} \right) z^{-1} \]

Taking the inverse z-transform gives

\[ D[n] = \left( \frac{2\tau - T_s}{2\tau + T_s} \right) D[n - 1] + \left( \frac{2}{2\tau + T_s} \right) (E[n] - E[n - 1]). \]
Integrator wind-up happens when the error $e(t)$ persists, causing the integrator to add area, so that $u_{\text{unsat}}$ is beyond saturation. When the error changes sign, so that $u$ should also change sign, the positive area under the integrator hold $u$ at the wrong sign until the integrator un-winds.

Anti-wind-up schemes are intended to limit the integrator from winding-up after $u$ is in saturation.
Integrator Anti-wind-up

Let the control before the anti-wind-up update be given by

\[ u_{\text{unsat}}^- = k_p e + k_d D + k_i I^- \]

and the control after the anti-wind-up update be given by

\[ u_{\text{unsat}}^+ = k_p e + k_d D + k_i I^+ . \]

Subtracting the two gives

\[ u_{\text{unsat}}^+ - u_{\text{unsat}}^- = k_i (I^+ - I^-) . \]

Therefore

\[ I^+ = I^- + \frac{1}{k_i} (u_{\text{unsat}}^+ - u_{\text{unsat}}^-) , \]

where \( u_{\text{unsat}}^+ \) is selected to be the saturation limit \( \bar{u}_{\text{sign}}(u_{\text{unsat}}^-) \).

Anti-wind-up is applied when \( |u_{\text{unsat}}^-| \geq \bar{u} \).
PID Implementation

\[
D(z) = \frac{2T_s}{\tau} \left(1 - \frac{z - 1}{1 + z - 1}\right)E(z)
\]

Transforming to the time domain, we have

\[
D[n] = \frac{2T_s}{\tau(T_s - 2\tau)} D[n-1] + \frac{2}{T_s} E[n] - \frac{2}{T_s} E[n-1].
\]  

Matlab code that implements a general PID loop is shown below.

```matlab
function u = pidloop(y_c, y, flag, kp, ki, kd, limit, Ts, tau)
persistent integrator;
persistent differentiator;
persistent error_d1;
if flag==1, % reset (initialize) persistent variables
    % when flag==1
    integrator = 0;
    differentiator = 0;
    error_d1 = 0; % d1 means delayed by one time step
end
error = y_c - y; % compute the current error
integrator = integrator + (Ts/2) * (error + error_d1); % update integrator
differentiator = (2*tau - Ts)/(2*tau + Ts) * differentiator +
    2/(2*tau + Ts) * (error - error_d1); % update differentiator
error_d1 = error; % update the error for next time through
    % the loop
u = sat(... % implement PID control
    kp * error + ...
    ki * integrator + ...
    kd * differentiator,... % derivative term
    limit... % ensure abs(u)<=limit
); % implement integrator anti-windup
if ki~=0
    u_unsat = kp*error + ki*integrator + kd*differentiator;
    integrator = integrator + Ts/kg * (u - u_unsat);
end

function out = sat(in, limit)
    if in > limit, out = limit;
    elseif in < -limit; out = -limit;
    else
        out = in;
end
```
Simulation Project

- Roll attitude loop. For Aerosonde model use $V_a = 17 \ m/s$. Note that $\phi^\text{max}$ is a design parameter. Put aircraft in trim, and command steps on roll.

- Course attitude loop. Command steps in $\chi$. Can by-pass simplified simulink files.

- For sideslip, assume no rudder, i.e., set $\delta_r = 0$.

- Pitch attitude loop. Note that $e_{\theta}^\text{max}$ is a design parameter. Don’t use simplified simulink file. Command steps in pitch angle.

- Altitude using pitch, airspeed using pitch, and airspeed using throttle: implement directly on full Simulink model.

- Implement full autopilot using state machine for longitudinal control. Simulation should be from take-off to altitude hold.
Outline

• Successive Loop Closure
• Total Energy Control
• LQR Control
Total Energy Control

- Developed in the 1980’s by Antonius Lambregts
- Based on energy manipulation techniques from the 1950’s
- Control the energy of the system instead of the altitude and airspeed

Total Energy Control

- Kinetic Energy: \( E_K \triangleq \frac{1}{2} m V_a^2 \)
- Potential Energy: \( E_P \triangleq mgh \)
- Total Energy: \( E_T \triangleq E_P + E_K \)
- Energy Difference: \( E_D \triangleq E_P - E_K \)
Total Energy Control

Original TECS proposed by Lambregts is based on energy rates:

- \( T^c = T_D + k_{p,t} \dot{E}_t + k_{i,t} \int_{t_0}^{t} \dot{E}_t \delta \tau \)
  
  - \( T_D \) is thrust needed to counteract drag
  - PI controller based on total energy rate

- \( \theta^c = k_{p,\theta} \dot{E}_d + k_{i,\theta} \int_{t_0}^{t} \dot{E}_d \delta \tau \)
  
  - PI controller based on energy distribution rate

- Stability shown for linear systems

We will show that the performance of this scheme is less than desirable.
Total Energy Control

If the trust needed to counteract drag is unknown, then one possibility is to use an integrator to find $T_D$:

- $T^c = k_{p,t} \tilde{E}_t + k_{i,t} \int \tilde{E}_t d\tau + k_{d,t} \dot{E}_t$
  - PID controller based on total energy (not energy rate)

- $\theta^c = k_{p,\theta} \tilde{E}_d + k_{i,\theta} \int \tilde{E}_d d\tau + k_{d,\theta} \dot{E}_d$
  - PID controller based on energy distribution (not rate)
Total Energy Control

Nonlinear re-derivation:

- Error Definitions
  
  \[
  \tilde{E}_K = \frac{1}{2} m \left( (V_a^d)^2 - V_a^2 \right) \\
  \tilde{E}_P = mg (h^d - h)
  \]

- Lyapunov Function
  
  \[
  V = \frac{1}{2} \tilde{E}_T^2 + \frac{1}{2} \tilde{E}_D^2
  \]

- Controller
  
  \[
  T^c = D + \frac{\tilde{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a} \\
  \gamma^c = \sin^{-1} \left( \frac{\dot{h}^d}{V_a} + \frac{1}{2mgV_a} \left( -k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right)
  \]

Total Energy Control

• Original: \[ T^c = D + k_{p,t} \frac{\dot{E}_T}{mgV_a} + k_{i,t} \frac{\tilde{E}_T}{mgV_a} \]

• Nonlinear: \[ T^c = D + \frac{\dot{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a} \]

Similar if \( k_{p,T} = mg \) and \( k_{i,T} = mgk_T \).

The nonlinear controller uses the desired energy rate.
Total Energy Control

- Modified Original (Ardupilot):

\[
\theta^c = \frac{k_{p,\theta}}{V_amg} \left( (2 - k) \dot{E}_P - k \dot{E}_K \right) + \frac{k_{i,\theta}}{V_amg} \tilde{E}_D
\]

\[k \in [0, 2]\]

- Nonlinear:

\[
\gamma^c = \sin^{-1} \left( \frac{\dot{h}^d}{V_a} + \frac{1}{2mgV_a} \left( -k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right)
\]

\[k_1 \triangleq |k_T - k_D|\]

\[k_2 \triangleq k_T + k_D\]

\[0 < k_T \leq k_D\]

Lyapunov derivation suggests potential energy error should be weighted more than kinetic energy

If the drag is unknown, then we can add an adaptive estimate:

\[ T^c = \hat{D} + \Phi^\top \hat{\Psi} + \frac{\dot{E}_T^d}{V_a} + k_T \frac{\tilde{E}_T}{V_a} \]

\[ \gamma^c = \sin^{-1} \left( \frac{\dot{h}^d}{V_a} + \frac{1}{2m g V_a} \left( -k_1 \tilde{E}_K + k_2 \tilde{E}_P \right) \right) \]

\[ \dot{\Psi} = \left( \Gamma_T \tilde{E}_T - \Gamma_D \tilde{E}_D \right) \Phi V_a \]
Total Energy Control

Step in Altitude, Constant Airspeed

Total Energy Control

Step in Airspeed, Constant Altitude

![Graph showing the response of different control systems to a step change in airspeed while maintaining constant altitude. The x-axis represents time (s) ranging from 0 to 50, the y-axis represents altitude (m) ranging from -0.4 to 0.2, and the graph also shows the response for airspeed (m/s) with values ranging from 10 to 13. Different lines represent the original TECS, nonlinear TECS, adaptive TECS, SLC, TECS PID, and ArduPilot PID.]
Total Energy Control

Step in Altitude and Airspeed

Altitude (m)

Airspeed (m/s)

Time (s)

<table>
<thead>
<tr>
<th>Original TECS</th>
<th>Nonlinear TECS</th>
<th>Adaptive TECS</th>
<th>SLC</th>
<th>TECS PID</th>
<th>ArduPilot PID</th>
</tr>
</thead>
</table>

Total Energy Control

• Observations
  – TECS seems to work better than successive loop closure.
  – Removes needs for different flight modes.
  – Nonlinear TECS seems to better, but the Ardupilot controller works very well.
Outline

• Successive Loop Closure
• Total Energy Control
• LQR Control
LQR Control

Augment the States with an Integrator.

Given the state space system

\[ \dot{x} = Ax + Bu \]
\[ z = Hx \]

where \( z \) represents the controlled output. Suppose that the objective is to drive \( z \) to a reference signal \( z_r \) and further suppose that \( z_r \) is a step, i.e., \( \dot{z}_r = 0 \). The first step is to augment the state with the integrator

\[ x_I = \int_{-\infty}^{t} (z(\tau) - z_r) \, d\tau. \]
LQR Control

Augment the States with an Integrator. (cont)

Defining the augmented state as $\xi = (x^\top, x_I^\top)^\top$, results in the augmented state space equations

$$\dot{\xi} = \bar{A}\xi + \bar{B}u,$$

where

$$\bar{A} = \begin{pmatrix} A & 0 \\ H & 0 \end{pmatrix}, \quad \bar{B} = \begin{pmatrix} B \\ 0 \end{pmatrix}.$$
LQR Control

Linear Quadratic Regulator Theory

Given the state space equation

\[ \dot{x} = Ax + Bu \]

and the symmetric positive semi-definite matrix \( Q \), and the symmetric positive definite matrix \( R \), the LQR problem is to minimize the cost index

\[ J(x_0) = \min_{u(t), t \geq 0} \int_0^\infty x^\top(\tau)Qx(\tau) + u^\top(\tau)Ru(\tau)d\tau. \]

If \((A, B)\) is controllable, and \((A, Q^{1/2})\) is observable, then a unique optimal control exists and is given in linear feedback form as

\[ u^*(t) = -K_{lqr}x(t). \]
LQR Control

Linear Quadratic Regulator Theory (cont)

The LQR gain is given by

$$K_{lqr} = R^{-1} B^\top P,$$

where $P$ is the symmetric positive definite solution of the Algebraic Riccati Equation

$$PA + A^\top P + Q - PBR^{-1} B^\top P = 0.$$
LQR Control

Linear Quadratic Regulator Theory (cont)

It should be noted that $K_{lqr}$ is the optimal feedback gains given $Q$ and $R$. The controller is tuned by changing $Q$ and $R$. Typically we choose $Q$ and $R$ to be diagonal matrices

$$Q = \begin{pmatrix} q_1 & 0 & \ldots & 0 \\ 0 & q_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & q_n \end{pmatrix}, \quad R = \begin{pmatrix} r_1 & 0 & \ldots & 0 \\ 0 & r_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & r_m \end{pmatrix},$$

where $n$ is the number of states, $m$ is the number of inputs, and $q_i \geq 0$ ensures $Q$ is positive semi-definite, and $r_i > 0$ ensure $R$ is positive definite.
LQR Control

Lateral Autopilot

As derived in Chapter 5, the state space equations for the lateral equation of motion are given by

\[ \dot{x}_{lat} = A_{lat}x_{lat} + B_{lat}u_{lat}, \]

where \( x_{lat} = (v, p, r, \phi, \psi)^\top \) and \( u_{lat} = (\delta_a, \delta_r)^\top. \)

The objective of the lateral autopilot is to drive course \( \chi \) to commanded course \( \chi_c. \) Therefore, we augment the state with

\[ x_I = \int (\chi - \chi_c) dt. \]

Since \( \chi \approx \psi, \) we approximate \( x_I \) as

\[ x_I = \int (H_{lat}x_{lat} - \chi_c) dt, \]

where \( H_{lat} = (0, 0, 0, 0, 1). \)
LQR Control

Lateral Autopilot (cont)

The augmented lateral state equations are therefore

\[ \dot{\xi}_{lat} = \bar{A}_{lat} \xi_{lat} + \bar{B}_{lat} u_{lat}, \]

where

\[ \bar{A}_{lat} = \begin{pmatrix} A_{lat} & 0 \\ H_{lat} & 0 \end{pmatrix} \quad \bar{B}_{lat} = \begin{pmatrix} B_{lat} \\ 0 \end{pmatrix} \]

The LQR controller designed using

\[ Q = \text{diag} ([q_v, q_p, q_r, q\phi, q\chi, qI]) \]
\[ R = \text{diag} ([r_{\delta_a}, r_{\delta_r}]) . \]
LQR Control

Longitudinal Autopilot

As derived in Chapter 5, the state space equations for the longitudinal equations of motion are given by

\[
\dot{x}_{lon} = A_{lon}x_{lon} + B_{lon}u_{lon},
\]

where \(x_{lon} = (u, w, q, \theta, h)\) and \(u_{lat} = (\delta_e, \delta_t)\).

The objective of the longitudinal autopilot is to drive altitude \(h\) to commanded altitude \(h_c\), and airspeed \(V_a\) to commanded airspeed \(V_{ac}\). Therefore, we augment the state with

\[
x_I = \begin{pmatrix}
\int (h - h_c)dt \\
\int (V_a - V_{ac})dt
\end{pmatrix}
\]

\[
= \int \left( H_{lon}x_{lon} - \begin{pmatrix} h_c \\ V_{ac} \end{pmatrix} \right) dt,
\]

where

\[
H_{lon} = \begin{pmatrix}
0 & 0 & 0 & 0 & 1 \\
\frac{1}{V_a} & 0 & 0 & 0 & 1 \\
\frac{1}{V_a} & 0 & 0 & 0 & 0
\end{pmatrix}.
\]
LQR Control

Longitudinal Autopilot (cont)

The augmented longitudinal state equations are therefore

$$\dot{\xi}_{lon} = \bar{A}_{lon} \xi_{lon} + \bar{B}_{lon} u_{lon},$$

where

$$\bar{A}_{lon} = \begin{pmatrix} A_{lon} & 0 \\ H_{lon} & 0 \end{pmatrix} \quad \bar{B}_{lon} = \begin{pmatrix} B_{lon} \\ 0 \end{pmatrix}$$

The LQR controller designed using

$$Q = \text{diag} ([q_u, q_w, q_q, q_\theta, q_h, q_I])$$

$$R = \text{diag} ([r_{\delta_e}, r_{\delta_t}]).$$