Robot Dynamics: Fixed-wing UAS
Exercise 5: Fixed-wing Simulation and Control
SOLUTION

Thomas Stastny

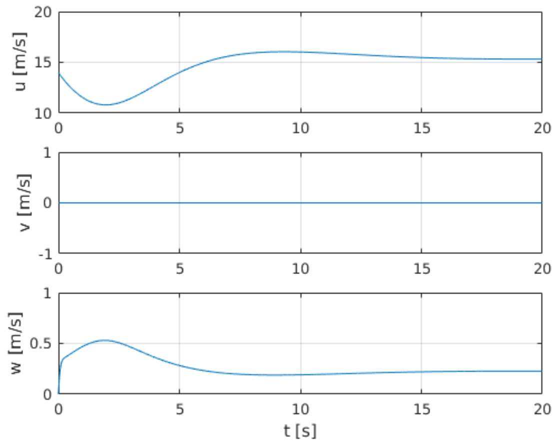
2020.12.09

1 Setup/Organization

N/A

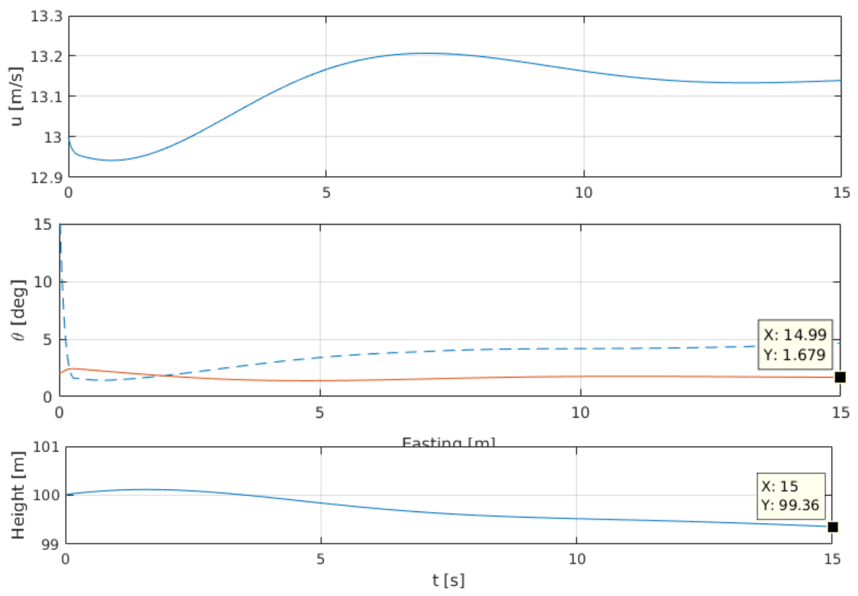
2 Model Dynamics / Open-loop Simulation

Task 1:
Initializing at $\boldsymbol{\delta} = \mathbf{0}$, $u = 14m/s$, $\theta = 5°$, the following open loop body velocities may be observed:



The u-body velocity stabilizes at $15\,\mathrm{m\,s^{-1}}$. The long, slow velocity oscillations over the first $10\,\mathrm{s}$–$15\,\mathrm{s}$ are respresentative of the aircraft's *phugoid* dynamics.

Task 2:
Setting $u_T = 0.4$ and $u_E = 0.025$, we can see the u-body velocity stabilize close to $13\,\mathrm{m\,s^{-1}}$, pitch angle θ to ca. $1.7°$, and h within $1\,\mathrm{m}$ of the starting altitude. Note the converged steady-state highly depends on the initial conditions from which the aircraft model is simulated – here initialized from $\theta = 2°$ and $u = 13\,\mathrm{m\,s^{-1}}$. This effect is mainly due to the nonlinearity of the model, where, though *dynamically stable* in this region – the final steady-state depends on the state-space through which the model travels over time. Further, we see that the aircraft is actually losing (very slowly) altitude, which tells us our trims are not exactly perfect – and we slightly violate our *level* flight condition.



3 Control

3.1 Attitude Stabilization

Task 1:
The cascaded approach shown in the lecture slides used the angular-rate transformation matrix to convert Tait-Bryan rates to body-axis rates, and subsequently used these transformed rates as references for rate control on the inner loop. Here, we see a similar structure - but somewhat simplified, neglecting the transformation, and removing the inner cascade in favor of a more standard PID structure – where attitude errors are penalized with a propotional gain, and rates are separately regulated to zero with dampers (D-gain). This is a popular approach for UAV flight control, as one may focus rate damping more on disturbance rejection, e.g. gusts/turbulence (not covered in this exercise), and primarily rely on the proportional attitude error regulation for actual attitude control. Note that consideration of the transformation between Tait-Bryan rates and body rates becomes more important as we get further outside of the normal operating flight regime – i.e. further outside of steady, level, flight conditions.

Task 2:

$$\begin{aligned}
u_E &= k_{P_\theta}\left(\theta_{\text{ref}} - \theta\right) + k_{I_\theta}\int\left(\theta_{\text{ref}} - \theta\right) - k_{D_\theta} q \\
u_A &= k_{P_\phi}\left(\phi_{\text{ref}} - \phi\right) + k_{I_\phi}\int\left(\phi_{\text{ref}} - \phi\right) - k_{D_\phi} p \\
u_R &= k_{\text{CT}}\left(\frac{g\tan\phi_{\text{ref}}}{V} - r\right)
\end{aligned}$$

Task 3: Rate damping.
See simulink model of solution.

Task 4: Pitch control.
See simulink model of solution.

Task 5: Roll control.

See simulink model of solution.

Task 6: Turn coordination.
See simulink model of solution.

3.2 Airspeed/Altitude Control

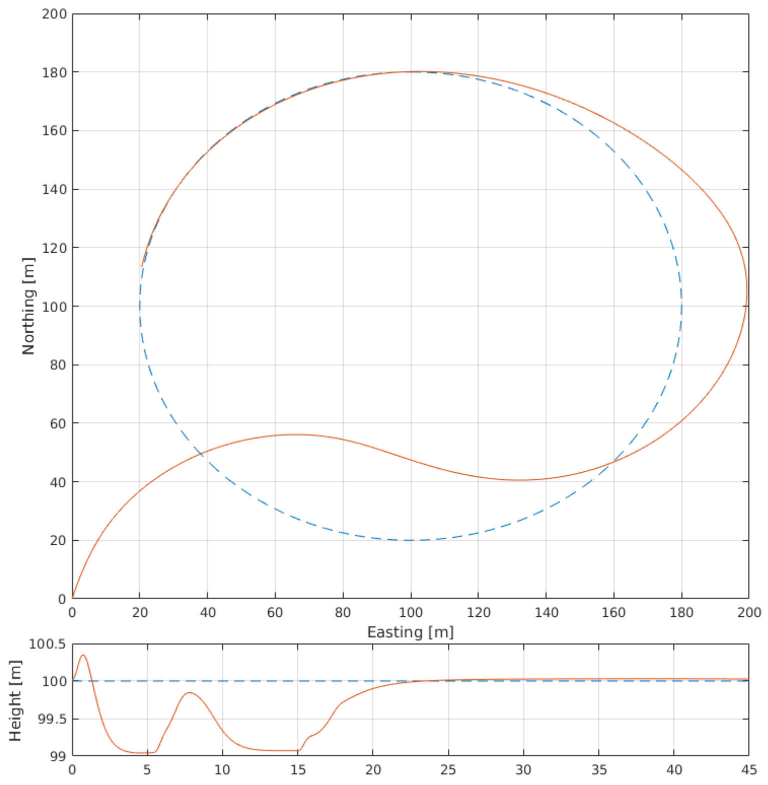
Task 1: TECS.
See simulink model of solution.

3.3 Lateral-directional Position Control

Task 1: L1 position control.
The following code snippet gives an example of the L1 guidance algorithm (see also the same code within the simulink model of the solution). Note the given example is only *one* implementation of the L1 algorithm. One could also (perhaps even more cleanly) use vector math, as oppose to calculating individual angles as done here.

```
function phi_ref = fcn(radius,dir,center,vG,ned,L1)

% calculate lateral-directional ground speed
ground_speed = norm(vG(1:2,1));

% vector from position to center of loiter
v_ned_to_center = center(1:2,1)-ned(1:2,1);
ned_to_center_bearing = atan2(v_ned_to_center(2),v_ned_to_center(1));

% calculate the distance from the aircraft to the circle
dist2circ = norm(v_ned_to_center)-radius;
abs_dist2circ = abs(dist2circ);

% check that L1 vector does not exceed reasonable bounds
if L1 >= (2*radius + dist2circ)
    L1 = 2*radius + dist2circ;
elseif L1 < abs_dist2circ
    L1 = abs_dist2circ;
end

% calculate L1 bearing (law of cosines)
cos_gam = (L1^2 + (dist2circ + radius)^2 - radius^2) ...
    / 2 / L1 / (dist2circ + radius);
if cos_gam > 1, cos_gam = 1; end
if cos_gam < -1, cos_gam = -1; end;
gam = acos(cos_gam);
L1_bearing = ned_to_center_bearing - dir * gam;
if (L1_bearing>pi), L1_bearing = L1_bearing - 2*pi; end;
if (L1_bearing<-pi), L1_bearing = L1_bearing + 2*pi; end;

% calculate error angle eta
ground_speed_bearing = atan2(vG(2,1), vG(1,1));
eta = L1_bearing - ground_speed_bearing;
if (eta>pi), eta = eta - 2*pi; end;
if (eta<-pi), eta = eta + 2*pi; end;

% limit eta to 90 degrees
if (eta>pi/2), eta = pi/2; end;
if (eta<-pi/2), eta = -pi/2; end;

% normal acceleration reference
aN_ref = 2*ground_speed^2*sin(eta)/L1;

% coordinated roll mapping
phi_ref = atan(aN_ref/9.81);
```

Task 2: L1 position control, part 2.
The following snapshot gives an example of the full automation following a loiter circle in a counter clockwise direction (please see the solution simulink model for details of the implementation to achieve this). Note this simulation includes approx. $6.4\,\mathrm{m\,s^{-1}}$ wind with a north-eastern vector.



General notes: Smaller L1 distances cause more aggressive control action. Note also that there is an inherent limitation in the algorithm, where a circle can not be tracked if the L1 distance is greater than the radius of the circle. One benefit that can be noticed in using the ground speed within the formulation is some inherent robustness to winds. However, another shortfall of this algorithm is the dependence on velocity for proper choice of L1 distance. – this of course will make tuning more challenging with high wind speeds, where throughout a given loiter circle, possibly large deviations in ground speed will be seen. One could gain schedule for this effect. However, there are more recent developments in line-of-sight guidance algorithms explicitly considering/finding solutions for these problems. For those interested, please see the following three papers:

1) L_2+, fixes velocity dependence and considers roll dynamics in stability analysis. This variant is currently implemented on standard *Pixhawk* autopilot firmware:

> Renwick Curry, Mariano Lizarraga, Bryant Mairs, and Gabriel Hugh Elkaim. “L_2+, an Improved Line of Sight Guidance Law for UAVs.” American Control Conference (ACC). 2013.

2) A three-dimensional line-of-sight law for path following with any path curvature:

> Namhoon Cho, Youdan Kim, and Sanghyuk Park. "Three-Dimensional Nonlinear Differential Geometric Path-Following Guidance Law." Journal of Guidance, Control, and Dynamics. Vol. 38(12). 2015.

3) An extension to the NDPFG (paper above) reformulating for explicit consideration of winds in the law and further considering the case of *excess-wind*, i.e. when the wind speed surpasses the airspeed:

> Luca Furieri, Thomas Stastny, Lorenzo Marconi, Roland Siegwart, and Igor Gilitschenski. "Gone with the Wind: Nonlinear Guidance for Small Fixed-wing Aircraft in Arbitrarily Strong Windfields." American Control Conference (ACC). 2017.

4) A further windy extension fixing some directional control issues in [3], developing a parallel airspeed reference compensation logic, and providing more extensive flight results/insights.

> Thomas Stastny and Roland Siegwart. "On Flying Backwards: Preventing Run-away of Small, Low-speed Fixed-wing UAVs in Strong Winds." International Conference on Intelligent Robots and Systems (IROS). 2019.