CROSSWIND CONTROLLER MODIFICATIONS

Kenneth Jensen

April 30th, 2010

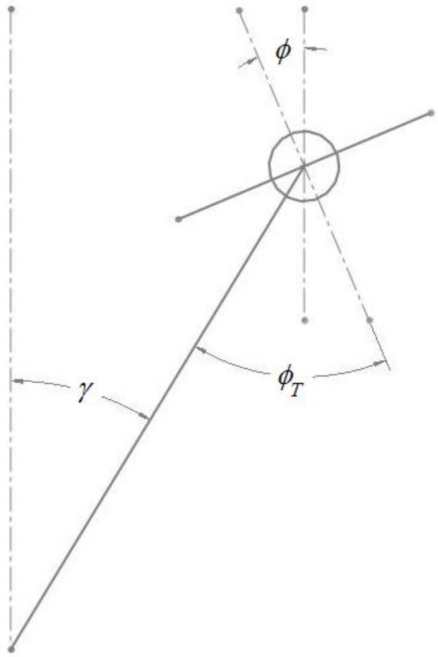

Figure 1. Diagram of crosswind kite control variables.

1. Introduction

The fundamental equation describing curvature control of a kite is:

$$\kappa = \frac{1}{l_T}\frac{C_L}{C_F}\sin\phi_T \quad \phi \ll 1 \tag{1}$$

where κ is the curvature, ϕ_T is the tether roll angle, and $C_F = \frac{m}{\frac{1}{2}\rho A l_T}$ is the dimensionless parameter describing the centrifugal force. After making the small ϕ_T approximation

$$\kappa = \frac{1}{l_T}\frac{C_L}{C_F}\phi_T \quad \phi_T \ll 1, \phi \ll 1 \tag{2}$$

Because $\phi \ll 1$, this equation is not strictly accurate for small ϕ_T or κ. Obviously, the smallest steady-state curvature possible on a sphere of radius l_T is $\kappa = 1/l_T$ and not zero as this equation would imply.

Taking into account the gravity and wind asymmetries,

$$\kappa \approx \frac{C_L}{l_T C_F}\left(1 - 2r_v\cos\psi_w + r_v^2\right)\left(\phi_T + \frac{mg_{\|}\sin\psi_g}{T}\right) \tag{3}$$

where $r_v = v_w/v_i$, ψ_w is the angle between the velocity vector and the wind, ψ_g is the angle around the projected circle of the flight path (defined to be zero on the right side of the loop and increasing CCW), and $g_{\|}$ and $g_{\perp}$ are the magnitudes of the gravity vector in and out of the flight plane.

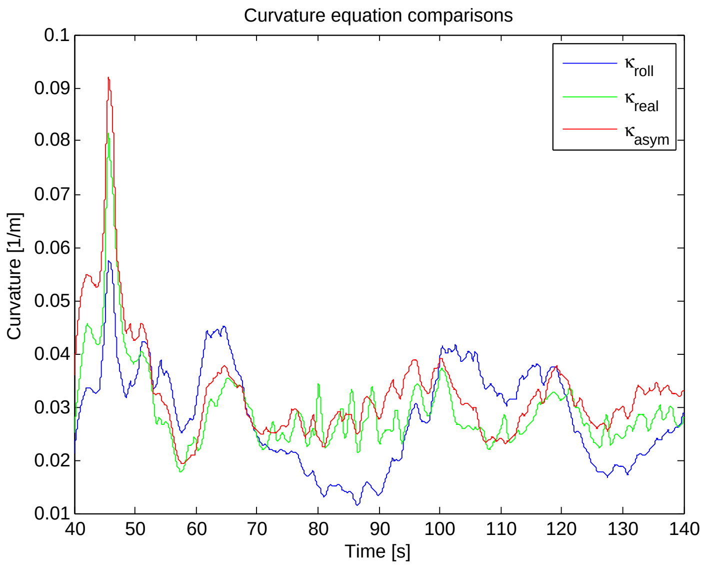

Figure 2. Comparision of equations for curvature.

Figure 2 shows a comparison of the true curvature to the curvatures calculated with Eq. 2 and Eq. 3. This figure demonstrates that Eq. 3 is an accurate description of curvature.

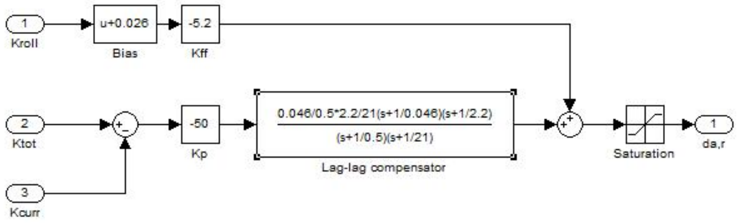

FIGURE 3. Basic structure of the modified predictor controller.

2. PREDICTOR MODIFICATIONS

The true curvature κ may be divided into two components: curvature from the tether roll, κ_{ϕ_T}, and curvature from the wind and gravity asymmetries.

$$\kappa \approx \kappa_{\phi_T} + \frac{g_{\parallel} \sin \psi_g}{{v_i}^2} - 2\kappa_{\phi_T} r_v \cos \psi_w \tag{4}$$

The tether roll curvature is what we directly control. However, the true curvature is what we most directly measure. Thus, we propose that the predictor should return two curvatures, κ and κ_{ϕ_T}. The feedforward term should only deal with the tether roll curvature, while the proportional term should deal with the true curvature (see Fig. 3).

Here we describe one possible method of modifying the predictor to account for the different curvatures. First, we change the list of possible curvatures to be a list of possible tether roll curvatures. We do not change the current curvature section. However, for the set curvature section, we calculate the wing's position assuming the set tether roll curvature and additional curvatures from wind and gravity. To do so, we calculate what ψ_g, ψ_w, and v_i would be if the wing continued at its current curvature for time $t = t_c + t_f/2$, where t_c is the predictor's current curvature time and t_f is the predictor's final curvature time. We then use Eq. 4 to calculate the modification to the curvature. This technique is a crude approximation, and its biggest flaw appears to be the difficulty of estimating the future velocity which figures prominently in gravity modification to curvature. The code for the modified predictor is in Appendix A.

Figure 4 shows the set tether roll curvature versus the true tether roll curvature. The dotted line down the center separates the modified predictor (left) from the original predictor (right). As expected, the modified predictor is actually controlling tether roll.

However, the true test of whether the modified predictor is performing better than the original predictor is in the curvature error signal. Ideally, the P term should not have to work as hard to control curvature if the feedforward term is performing as expected. Unfortunately, there does not seem to be a significant difference between the error signal for the modified versus originial predictor (see Fig. 5).

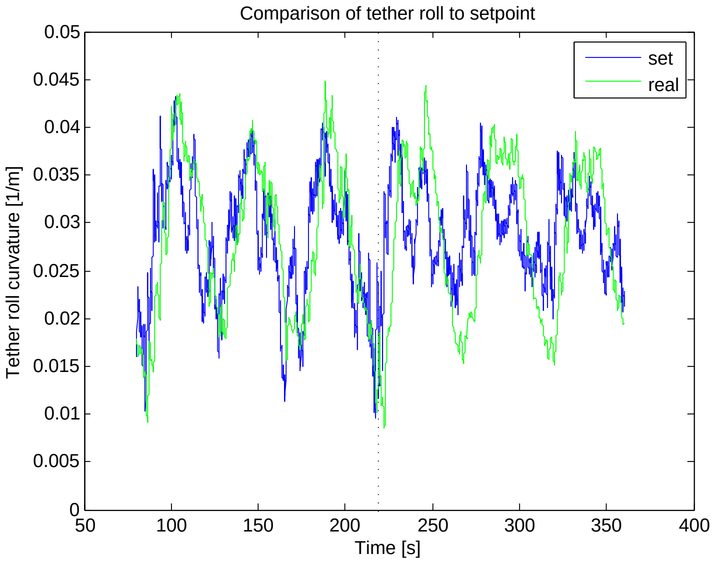

Figure 4. Tether roll curvature setpoint versus real tether roll curvature for the modified predictor (left of dotted line) and the original predictor (right of dotted line).

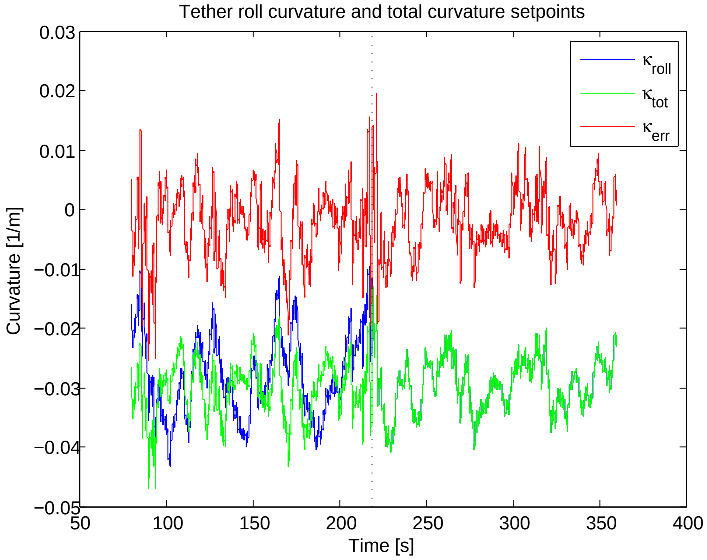

Figure 5. Tether roll and total curvature setpoints and the total curvature error for the modified predictor (left of dotted line) and the original predictor (right of dotted line).

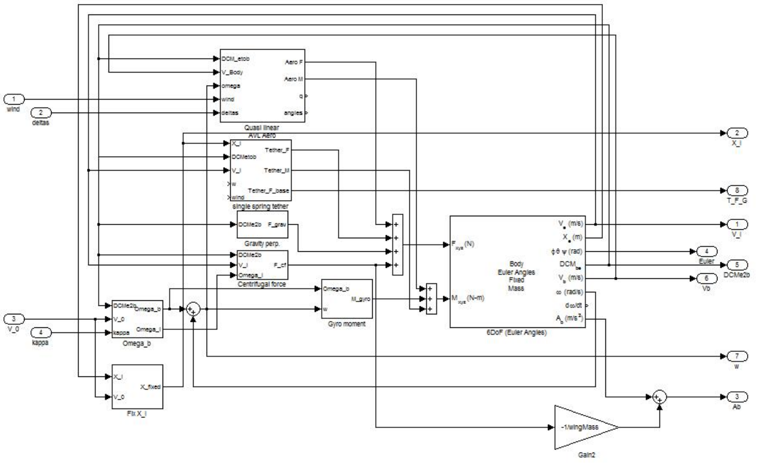

Figure 6. Rotating frame model.

3. Linearization in a rotating frame

To analyze the stability of the controller once the kite is flying a steady, circular path, we must linearize the problem in the reference frame that rotates with the kite at angular frequency, $\Omega_r = \kappa_0 v_0$. To do this, we first remove the wind and gravity asymmetries by only including the components of wind and gravity that are perpendicular to the flight plane. Next, we add the centrifugal force, $F_r = m\mathbf{v} \times \mathbf{\Omega_r}$ and gyroscopic moment $M_r = I\omega \times \mathbf{\Omega_r}$. Finally, to ensure that the aerodynamic and tether forces are correct, we correct the ω vector sent to the aerodynamics block to include both $\mathbf{\Omega_r}$ and ω in the rotating frame, and we correct the $\mathbf{x}$ vector sent to the tether block to discount the constant forward velocity. The final rotating frame model is shown in Fig. 6.

From the linearized model, it is possible to determine both the bias and "feedforward" gain in the controller. Fig. 7 shows the trimmed value of $\delta_{a,r}$ assuming a mixing ratio of 0.8 with various elevator deflections. For $\delta_f = -0.1$, the zero aileron deflection curvature is $\kappa_{\text{bias}} = 0.026\ \text{m}^{-1}$, which is the bias appearing on the feedforward path. The slope of the line here gives the feedforward gain as $K_{\text{ff}} \approx 5.2\ \text{rad} \cdot \text{m}$. The solid lines were calculated for a wind speed of 9 m/s. The dashed line was for a wind speed of 12 m/s demonstating that the trim values are insenstive to wind speed.

It it also interesting to calculate the feedforward gain analytically. To do so, we determine what flap deflections result in a given tether roll. From moment balance,

$$\phi_T - \phi_{T0} = \frac{b}{C_L l_b}\left(C_{l_{\delta_a}} + mC_{l_{\delta_r}}\right)\delta_a \tag{5}$$

(Here, m is the mixing ratio between the the rudder and aileron signals.) Thus, the feedforward term should be (assuming a flap effectiveness of 1):

$$K_{\text{ff}} = \frac{\delta_a}{\kappa_{\phi_T}} = \frac{C_F l_b l_T}{b\left(C_{l_{\delta_a}} + mC_{l_{\delta_r}}\right)} \approx 7.7 \tag{6}$$

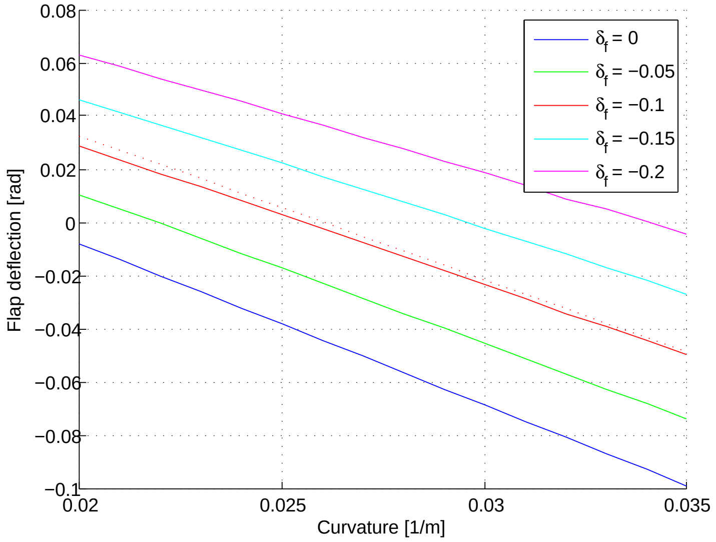

Figure 7. Trimming model in rotating frame

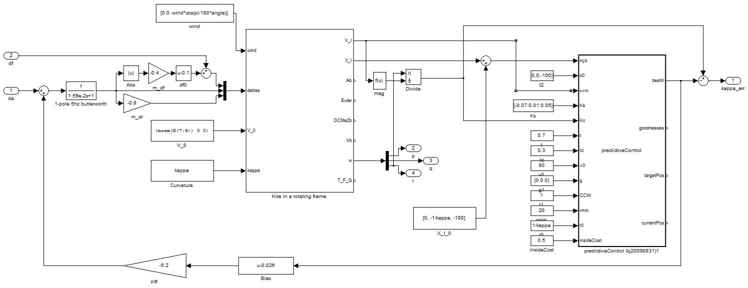

Figure 8. Linearizing around the predictor.

It is interesting and fortunate that this equation does not depend on C_L. The value determined here is slightly larger than the true value because we are not accounting for effects such as the gyroscopic moment and the moments from some of the other stability derivatives such as C_{l_r}.

4. Linearization around the predictor

The predictive controller has proven itself to be an excellent and robust method of controlling the wing's path. However, it does make it difficult to analyze the controller from a classical controls perspective. To sidestep this difficulty, we linearize the model *and* predictor together. Note that we also include the gyro-based estimation of curvature in this linearization. (With the real curvature in the loop, it is possible to push gains up to unrealistic levels.) The Simulink model that demostrates our method for this is shown in Fig. 8.

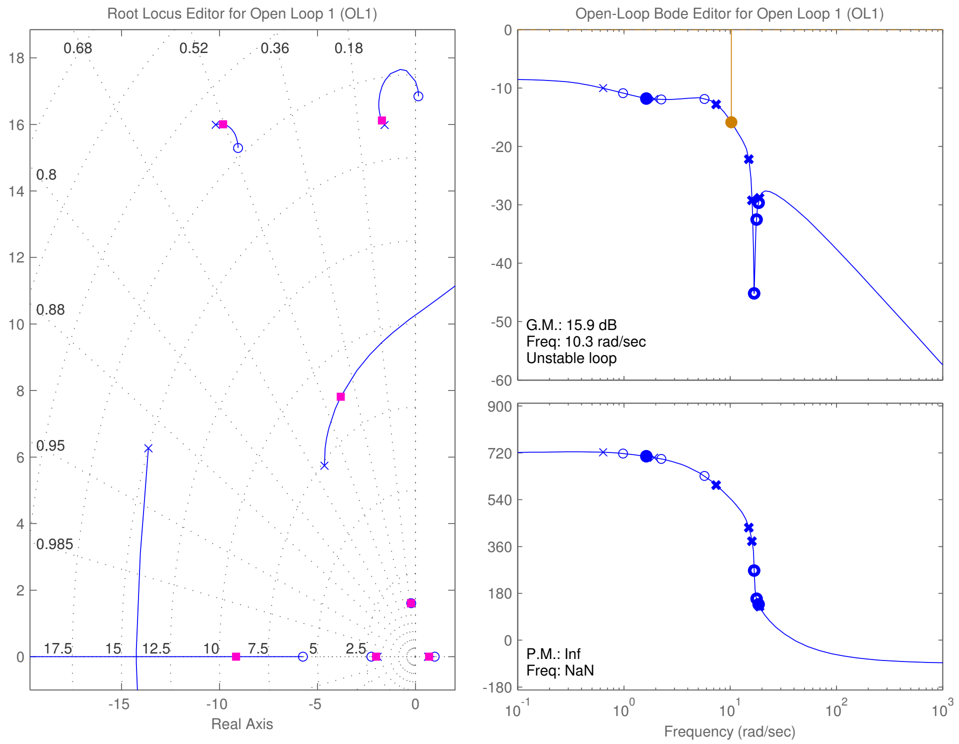

FIGURE 9. Root locus with $K_{\text{ff}} = 3$, $v_w = 12$ m/s, $b_x = -6$ cm

Figure 9 shows a root locus plot for the linearized model and predictor with a feedforward gain of $K_{\text{ff}} = 3$. (We chose a lower feedforward gain than 5.2 because this is closer to what has worked well in the past.) The plant is unstable with a slow spiral divergence. However, the predictor appears to be able to compensate for this slow instability. The higher frequency poles are mostly what we are concerned with.

We have multiple goals in adding a compensator to this control system. Obviously, we first want the system to be stable with a large gain and phase margin. However, we also want the steady-state curvature error to be as small as possible, so the wing is flying appropriately sized loops. Finally, it appears to be important to maintain a high bandwidth (relative to the frequency of the loops) for the proper functioning of the predictor.

Moreover, the design requirements for the compensator change with windspeed. For low wind speeds, $v_w < 6$ m/s, the bandwidth of the control system is usually more than sufficient to maintain a circular path. The main concern is increasing robustness to uncertainties in control effectiveness. Thus, we propose using a lag-lag compensator that increases the low frequency gains.

$$C(s) = 5 \times \frac{(1 + 1.7s)(1 + 1.7s)}{(1 + 17s)(1 + 17s)} \tag{7}$$

This compensator performs well for both the 6 cm back and 12 cm back bridle cases.

For moderate wind speeds, 6 m/s $< v_w <$ 10 m/s, we want to increase the low-frequency gain while maintaining sufficient bandwidth. To do so, we chose a lag-lead controller.

$$C_{6\mathrm{cm}}(s) = 45 \times \frac{(1 + 0.37s)(1 + 0.31s)}{(1 + 28s)(1 + 0.04s)} \tag{8}$$

$$C_{12\mathrm{cm}}(s) = 42 \times \frac{(1 + 0.38s)(1 + 0.33s)}{(1 + 28s)(1 + 0.018s)} \tag{9}$$

For high wind speeds, $v_w > 10$ m/s, we again want to increase the low-frequency gains while maintaining sufficient bandwidth. Again, we chose a lag-lead controller.

$$C_{6\mathrm{cm}}(s) = 11.9 \times \frac{(1 + 0.35s)(1 + 0.27s)}{(1 + 7s)(1 + 0.013s)} \tag{10}$$

$$C_{12\mathrm{cm}}(s) = 11.9 \times \frac{(1 + 0.4s)(1 + 0.36s)}{(1 + 7s)(1 + 0.067s)} \tag{11}$$

Unfortunately, all of these controllers are currently “hand-tuned.” An important project for future crosswind controllers is to automate and optimize the process of choosing compensators.

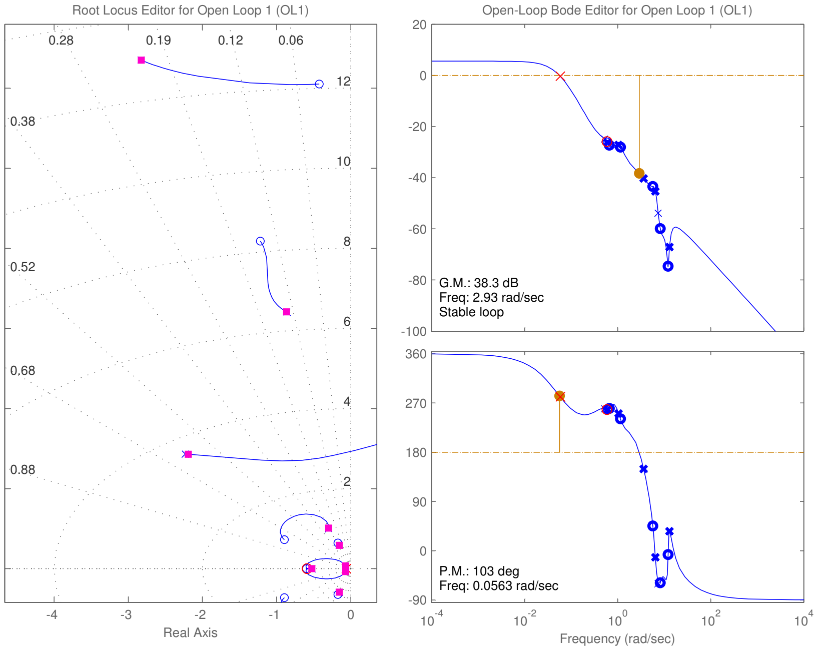

Figure 10. $K_{\mathrm{ff}} = 3$, $v_w = 4$ m/s, $b_x = -6$ cm, DVL

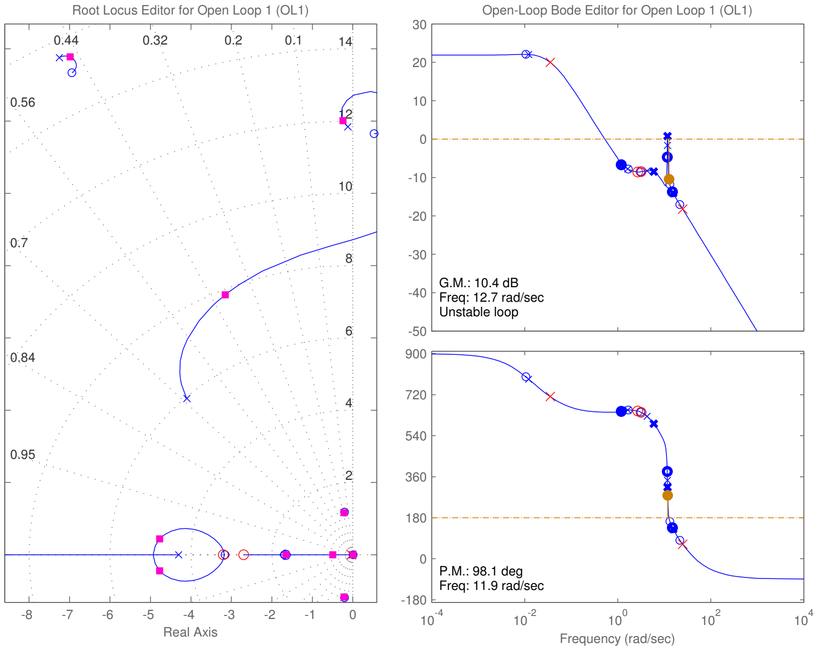

Figure 11. $K_{\mathrm{ff}} = 3$, $v_w = 8$ m/s, $b_x = -6$ cm, DVL

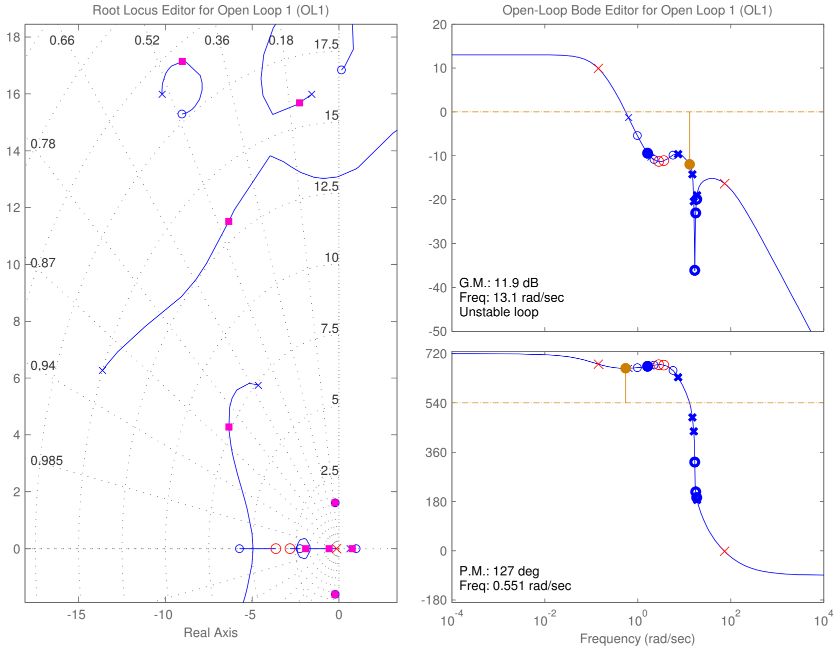

Figure 12. $K_{\text{ff}} = 3$, $v_w = 12$ m/s, $b_x = -6$ cm, DVL

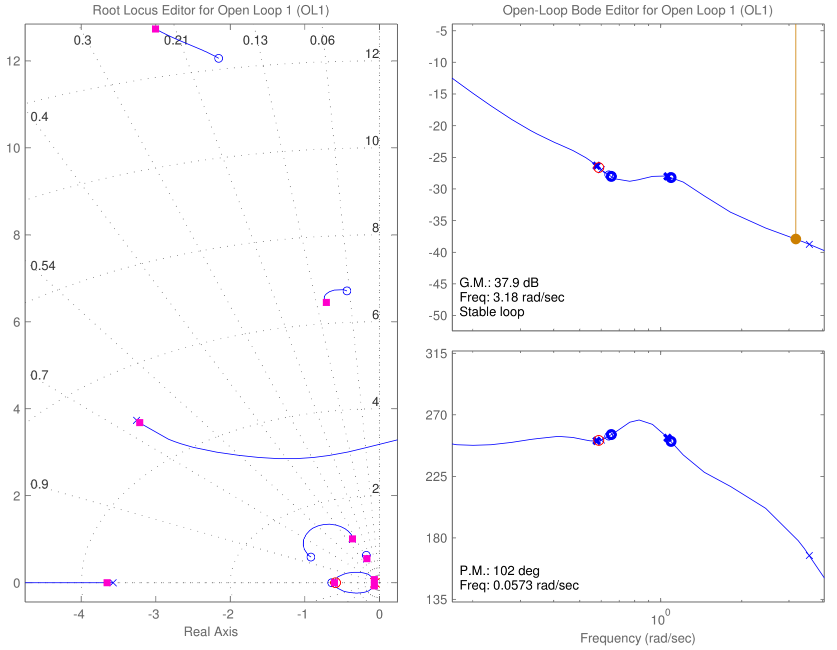

Figure 13. $K_{\text{ff}} = 3$, $v_w = 4$ m/s, $b_x = -12$ cm, DVL

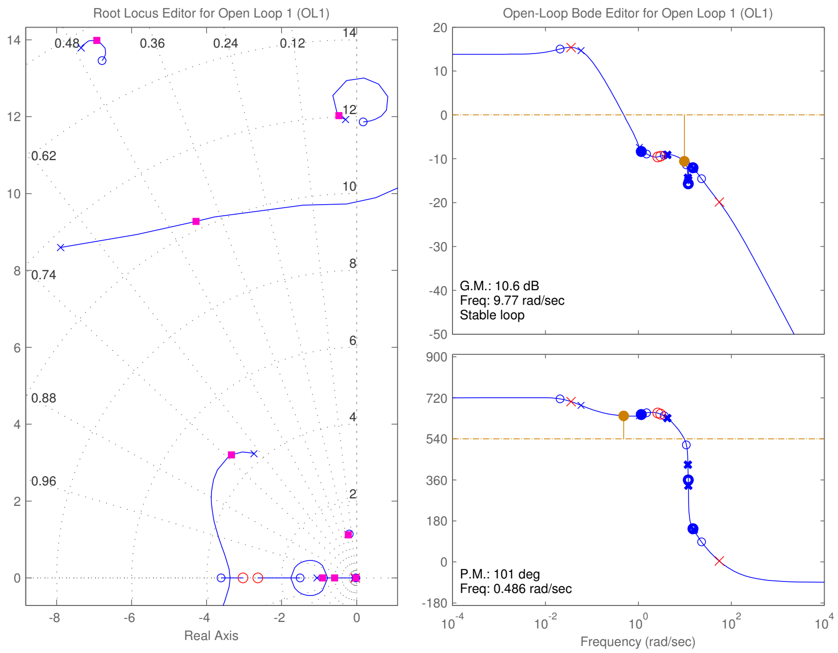

Figure 14. $K_{\mathrm{ff}} = 3$, $v_w = 8$ m/s, $b_x = -12$ cm, DVL

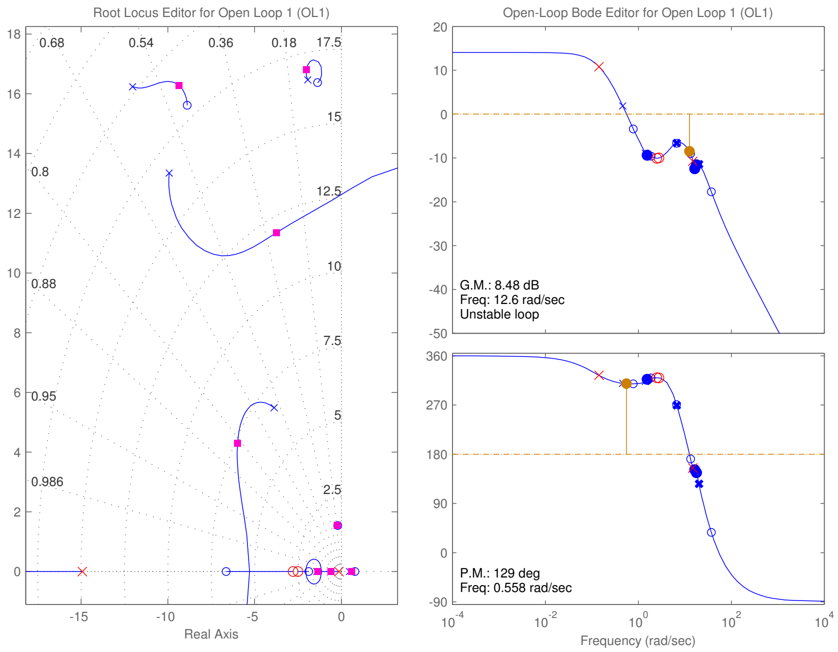

Figure 15. $K_{\mathrm{ff}} = 3$, $v_w = 12$ m/s, $b_x = -12$ cm, DVL

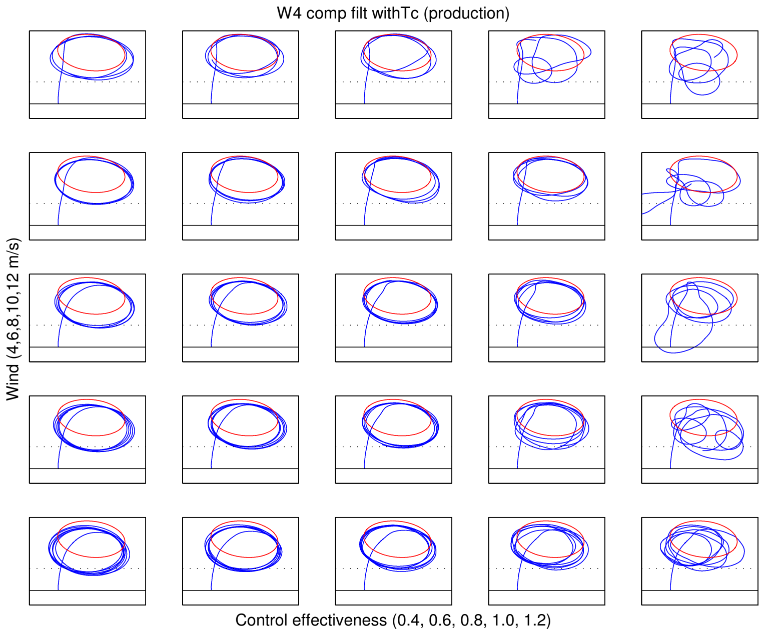

Figure 16. Standard controller under different wind speeds and control effectivenesses.

We have compared the various controllers using a test suite that runs the controllers under different starting conditions, wind speeds, and control effectivenesses.

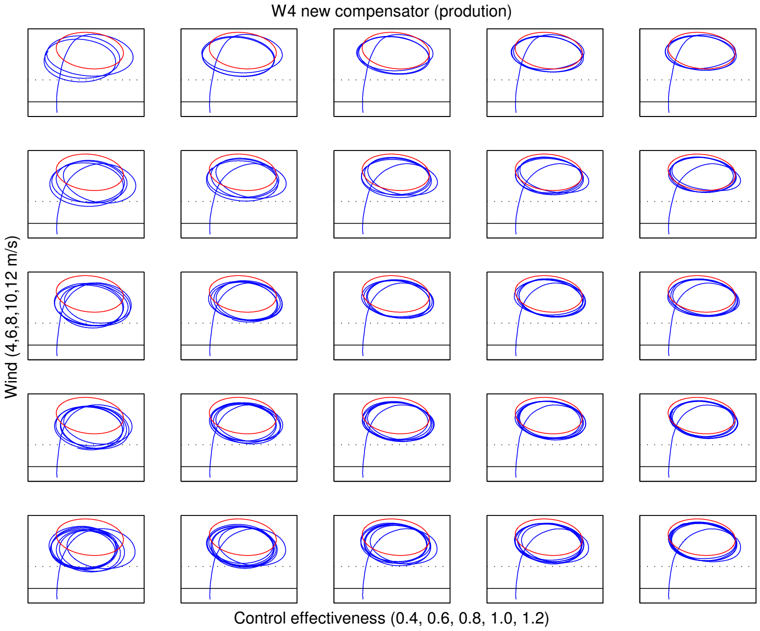

Figure 17. Controller with 12 cm bridle compensator under different wind speeds and control effectivenesses.

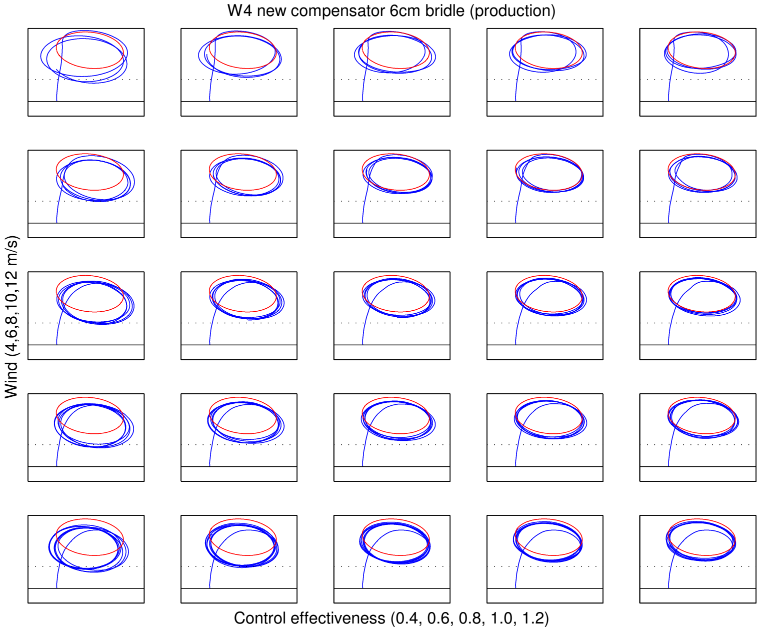

Figure 18. Controller with 6 cm bridle compensator under different wind speeds and control effectivenesses.

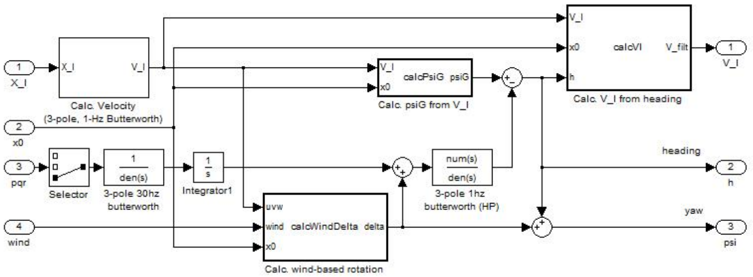

Figure 19. Diagram of the complimentary velocity filter

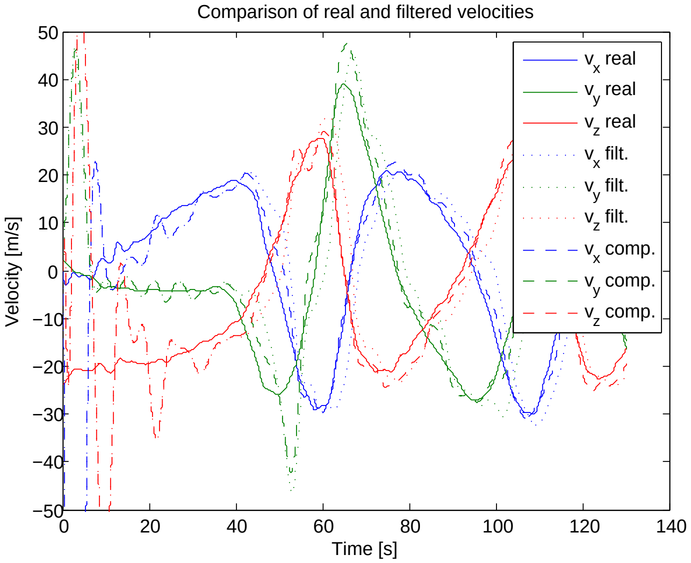

Figure 20. Comparison of real and filtered velocities.

5. Velocity (heading) filter

The current controller uses an estimation of the kite's velocity vector to predict where the kite will be in the future. The velocity vector estimation is simply a derivative of the position, which is measured through the ground line angle sensor (GLAS). The velocity signal is fairly noisy and is low-pass filtered at 1 Hz to yield a useful signal.

To improve the high frequency characteristics of the heading, we have combined the low frequency heading information from the GLAS-based velocity vector with the high frequency heading information from the yaw rate gyro. The structure of this complimentary is shown in Fig. 19. Note that the heading will be different than the yaw due to the asymmetry of the wind around the loop (see Fig. 21).

Figure 20 shows a comparison of the true inertial velocity with that of a 1 Hz filtered signal, and the complimentary filtered signal. As expected, the complimentary filtered signal is much closer to true velocity than the 1 Hz filtered signal.

6. Curvature estimation

The current controller is based around the concept of curvature. The predictor calculates a curvature set-point and the controller then attempts to minimize the error between the measured curvature and the set-point using a traditional linear controller. Unfortunately, we do not measure curvature directly, and thus we must estimate it using a variety of sensors.

Curvature is defined as

$$\kappa = \frac{|\mathbf{a} \times \mathbf{v}|}{|\mathbf{v}|^3} = \frac{\sqrt{(a_z v_y - a_y v_z)^2 + (a_x v_z - a_z v_x)^2 + (a_y v_x - a_x v_y)^2}}{({v_x}^2 + {v_y}^2 + {v_z}^2)^{3/2}} \tag{12}$$

This is equivalent to

$$\kappa = \frac{a_\perp}{v^2} \tag{13}$$

where $a_\perp$ is the magnitude of the acceleration perpendicular to the velocity vector in the plane of curvature. This is also equivalent to

$$\kappa = \frac{\dot{h}}{v} \tag{14}$$

where h is the heading angle of the velocity vector in the plane of curvature.

We have developed numerous ways of estimating curvature. One of the simplest means of estimating curvature, based on the gyros, is

$$\kappa_{\text{gyro}} \approx \frac{r}{v} \tag{15}$$

This estimate assumes small sideslip, small roll angle, and a loop that is centered directly downwind. This estimate has the advantages that it is incredibly simple (it only depends on the GLAS and the z gyro!), and also that it will never be too far off from the real curvature.

Here are a few simple corrections to Eq. 15 that take into account the asymmetry of the wind, position of the flight plane, and non-zero sideslip:

$$\kappa_{\text{gyro}} = \frac{r}{v}\left(1 - \frac{w_\|}{v}\cos\psi_g\right) \tag{16}$$

$$\kappa_{\text{gyro}} = \frac{r}{v}\left(1 - \frac{w_\|(\mathbf{x}_0)}{v}\cos\psi_g\right) \tag{17}$$

$$\kappa_{\text{gyro}} = \frac{r}{v}\left(1 - \frac{w_\|(\mathbf{x}_0)}{v}\cos\psi_g\right) + \frac{\dot{\beta}}{v} \tag{18}$$

Each equation is successively more accurate. However, the last two rely on additional measurements. The first correction, Eq. 16, is our standard definition for gyro-based curvature. A comparison of the various gyro-based curvatures is shown in Fig. 22.

Another simple means of estimating curvature is based on the accelerometers.

$$\kappa_{\text{acc}} = \frac{a_\perp}{v^2} \approx \frac{a_{y,z}\cos\Delta - a_x \sin\Delta}{v^2} \approx \frac{a_y}{v^2} \tag{19}$$

where $\Delta \approx w_\|/v \sin\psi_g$ is the angle between $\hat{\mathbf{x}}_b$ and $\mathbf{v}$.

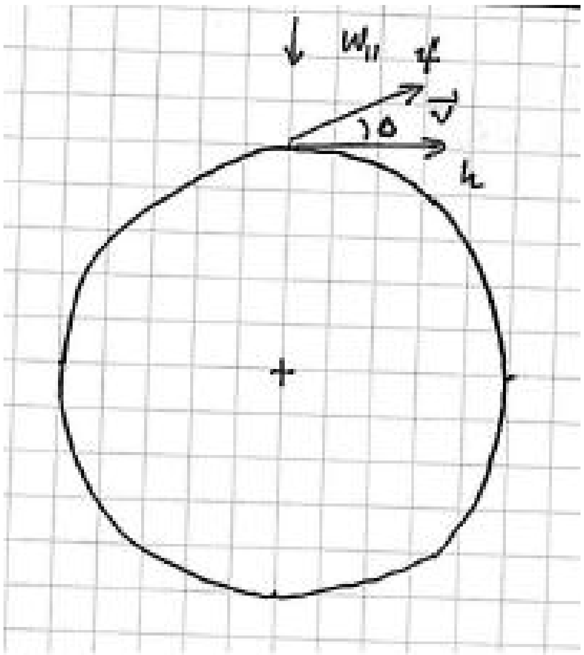

Figure 21. Difference between heading and yaw

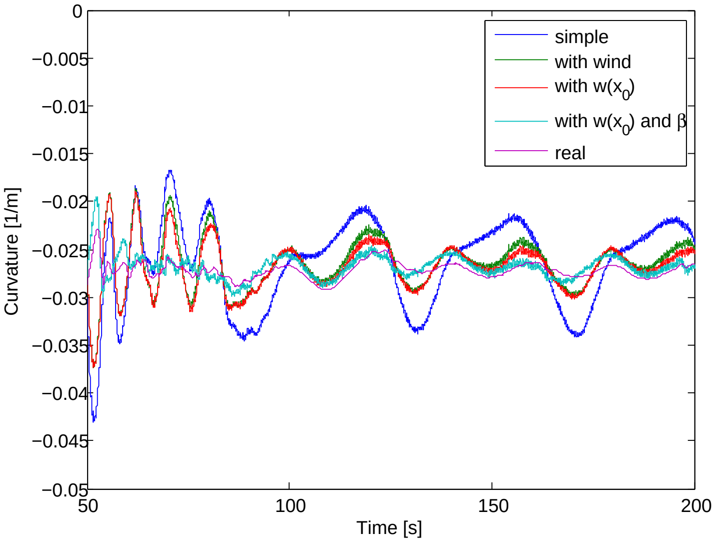

Figure 22. Comparison of κ_{gyro} approximations. (Note: these are with perfect velocity measurements.)

6.1. **Simple estimator: cartesian coordinates in the flight plane.** This curvature estimator was meant to be a simple way of combining information from the gyro-based and accelerometer-based curvature estimates. The state vector includes the wing's projected x

and y coordinates in the flight plane, the heading in the flight plane ψ, the curvature κ, the magnitude of the velocity, v, and the magnitude of the tangential acceleration, a:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ \kappa \\ v \\ a \end{bmatrix} \tag{20}$$

The update equation for this state vector is:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \begin{bmatrix} \sin(\psi)v \\ -\cos(\psi)v \\ \kappa v \\ 0 \\ a \\ 0 \end{bmatrix} \Delta T \tag{21}$$

From the update equation, we derive the Jacobian:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} 0 & 0 & \cos(\psi)v & 0 & \sin(\psi) & 0 \\ 0 & 0 & \sin(\psi)v & 0 & -\cos(\psi) & 0 \\ 0 & 0 & 0 & v & \kappa & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

Finally, we take x, y, ψ, $r = \kappa v$, and $a_y = \kappa v^2$ as the measurements:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x & y & \psi & \kappa v & \kappa v^2 \end{bmatrix}^T \tag{23}$$

This gives the following measurement matrix:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v & \kappa & 0 \\ 0 & 0 & 0 & v^2 & 2v\kappa & 0 \end{bmatrix} \tag{24}$$

Although we have not explored this thoroughly, it may be useful to incorporate the substantial differences between heading and yaw in the filter. Here are the measurement matrices that outline how this may be done.

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} x & y & \psi & \dfrac{\kappa v}{1 - \frac{w_\parallel}{v}\cos(\psi)} & \kappa v^2 \end{bmatrix}^T \tag{25}$$

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{\kappa w_\parallel \sin\psi}{(1-\frac{w_\parallel}{v}\cos\psi)^2} & \frac{v}{1-\frac{w_\parallel}{v}\cos\psi} & \frac{\kappa}{1-\frac{w_\parallel}{v}\cos\psi} - \frac{\kappa\frac{w_\parallel}{v}\cos\psi}{\left(1-\frac{w_\parallel}{v}\cos\psi\right)^2} & 0 \\ 0 & 0 & 0 & v^2 & 2v\kappa & 0 \end{bmatrix} \tag{26}$$

6.2. **Simple estimator: spherical coordinates.** In an attempt to improve the "off circle" behavior of the previous estimator, we converted the estimator to spherical coordinates.

$$\mathbf{x} = \begin{bmatrix} \theta \\ \phi \\ \psi \\ \kappa \\ v \\ a \end{bmatrix} \tag{27}$$

The update equation for this state vector is:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \begin{bmatrix} \frac{\cos(\psi)v}{l_T} \\ \frac{-\sin(\psi)v}{l_T\sin(\theta)} \\ \kappa v \\ 0 \\ a \\ 0 \end{bmatrix} \Delta T \tag{28}$$

From the update equation, we derive the Jacobian:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} 0 & 0 & -\sin(\psi)v/l_T & 0 & \cos(\psi)/l_T & 0 \\ \frac{\sin(\psi)v\cos(\theta)}{l_T\sin^2(\theta)} & 0 & \frac{-\cos(\psi)v}{l_T\sin(\theta)} & 0 & \frac{-\sin(\psi)}{l_T\sin(\theta)} & 0 \\ 0 & 0 & 0 & v & \kappa & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{29}$$

Finally, we take θ, ϕ, ψ, $r = \kappa v$, and $a_y = \kappa v^2$ as the measurements:

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} \theta & \phi & \psi & \kappa v & \kappa v^2 \end{bmatrix}^T \tag{30}$$

This gives the following measurement matrix:

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & v & \kappa & 0 \\ 0 & 0 & 0 & v^2 & 2v\kappa & 0 \end{bmatrix} \tag{31}$$

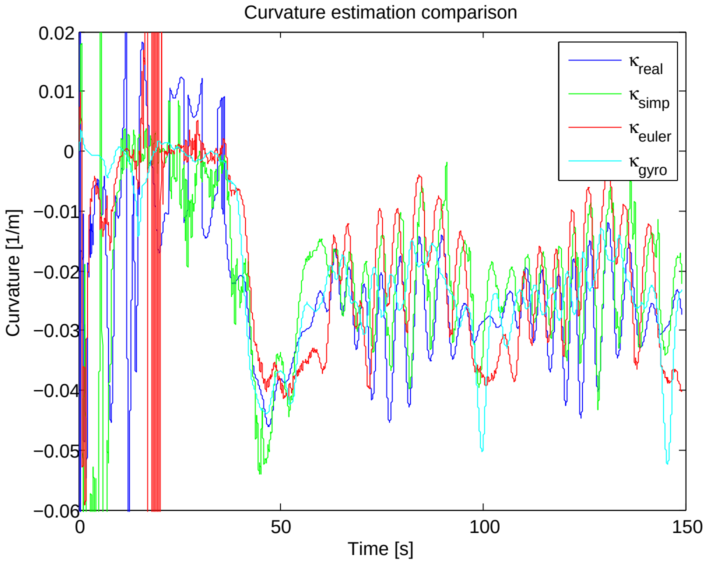

Figure 23. Comparison of curvature estimation techniques.

This estimator would also benefit from the incorpation of the difference between heading and yaw.

6.3. **Full state estimation.** The curvature may also be estimated using the definition of curvature (Eq. 12) and the values of a_b and v_b determined from the full state estimator. The full state estimator is described in detail in *Makani VTOL estimator*. This method was tried, but as expected, it performed worse than the simpler Kalman estimators for curvature.

6.4. **Comparison of curvature estimation techniques.** The simple curvature estimator performs better than both the euler-angle-based curvature estimator and the gyro-based curvature estimator. Of course, the fidelity of the curvature estimation also depends of the particular controller used.

Due to the simplicity of the gyro-based methods and the ability to fine tune the controller to be stable assuming a gyro-based curvature, my preference is to keep using the gyro-based curvature, but with the addition of the wind effects. If future versions of the simple estimator prove to be as robust, we can switch over.

7. Tether roll estimation

Equation 2 suggests that estimating the tether roll angle would be extremely useful in estimating curvature. At the very least, the high-frequency components of tether roll angle should tell us the high-frequency components of curvature. The high-frequency behavior of the tether roll angle, ϕ_T, may be estimated from the rate gyros. Specifically,

$$HPF(\phi_T) \approx HPF(\int dt(p + \theta r)) \tag{32}$$

to be continued...

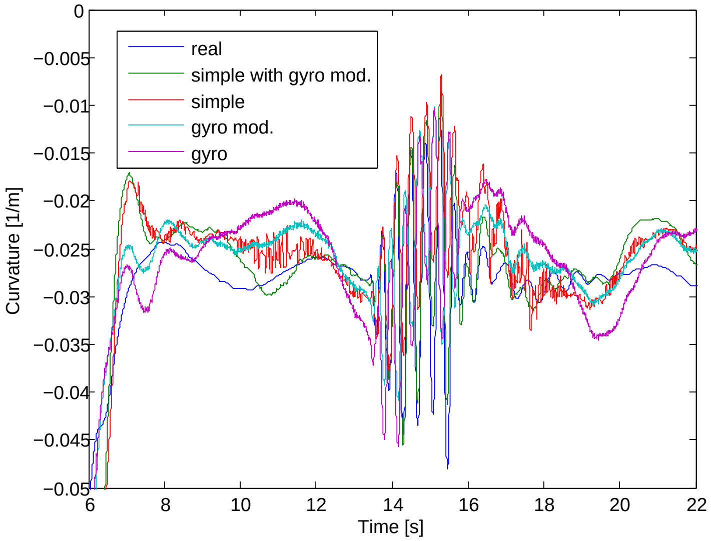

Figure 24. Comparison of various gyro-based curvature estimation schemes.

8. Coordinated turning

Many of our techniques for estimating curvature rely on the assumption of zero sideslip. To minimize sideslip and its effects on curvature estimation, we should attempt to perform coordinated turns. In a coordinated turn ($\dot{\phi} = 0$, $\dot{\theta} = 0$), the following relations hold:

$$p = -\sin(\theta)\dot{\psi} \tag{33}$$

$$q = \sin(\phi)\cos(\theta)\dot{\psi} \tag{34}$$

$$r = \cos(\phi)\cos(\theta)\dot{\psi} \tag{35}$$

to be continued...

9. Appendix A

```
%%Predictive controller: Looks ahead some distance, assuming the kite is commanded to a range of
%curvatures. The curvature which best flies the kite in the desired direction
%is then selected for the curvature command given out. The controller also
%puts out a roll command, in the case that roll control is used.
%
%Usage:
%[bestK,goodnesses,targetPos,currentPos] = predictiveControl(xyz,x0,uvw,Ks,Kc,t,tc,g,CCW,rmin,r0
%
%Inputs:
%xyz: the xyz position of the kite
%x0: the center of the circle to fly about. This should be a vector which points at the center o
%uvw: the current velocity of the kite in the inertial frame
%Ks: a list of curvatures to try. With parabolic fit, 5 curvatures will work, while without para
%Kc: the current curvature the kite is flying at
%g: gravity vector
```

```
%CCW: 1 or -1. Fly counter-clockwise loops if 1
%rmin: the minimum turning radius of the kite. Adjusts the level of aileron deflection versus positiona
%r0: the radius of the circle to fly about.
%insideCost: Amount to discount goodness of inside of circle
%wind: Current wind speed, slowly filtered!
%kitespeed: mean kite speed
function [Kroll, Ktot, goodnesses, targetPos, currentPos] = predictiveControl(xyz, x0, uvw, Ks, Kc, t,


%%
% Project onto plane
ex = -x0/norm(x0);        % x axis in projected frame
ey = cross([0 0 1], ex)/norm(cross([0 0 1], ex));  % y axis in projected frame
ez = cross(ex, ey);       % z axis in projected frame

xyz_p = [dot(xyz,ex), dot(xyz,ey), dot(xyz,ez)];
uvw_p = [dot(uvw,ex), dot(uvw,ey), dot(uvw,ez)];
g_p = [dot(g,ex), dot(g,ey), dot(g,ez)];
windspeed = norm(wind);

%%
% Calc psiW, psiG, vKf
vKc = max(sqrt(uvw_p(2)^2 + uvw_p(3)^2),1);

tc0 = tc+t/2;
DCMbtoec = (1/max(sqrt(uvw_p(2)^2 + uvw_p(3)^2),1e-5))*[uvw_p(2), -uvw_p(3); uvw_p(3), uvw_p(2)];
distc = vKc*tc0;
Rc = max(-1e5, min(1/Kc, 1e5));
thetac = distc/Rc;

xcb = Rc*sin(thetac);
ycb = Rc*(1-cos(thetac));
ucb = vKc*cos(thetac);
vcb = vKc*sin(thetac);

uce = DCMbtoec(1,1)*ucb + DCMbtoec(1,2)*vcb;
vce = DCMbtoec(2,1)*ucb + DCMbtoec(2,2)*vcb;
xce = DCMbtoec(1,1)*xcb + DCMbtoec(1,2)*ycb + xyz_p(2);
yce = DCMbtoec(2,1)*xcb + DCMbtoec(2,2)*ycb + xyz_p(3);

% Note: The 0.2 and the linear dependence on altitude are arbitrary
vKf = max(10, vKc + 0.2*kitespeed*(yce-xyz_p(3))/(2*r0));

g_para = sqrt(g_p(2)^2+g_p(3)^2);
uvw2 = (uce*ey+vce*ez);
psiW = acos(dot(uvw2/max(1e-5,norm(uvw2)), wind/max(1e-5,norm(wind)) ));
psiG = -atan2(uce, vce) - 2*pi;
```

```
%%
% Apply current curvature for tc
vKc = max(sqrt(uvw_p(2)^2 + uvw_p(3)^2),1);

tc = min(tc, tc/(vKc/v0)^2);
DCMbtoec = (1/max(sqrt(uvw_p(2)^2 + uvw_p(3)^2),1e-5))*[uvw_p(2), -uvw_p(3); uvw_p(3), uvw_p(2)]
distc = vKc*tc;
Rc = max(-1e5, min(1/Kc, 1e5));
thetac = distc/Rc;

xcb = Rc*sin(thetac);
ycb = Rc*(1-cos(thetac));
ucb = vKc*cos(thetac);
vcb = vKc*sin(thetac);

uce = DCMbtoec(1,1)*ucb + DCMbtoec(1,2)*vcb;
vce = DCMbtoec(2,1)*ucb + DCMbtoec(2,2)*vcb;
xce = DCMbtoec(1,1)*xcb + DCMbtoec(1,2)*ycb + xyz_p(2);
yce = DCMbtoec(2,1)*xcb + DCMbtoec(2,2)*ycb + xyz_p(3);


%%
% Apply new curvature for t
t = min(t, t/(vKf/v0)^2);
DCMbtoe = (1/max(sqrt(uce.^2 + vce.^2),1e-5))*[uce, -vce; vce, uce];
dist = vKf*t;
Ks2 = Ks + CCW*g_para*sin(psiG)/vKf^2 - 2*windspeed/vKf*cos(psiW)*Ks;
Rs = max(-1e5, min(1./Ks2, 1e5));
theta = dist./Rs;

xb = Rs.*sin(theta);
yb = Rs.*(1-cos(theta));
ub = vKf.*cos(theta);
vb = vKf.*sin(theta);

ue = DCMbtoe(1,1)*ub + DCMbtoe(1,2)*vb;
ve = DCMbtoe(2,1)*ub + DCMbtoe(2,2)*vb;
xe = DCMbtoe(1,1)*xb + DCMbtoe(1,2)*yb + xce;
ye = DCMbtoe(2,1)*xb + DCMbtoe(2,2)*yb + yce;



%%
% Calculate goodness
xt = [zeros(size(xe)), xe, ye]’;
ut = [zeros(size(xe)), ue, ve]’;


%% returns the best direction for the kite to be pointing at any given location.
```

```
% x = kite location
% r0 = radius to attempt to follow
% rmin = tightest turning radius to request of the kite
% CCW = clockwise or counter-clockwise
r = sqrt(sum(xt.^2,1));
radialDirOut = xt./(ones(3,1)*r);
pathDir = CCW*cross([1 0 0]'*ones(1,size(xt,2)),radialDirOut);
pathError = r-r0;

%for un-biased inner and outer radii
c1 = abs(pathError)<rmin;
tangComp = c1.*(cos(0.5*pi*pathError/rmin));
radialComp = c1.*(-sin(0.5*pi*pathError/rmin))+(1-c1).*(-pathError./max(1e-5,abs(pathError)));

%the best direction to be pointing for the given location
bestDir = (ones(3,1)*radialComp).*radialDirOut + (ones(3,1)*tangComp).*pathDir;

% Dot preferred directions with predicted directions
goodnesses = (dot(bestDir,ut)./max(1e-5,sqrt(dot(ut,ut))))';
insideCircleBadness = (r<r0).*(sin((pi/2)*r/r0)-1)*insideCost;
goodnesses = goodnesses + insideCircleBadness';

%%
% Choose best path
[maxGood, iMaxGood] = max(goodnesses);
bestK = Ks(iMaxGood);
if length(goodnesses)>2
    iMaxGood = max(2, min(iMaxGood, length(goodnesses)-1));
    P = polyfit(Ks([iMaxGood-1, iMaxGood, iMaxGood+1]), ...
                goodnesses([iMaxGood-1, iMaxGood, iMaxGood+1]), 2);
    if P(1)<0
        bestK = -P(2)/(2*P(1));
    end;
end;

Kroll = max(min(Ks), min(max(Ks), bestK));
Ktot = Kroll + CCW*g_para*sin(psiG)/vKf^2 - 2*windspeed/vKf*cos(psiW)*Kroll;
targetPos = [xe(iMaxGood), ye(iMaxGood)]';
currentPos = xyz_p(2:3)';
```