The Pyramid

By Tallak Tveide [Kitemill AS, tt@kitemill.com] with the help of Roderick Read and Oliver Tulloch [Windswept and Interesting Ltd, rod.read@windswept-and-interesting.co.uk and oliver.tulloch@windswept-and-interesting.co.uk respectively]

Introduction

The pyramid is an airborne wind energy concept that utilizes [initially] three efficient kites in a TRPT [Torsional Rotary Power Transfer] setup. The shaft is as simple as possible with only one tether per kite and a triangular bridle at the kites.

The benefits of this design compared to other AWE concepts are

- Low mass at the kite
- No gravitational speed change
- Constant power output
- No lifter kite
- Sane and defined takeoff and landing concept
- Works with shorter tethers so that tether drag will not kill the performance
- No reeling of the tether and accompanying tether wear

- Unparalleled scalability
- The plant can fly even in zero wind
- No dedicated launch/land hardware such as runway, VTOL, props or batteries necessary
- As we use three kites per unit, scale per kite is reduced by $\sqrt{3}$ or mass by 42%, assuming cubic scaling of mass, compared to a single kite unit

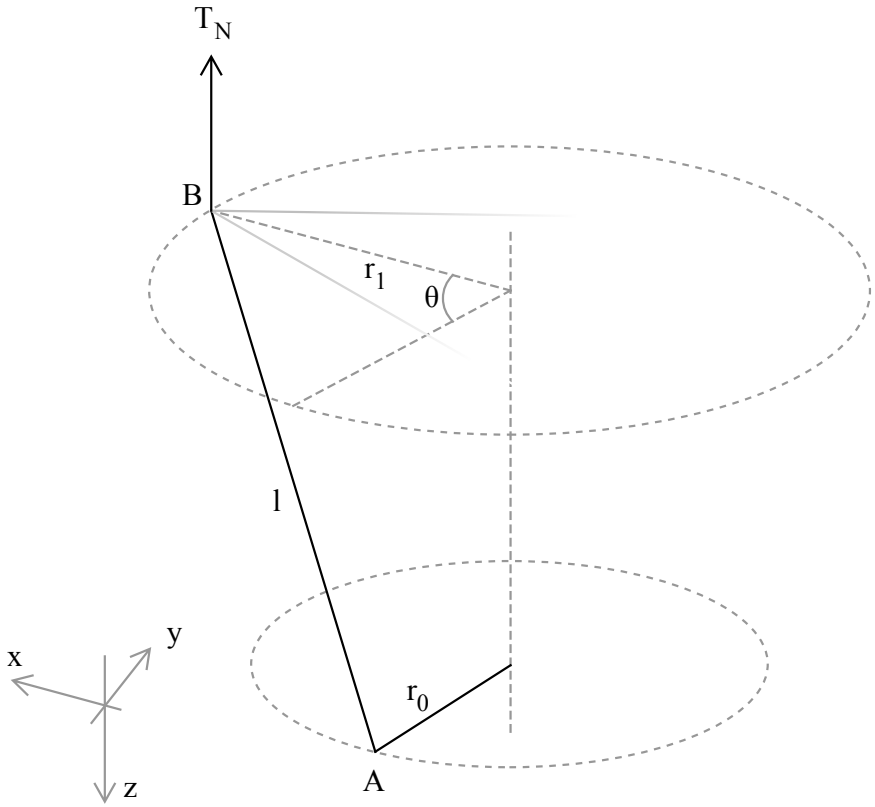
Drawbacks would be

- Launch plan needs to be validated
- Tether length is limited
- Tower is required at the ground [though it is lower than a HAWT]
- Some elevation angle is necessary
- Cartwheel at ground is necessary

The shaft

When analyzing the shaft we will consider only one tether and one kite. The complete assembly will have three [or more] kites, but extending the calculations for such scenarios should be straightforward.

The figure below shows the basic shaft geometry. A coordinate system is shown, aligned to the arm at the kite side [top, x-axis], the center of the shaft [z-axis] and the y-axis being perpendicular to both of these to form a right handed coordinate system. The origin is at the center of rotation in the kite rotation plane. This coordinate system is used only for calculations, and is not otherwise important.



T_N	tether tension component normal to the plane of rotation
T	tether tension along the tether
θ	the phase angle difference between the ground station [bottom] and kite layer [top]
r_0	the radius of the cartwheel at the ground side
r_1	the radius of rotation of the kites
l	the length of the tether
A	the position of the tether attachment on the cartwheel
B	the position of the kite
$\hat{\imath}, \hat{\jmath}, \hat{k}$	the unit vectors in x, y and z direction respectively

The vector value of $\overrightarrow{T}_N$ is

$$\overrightarrow{T}_N = T_N \hat{k} \tag{1}$$

We may see the tension along the centerline to be the tether along the tether projected along the z-axis. We may use this to find the tension along the tether itself. We see that the tether tension T along the tether itself is given by [where $\_ \cdot \_$ is the vector dot product]

$$T_N = T \left[\hat{k} \cdot \frac{\overrightarrow{AB}}{\left| \overrightarrow{AB} \right|} \right] \tag{2}$$

The coordinate of A is [$\mathrm{RotateZ}(x)$ represents a function returning a rotation matrix rotating a point an angle x around the z-axis]

$$\overrightarrow{A} = \mathrm{RotateZ}(-\theta) \left(r_0 \hat{\imath} \right) - z_A \hat{k} \tag{3}$$

The coordinate of B is

$$\overrightarrow{B} = r_1 \hat{\imath} \tag{4}$$

Thus the vector $\overrightarrow{AB}$ is given by

$$\overrightarrow{AB} = \overrightarrow{B} - \overrightarrow{A} \tag{5}$$

We can replace the use of z_A with l by solving

$$\left| \overrightarrow{AB} \right| = l \tag{6}$$

To find the moment applied by the shaft we use force times arm length

$$\tau = T \frac{\overrightarrow{AB} \cdot \hat{\jmath}}{\left| \overrightarrow{AB} \right|} r_1 \tag{7}$$

A this point we have all the necessary equations to solve the system using a computerized equation solve such as Maple. We get

$$T_N = T \frac{\sqrt{2\cos(\theta) r_0 r_1 + l^2 - r_0^2 - r_1^2}}{l} \tag{8}$$

$$\tau = \frac{T_N \sin(\theta) r_0 r_1}{\sqrt{2\cos(\theta) r_0 r_1 + l^2 - r_0^2 - r_1^2}} \tag{9}$$

If we assume that $l \gg r_0$ and $l \gg r_1$ we see that τ has its maximum around $\theta \approx 90°$. We may calculate the exact maximum though it is not so important here and we will skip it.

The maple code to arrive at these results is

```
Av := RotationMatrix(-theta, <0, 0, 1>) . (r__0 * <1, 0, 0>) - <0, 0, z__A>;
Bv := r__1 * <1, 0, 0>;
ABv_tmp := Bv - Av;
sol1 := solve(ABv_tmp . ABv_tmp = l^2, z__A)[1];
ABv := subs(z__A = sol1, ABv_tmp);
T := T__N / (<0, 0, 1> . ABv) * l;
tau := simplify(T * (ABv . <0, 1, 0>) / l * r__1);
```

Scaling the shaft

We will look at scaling the shaft by a factor x. This means that every dimension should also scale by x. Some consequences of this are

- Any geometric relations are preserved (eg. angles)
- The wingspan and chord are each scaled by a factor x
- The wing area is scaled by a factor x^2
- Any tether length and diameter are scaled by a factor x
- The tether cut area and therefore ultimate strength are scaled by x^2
- Tether mass scales by x^3 as a result of length and diameter scaling
- The aerodynamic properties only change according to Reynolds number, a scaling effect that we will ignore in this context
- The flying speed should remain constant under scaling (it depends only on aerodynamic effects, geometry and wind speed)
- The lift of the wing scales by x^2 due to the scaling of wing area
- The mass of the kite scales by an unknown factor. A scaling of x^2 would describe a wing increasing in area but using with the same material and thickness eg using a canvas material of the same type for the scaled and unscaled kites. A scaling of x^3 would describe cubic scaling of mass when all three dimensions are scaled. For a beam, use of materials may be optimized and may not scale to either of these. We will not dwell too much on kite scaling here, but it should be safe to assume that it must scale by a factor y such that mass scales by x^y where $2 < y < 3$.
- The looping radius is scaled by x
- The rotational speed is scaled by a factor $\frac{1}{x}$ as the forward speed of the kite is constant but the looping radius would increase with scale

The shaft tension T_N should scale by x^2 because of lift scaling.

We assume the unscaled moment is τ_0. A scaled moment thus is given by [using equation (9)]

$$\tau_x = \frac{x^2 T_N \sin(\theta)\, x r_0 x r_1}{\sqrt{2\cos(\theta)\, x r_0 x r_1 + (xl)^2 - (x r_0)^2 - (x r_1)^2}} \tag{10}$$

$$\tau_x = \frac{x^4 T_N \sin(\theta)\, r_0 r_1}{x\sqrt{2\cos(\theta)\, r_0 r_1 + l^2 - r_0^2 - r_1^2}} \tag{11}$$

$$\tau_x = x^3 \tau_0 \tag{12}$$

To simplify working with plants at different scale, we introduce the Λ value. This number is very central to this description of TRPT airborne wind. Λ is non-dimensional and describes a certain shape of shaft and the amount of moment it can transfer from the kites to the ground without collapsing. To go from an Λ value to the actual moment, multiply the Λ

value by the looping radius of the kite and the shaft tension. You may also use the Λ value to describe the induction factor of the plant, ie. how hard the kites are loaded with a harvesting moment/force. A high Λ value indicates that a large braking moment is transferred from the kites to the cartwheel, while a small Λ value indicates that the kites are closer to free wheeling. Life is not that simple though if you want to be exact, because changing the Λ value also changes the kite flying speed, again affecting the tension of the shaft. Still, the Λ value serves a useful purpose in describing the ability to transmit moment in a shaft, free of scaling effects.

If you use the Λ value as a way to indicate induction factor, make sure the selected Λ value is less than the Λ value of the shaft geometry. If you don't, using a larger Λ value for induction means that the shaft would collapse.

The Λ value is defined as moment per tension per looping radius. A formula to express this would be

$$\Lambda = \frac{\tau}{T_N \, r_1} \tag{13}$$

$$\Lambda = \frac{\sin(\theta) \, r_0}{\sqrt{2\cos(\theta) \, r_0 r_1 + l^2 - r_0^2 - r_1^2}} \tag{14}$$

The above equation (14) states the Λ value as a function of shaft twist θ. If we assume that the tether length l is much bigger than either radius, we can find an approximate maximum mtr value shaft can sustain, assuming that it occurs at $\theta = 90°$, as

$$\hat{\Lambda} \approx \frac{r_0}{\sqrt{l^2 - r_0^2 - r_1^2}} \qquad (?)$$

Which may be further approximated to

$$\hat{\Lambda} \approx \frac{r_0}{l} \qquad (?)$$

This formula is useful for arriving at a shaft geometry simply and quickly without any calculators.

We will show that the Λ value is constant under scaling. Let's see what the value of an Λ value scaled in every dimension by x is.

$$\Lambda_x = \frac{\sin(\theta)\, x r_0}{\sqrt{2\cos(\theta)\, x r_0 x r_1 + \left(xl\right)^2 - \left(x r_0\right)^2 - \left(x r_1\right)^2}} \qquad (?)$$

$$\Lambda_x = \frac{\sin(\theta)\, r_0}{\sqrt{2\cos(\theta)\, r_0 r_1 + l^2 - r_0^2 - r_1^2}}$$

$$\Lambda_x = \Lambda_0$$

The mtr value only depends on the geometry of the shaft, not any aerodynamic or other parameters.

Furthermore we can say that the power of the plant is given by the torque multiplied by the rotational speed of the shaft, ω. The speed of the kites themselves will not change much with scale as it depends mostly on wind speed and glide number of the kites themselves. The rotational speed of a scaled plant is expected to be close to $\omega_x = \frac{1}{x}\omega_0$ Thus, for a scaled plant, the power is

$$P_x = \omega_x \tau_x = \frac{1}{x}\omega_0 x^3 \tau_0 = x^2 P_0 \qquad (15)$$

What we can conclude from this is that if we scale a TRPT based plant by a factor of x, the power it may produce is x^2. This means that the power transmitted to the ground by the shaft is proportional to the swept area of the plant. Thus when scaling such a plant, the TRPT shaft does not introduce a scaling penalty, not any scaling benefit.

The flying speed of a kite is mostly dependent on glide ratio and wind speed, and the pull is dependent on wing area. So we may assume that a kite scaled by a factor x should produce a power scaled by x^2, just like the shaft will. It is nice to know that the TRPT shaft scales in power relatively the same as the kite itself. So a scaled up version of a design should work as well as the scaled down one.

We will not go into more details here, but tether drag will also scale by x^2 and thus does not affect a scaled plant. The ratio tether drag to kite pull remains constant.

We should talk about mass scaling separately. The mass of the TRPT shaft will scale by x^3, assuming it is purely built with ropes. The same could maybe be said for the kite, though a detailed analysis of rigid kite mass when scaling is out of the scope of this text.

However this analysis pans out, it is safe to assume that the mass will scale faster than the plant power output. This may ultimately lead to hitting a scaling wall. The Pyramid does have a few benefits though compared to a single kite AWE plant.

We may imagine that connecting a plant to the grid is a substantial part of the installation cost. For every grid connection, for the same power output The Pyramid will have three kites where a single kite plant only has one. The wing area of the single kite is $A_1 = \frac{b_1^2}{\mathrm{R}}$, with b_1 being the wingspan and R being the aspect ratio. The wing area of the three smaller kites should match that of the single kite

$$A_1 = 3A_3$$

$$b_1^2 = 3b_3^2$$

$$b_3 = \frac{1}{\sqrt{3}} b_1$$

If mass scales by a factor x^3, the mass of the three smaller wings should thus be

$$\Sigma m_3 = 3 \left(\frac{1}{\sqrt{3}} \right)^3 m_1$$

$$\Sigma m_3 = \frac{1}{\sqrt{3}} m_1$$

Thus we see that the kite mass of a three kite Pyramid is 42% less than a similarly sized single kite plant.

Furthermore, the TRPT is hardly affected by gravity slowdown while traveling crosswind. This is due to the triangular bridle transferring forward and braking forces between the three kites. For this reason we may expect a Pyramid plant to be much less affected by heavy kites compared to a single kite plant.

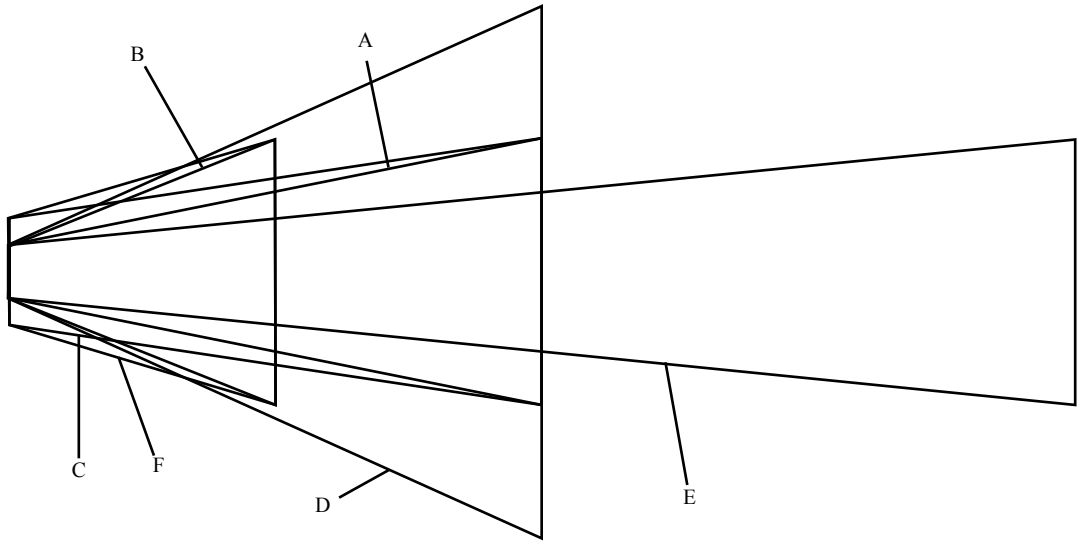
Furthermore, the Pyramid does not have any launch and land equipment onboard the kites, something that single kite plants would normally have.

All of these arguments point to a conclusion that the Pyramid will scale substantially higher in unit power per ground station compared to a single kite plant, before hitting the scaling wall.

What can we expect of the Λ value?

Let's go ahead and calculate some Λ values.

Using the equation (14) we calculate an Λ value for a few selected shafts at $\theta = 90°$, imagined to be suitable for a 5 - 10 m wingspan kite.



	l	r_0	r_1	Λ
A	200 m	10 m	50 m	0.052
B	100 m	10 m	50 m	0.116
C	200 m	20 m	50 m	0.103
D	200 m	10 m	100 m	0.057
E	400 m	10 m	50 m	0.025
F	100 m	20 m	50 m	0.237

As the mtr values are pretty small for long tethers, we may conclude that we would like to maximize both the tether tension T_N and the rotational speed ω. This can be done by having a high efficiency wing with high glide number G_e.

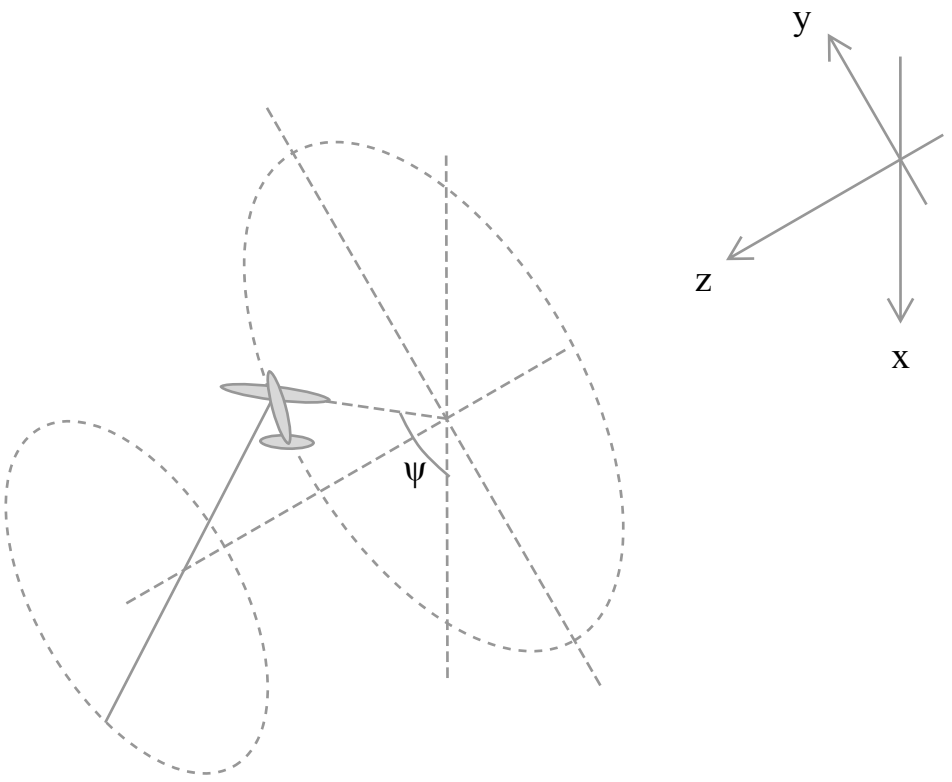
The optimum MTR value

Like for the previous example, we will assume a kite that is rotating in the kite plane with the wing aligned to said plane in roll. This is possible until the scale of the plant is so large that centrifugal forces alone is not enough to tension the triangular bridle at the kites.

The harvesting torque τ from the shaft is manifested at the kite as a force pointing opposite of the direction of travel with a value $F_H = \frac{\tau}{r_1}$. Furthermore we have

C_L	The lift coefficient of the kite
C_D	The drag coefficient of the kite, including a tether component
ρ	Density of air
w	Wind speed
S	Wing area
θ_e	Elevation angle of the shaft
ψ	The phase angle of the kite
v_k	Forward speed of kite, relative to ground

The sketch shows the coordinate system used for calculations. The z-axis is aligned along the center of the shaft pointing towards the ground. The y-axis would point up if the elevation angle was zero. The x-axis points horizontally. The wind is assumed to be directed horizontally aligned [no azimuth] to the shaft.



Kite forward direction unit vector

$$\overrightarrow{k_f} = \text{RotateZ}\left(\psi\right)\hat{j} \tag{16}$$

Unit vector pointing along the right wing

$$\overrightarrow{k_r} = \text{RotateZ}\left(\psi\right)\left(-\hat{\imath}\right) \tag{17}$$

Wind speed vector

$$\overrightarrow{w} = \text{RotateX}\left(-\theta_e\right)\left(-w\,\hat{k}\right) \tag{18}$$

Apparent speed at the kite

$$\overrightarrow{v_a} = v_k\overrightarrow{k_f} - \overrightarrow{w} \tag{19}$$

The direction of the lift is perpendicular to both the apparent wind and the right wing vector. We can use the cross product to produce a normalized vector in this direction

$$\overrightarrow{k_l} = \frac{\overrightarrow{k_r} \times \overrightarrow{v_a}}{\left|\overrightarrow{k_r} \times \overrightarrow{v_a}\right|} \tag{20}$$

The aerodynamic forces are given by [note drag includes a portion related to the tether drag, included in the C_D factor]:

$$L = \frac{1}{2}\rho C_L S \left|\overrightarrow{v_a}\right|^2 \tag{21}$$

$$D = \frac{1}{2}\rho C_D S \left|\overrightarrow{v_a}\right|^2 \tag{22}$$

We may now produce an equation for force equilibrium in the direction of travel. Using Newtons first law, the sum of forces should be zero

$$\left[L\overrightarrow{k_l} - D\frac{\overrightarrow{v_a}}{\left|\overrightarrow{v_a}\right|}\right] \cdot \overrightarrow{k_f} - F_H = 0 \tag{23}$$

We have left out gravitational forces here. This is a simplification we may do at this point as the triangular bridle will make sure that the kites are traveling at near constant speed. We only need to provide a vertical force at a macro level to keep the shaft from falling down, something we will describe later in the control system section. We will assume that creating such a force does not incur great losses. This also means that equation (23) will

not describe the speed in the loop accurately, as the same bridle will even out the speed calculation relating to different values of ψ.

At this point, we may again use Maple to solve these equations. unfortunately, the results are quite intractable, so we will have to look at some simplified equations instead. First, we select $\theta_e = 30°$ and $\psi = 90°$. Then we assume that $v_k \gg w$ and remove w from some parts of the equation. This leaves us with an estimate kite speed $\tilde{v}_k$

$$\tilde{v}_k = \frac{\sqrt{3}}{4}\frac{C_L}{C_D}w + \sqrt{\frac{3}{16}\left(\frac{C_L}{C_D}\right)^2 w^2 - 2\frac{F_H}{\rho C_D S}} \tag{24}$$

The power of the plant is given by

$$\tilde{P} = F_H \tilde{v}_k \tag{25}$$

The maximum power is given when

$$\tilde{F}_{H,opt} = \frac{\rho C_L^2 S w^2}{12 C_D} \tag{26}$$

If we assume that the lift force L is equal to T_N we end up with an Λ value of

$$\Lambda_{opt} \approx \frac{1}{2\frac{C_L}{C_D}} \tag{27}$$

So for a kite and tether with a combined glide number of 20, an mtr value of 0.025 should be sufficient. In practice the maximum Λ value the shaft must support will be somewhat greater than the Λ value in common use, to avoid over-twisting the shaft. But also, sufficient performance may be achieved with a lower Λ value and running at lower than optimal induction factor for the kite.

The power produced by the plant at optimum Λ value is approximated to

$$\tilde{P}_{opt} = \tilde{F}_{H,opt}\left[\tilde{v}_k\Big|_{F_H=\tilde{F}_{H,opt}}\right] \tag{?}$$

$$\tilde{P}_{opt} = \frac{\sqrt{3}\, C_L^3 w^3 S \rho}{36\, C_D^2} \tag{?}$$

Like for other AWE designs, it seems important to maximize $\frac{C_L^3}{C_D^2}$.

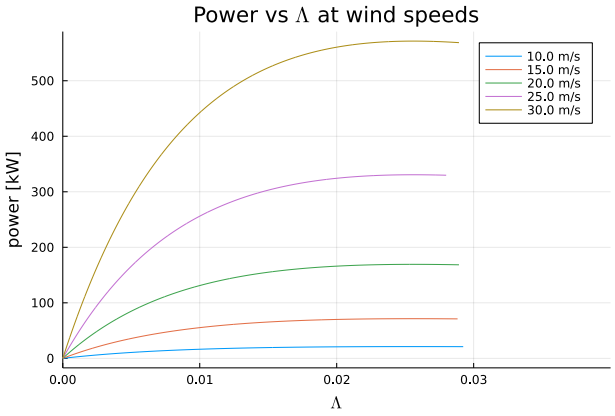
The flying speed at optimum harvesting force is

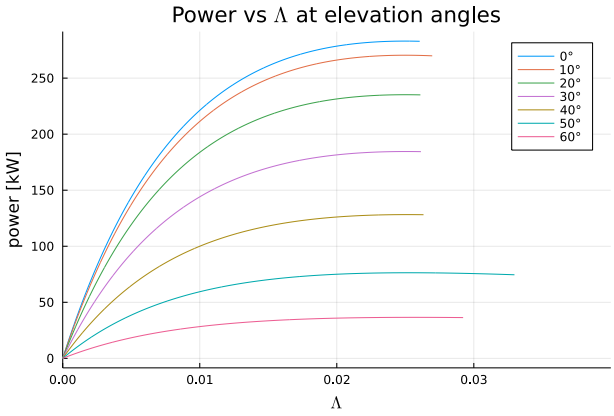
$$\tilde{v}_{k,\mathrm{opt}} = \frac{1}{\sqrt{3}} \frac{C_L}{C_D} w \qquad (?)$$

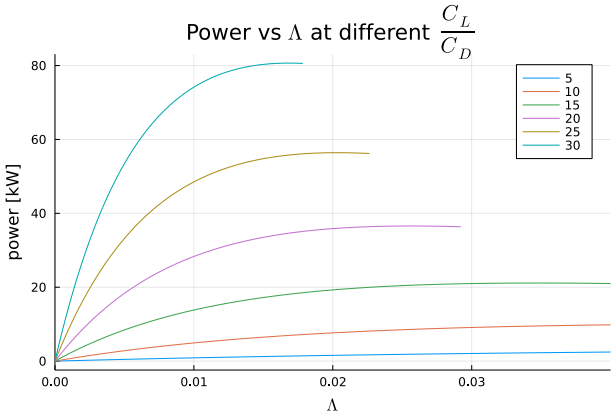
Maple code:

```
with(LinearAlgebra):

k__f := RotationMatrix(psi, <0, 0, 1>) . <0, 1, 0>;
k__r := RotationMatrix(psi, <0, 0, 1>) . <-1, 0, 0>;
w__v := RotationMatrix(-theta__e, <1, 0, 0>) . <0, 0, -w>;
v__a := k__f * v__k - w__v;
k__l := simplify((k__r &x v__a) / sqrt((k__r &x v__a) . (k__r &x v__a)));
L__a := simplify(1/2 * rho * C__L * S * (v__a . v__a));
D__a := simplify(1/2 * rho * C__D * S * (v__a . v__a));

newton := simplify((L__a * k__l - D__a * v__a / sqrt(v__a . v__a)) . k__f - F__H);
# we can multiply newtons equation by this nonzero term to simplify matters
newton2 := newton * sqrt((-cos(theta__e)^2*w^2 + w^2)*cos(psi)^2 +
2*sin(theta__e)*cos(psi)*v__k*w + cos(theta__e)^2*w^2 + v__k^2);

# we select theta__e = 30 deg and psi = 90
newton2_simpl := simplify(subs({theta__e = 30 * Pi / 180, psi = Pi / 2}, newton2));

# simplify by assuming v__k >> w
newton2_simpl2 := simplify(subs({sqrt(v__k^2 + w^2) = v__k, v__k^2 + w^2 = v__k^2, sqrt(4*v__k^2
+ 3*w^2) = 2 * v__k}, newton2_simpl));
sol2 := solve(newton2_simpl2, v__k);

# select the only solution that yields a positive speed at F__H = 0
v__k_solution := simplify(sol2[2]);



# optimal power F__H
F__H_opt := solve(diff(F__H * v__k_solution, F__H), F__H)[2];

# find Lambda at the optimal point
lambda__opt := simplify((F__H_opt * r__1) / r__1 / (1/2 * rho * C__L * S * subs(F__H = F__H_opt,
v__k_solution)^2), assume = positive);

# calculate the power output at optimum
power_opt := simplify(F__H_opt * subs(F__H = F__H_opt, v__k_solution), assume = positive);
```

As there were some approximations made, we will do some numerical calculations to verify the accuracy of our results [see optimal_force.jl for source code].







The plots are done with a nominal $\frac{C_L}{C_D} = 20$, $C_L = 1.5$, wind speed 12 m/s, wing area 3 m², elevation angle 30° and $\rho = 1.225$, unless otherwise stated.

We see that the optimal Λ value does not depend [much] on wind speed or elevation angle. We also see that the optimal Λ value depends a lot on the glide number $\frac{C_L}{C_D}$, and the numerical solution matches well with the one calculated by equation (27).

$\frac{C_L}{C_D}$	Numeric solution	Equation (27)
5	> 0.040	0.100
10	0.040	0.050
15	0.035	0.033
20	0.025	0.025
25	0.018	0.020
30	0.015	0.017

We should also take note of another observation in the graphs. The slope of the power vs mtr curve is pretty flat around the optimum point. If we increase the mtr value, the kite will generally produce more power, but at one point our algorithms will not produce results [presumably the kite does not fly anymore].

If we decrease the Λ value, the power output does not drop a lot initially, then it drops faster. This means that we may choose to use use a longer shaft without losing a lot of power output, if that is beneficial for the whole design. With a glide number of 20, Λ values as low as 0.025 still provide a large power output, a number as low or lower than all the geometries we looked at previously.

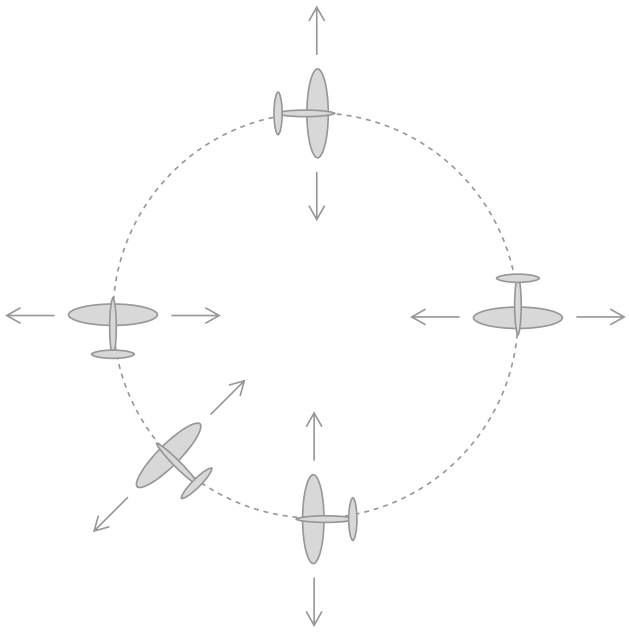
Control system

We have assumed that the triangular bridle connecting the kites is always tensioned. This is plausible because of the large centrifugal acceleration involved in spinning the kites around, but it may also be verified by simulation. One may note that the centrifugal acceleration does decrease with scale.

If the tension of either kite's tether differs from the other two, the shaft will deform. For this reason, to keep the plant running smoothly, the kites must all always try to maintain a constant tether tension. This must account for effects such as the wind affecting the direction of the lift vector, gravitational forces aligned with the shaft and rolling the kites to provide lateral forces. This may be done by not running the kites at maximum lift, but rather a lift that each kite is always able to maintain. Then the C_L may be manipulated to increase lift to compensate for said losses in tether tension. It is envisioned that the algorithm to perform this task may be distributed, only observing the common rotational speed of the plant.

Because the bridle is tensioned, the rotational speed of all three kites must always be the same. As they are traveling up and down in the gravitational field of earth they are subjected to forces that would otherwise speed them up or slow them down. As there are three kites the average effect of gravity should though be zero or close to zero. The bridle has the effect of «lending» propulsion from kites falling to those kites rising. The benefit of running at a constant speed yields higher power output compared to designs with only one kite.

Still, we need to provide a net vertical force facing upwards that keeps the shaft elevated above ground at the desired elevation angle. In a similar fashion we may want to generate a horizontal force to align the shaft to an azimuth angle relative to the wind (eg. for a sailboat).



We see that we can produce such a force by rolling the kite relative to the plane of rotation. This will produce a component of lift force facing perpendicular to the path of travel. We can only produce vertical forces when we are not at the left and right sections of the looping circle. Producing vertical forces is most efficient at the top and bottom of the circle. Most of the time, we produce both a horizontal and vertical component whether we want to or not.

We may use any number of control strategies to control the vertical and horizontal average force components create by the kites. The only requirement is that the average force produced during the loop is equal to the desired force.

Even if we are not able to accurately control the average force components, an actual implementation would probably need to implement a feedback controller to stabilize the shaft elevation angle. This feedback loop would form the outer layer of a cascaded controller.

When we create the controller, we will still just look at the one kite, and assume that it provides just a third of the total average force in either direction.

We select a controller that, for simplicity, generates side forces in the kite rotational plane [as opposed to vertical and horizontal]. The same coordinate system we used for mtr values is used here, so that the x component points horizontally, and the y component points vertically if the elevation angle is zero. We define ϕ as the roll angle of the kite relative to the rotation plane. Positive ϕ rolls the kite towards the center of rotation, and we are assuming the kite is flying in the direction of increasing ψ. Remember we already have a system to provide a constant T_N in the direction perpendicular to the plane of rotation. The forces we create are

$$F_y = -T_N \tan\phi \sin\psi \tag{28}$$

$$F_x = -T_N \tan\phi \cos\psi \tag{29}$$

Our controller will be in the form

$$\tan\phi = K_1 \sin\psi + K_2 \cos\psi \tag{30}$$

We combine these and calculate the average force over a loop

$$\overline{F_y} = \frac{1}{2\pi}\int_0^{2\pi} -T_N \left(K_1 \sin\psi + K_2 \cos\psi\right) \sin\psi \, \mathrm{d}\psi$$

$$\overline{F_x} = \frac{1}{2\pi}\int_0^{2\pi} -T_N \left(K_1 \sin\psi + K_2 \cos\psi\right) \cos\psi \, \mathrm{d}\psi$$

These may be solved with Maple, giving us the controller

$$\tan\phi = -\frac{2}{T_N}\left(\overline{F}_y \sin\psi + \overline{F}_x \cos\psi\right)$$

The Maple code to solve the equations

```
tmp_int_y := 1/2 / Pi * int(-T__N * (K__1 * sin(x) + K__2 * cos(x)) * sin(x) , x = 0 .. (2 *
Pi));
tmp_int_x := 1/2 / Pi * int(-T__N * (K__1 * sin(x) + K__2 * cos(x)) * cos(x) , x = 0 .. (2 *
Pi));
K__1_sol := solve(tmp_int_y = F__y_avg, K__1);
K__2_sol := solve(tmp_int_x = F__x_avg, K__2);
```

Simulations

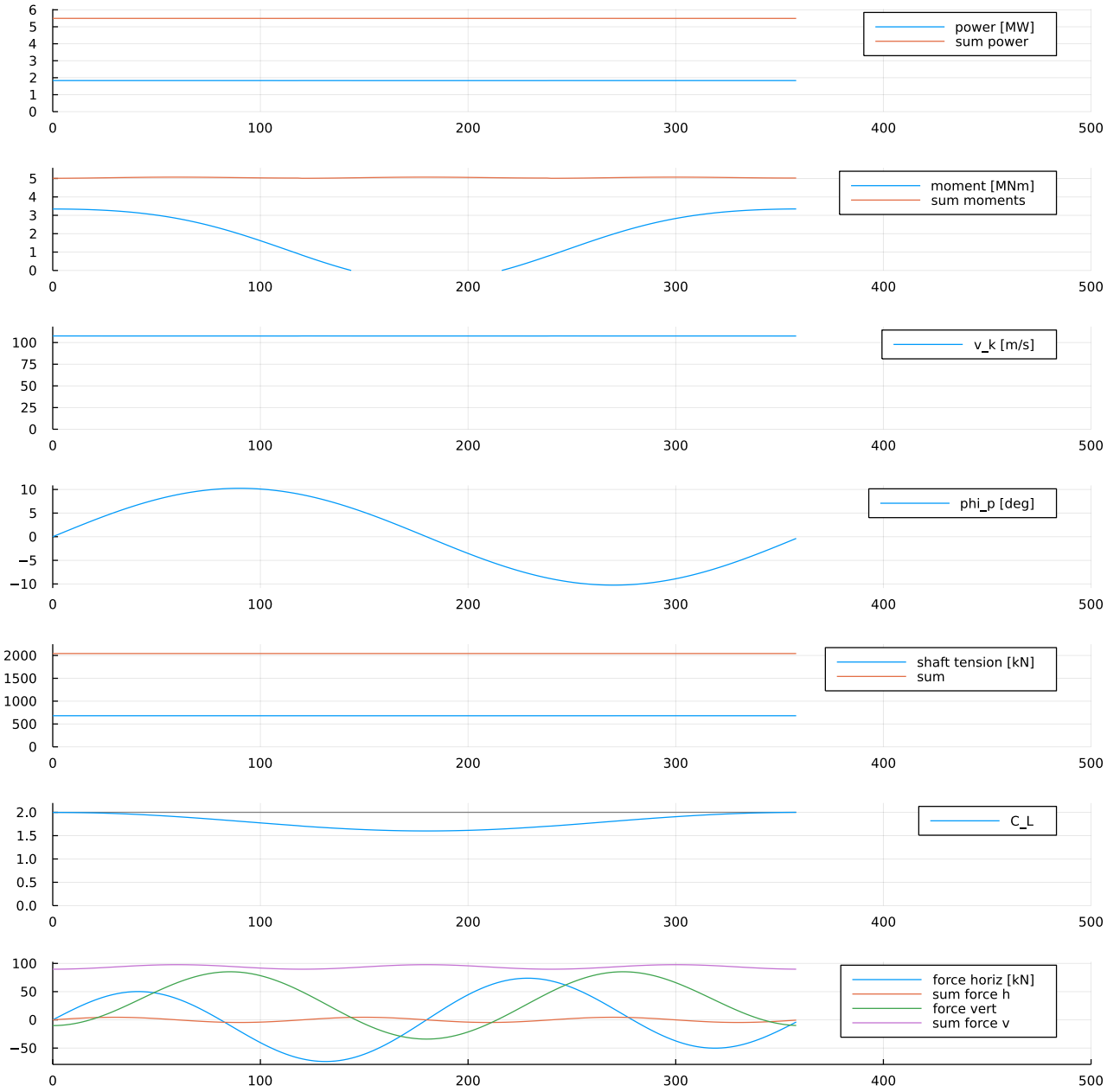
The simulations are done in a custom made Julia simulation tool called TRPTSim. It will simulate the three kites each offset in ψ by 120°. The kites are simulated using a horizontal-vertical average force controller similar to the one described previously, as well as a system to find the appropriate tension of the tether and then control it to a level slightly lower than the max capability of the kite, and mostly constant for all tethers throughout the loop.

As the kite, we will use something loosely based on the Makani 600 kW «October kite». Chosen and supplied parameters are

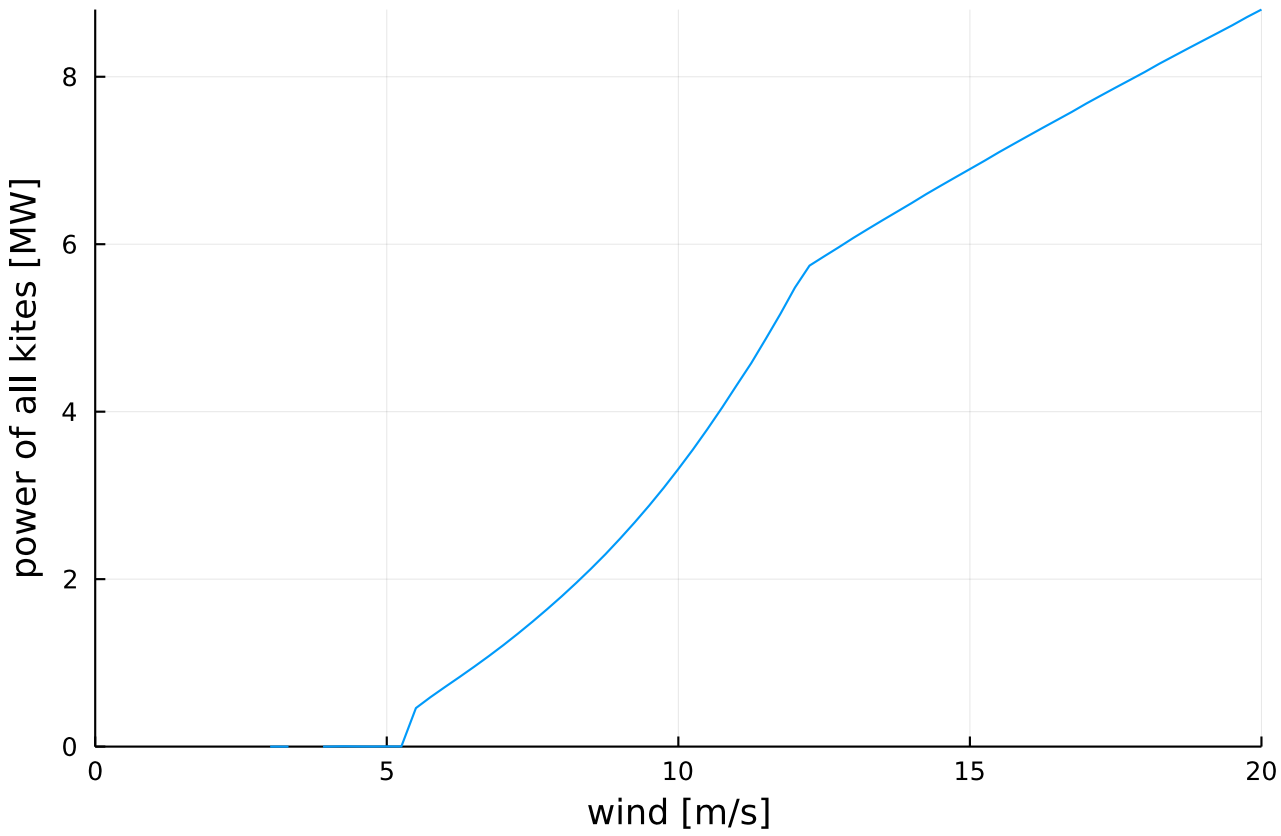
Wingspan	26 m
Wing surface area	54 m²
Weight	1852 kg
Nominal C_L	2.0
Glide number at nominal C_L	19.2
Glide number incl. tether, at nominal C_L	13.1
Looping radius	100 m
Tether diamater	40 mm
Tether length	250 m
Tether safety factor	3x
Tether drag coefficient	1.1
Elevation angle	30°
Λ value	0.025

Julia code to execute the simulation:

```
using TRPTSim
cfg = october_kite()
mtr = 0.025
azimuth = 0.0
wind_speed = 12.0
force_horizontal = 0.0

# note mtr is old naming for Lambda
(df, input) = solve_sector_df(cfg, wind_speed, azimuth, mtr, force_horizontal);
plot!(plot_solution(df, input), size=(1000, 1000)) # first figure

plot_power_curves(power_curve(cfg, 3:0.25:20, azimuth, mtr, force_horizontal)) # second figure
```



Simulation output for 12 m/s wind. The red/pink curves indicate values for all three kites, the blue/green curves for a single kite



Power curve for 3 × 600 kW October kites. Some observations

- The cut-in wind speed is very low for such a large kite, around 5 - 6 m/s
- At 12 m/s the kites are producing 6 MW, which is a factor 3.3x more than the rated power of the Makani design for the same kite
- At 12 - 13 m/s the simulator limits lift in the kite due to high tether tension. Still, the output power increases due to faster rotation of the shaft

The power curve seems very good for such a detailed simulation.

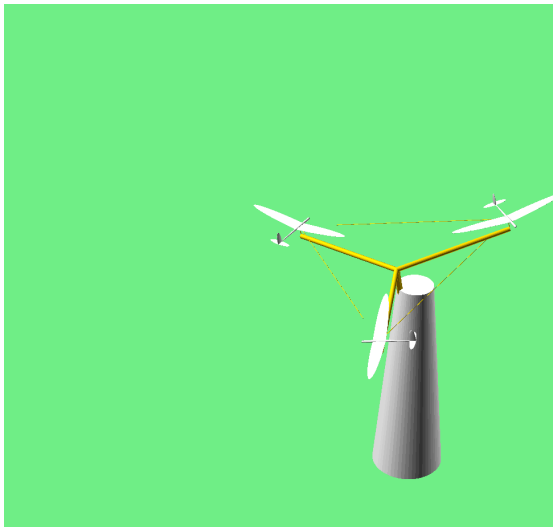
Take off and landing

The takeoff and landing sequences are similar except in opposite step order.

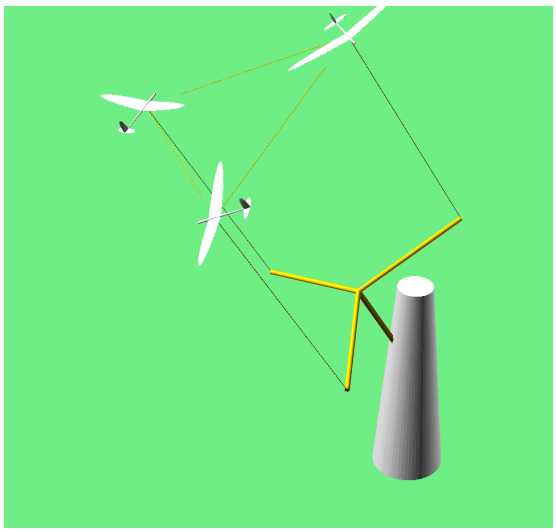
Takeoff is initiated when the kites are attached directly to the cartwheel by means of a tether winch. As the triangular kite bridle is too large for this configuration of kites, each kite also has a small onboard bridle winch so that the slack in the bridle may be picked up while launch and landing is progressing.

Note that once the plant is operational, both the cartwheel tether winch and the onboard bridle winch are not actively used.

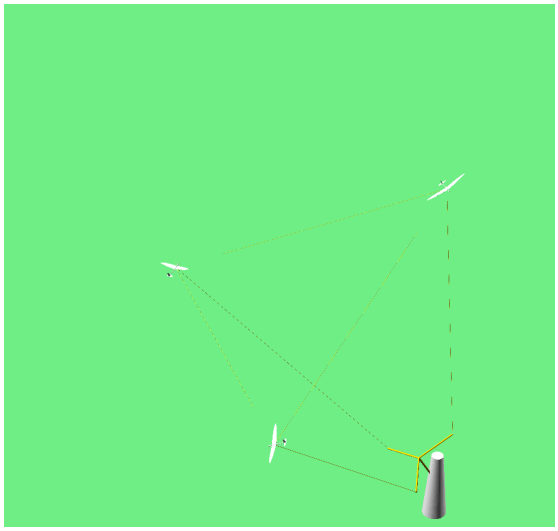
The sequence starts with the generator acting as a motor, spinning the cartwheel to a high speed. Thus the airspeed of the kites will be larger than the wind speed and the roll is controllable. The elevation angle of the shaft may be increased to make more space for the kites to fly.



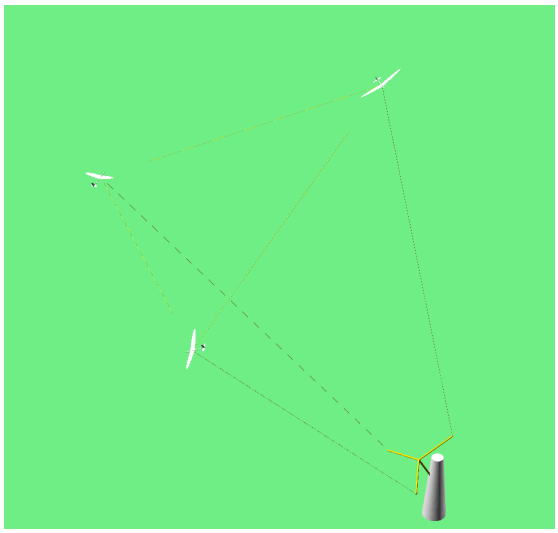
As the tethers reel out, the elevation angle is lowered gradually to let the kites face into the wind.



As the tether get longer, the triangular bridle is gradually reeled out to its full size.

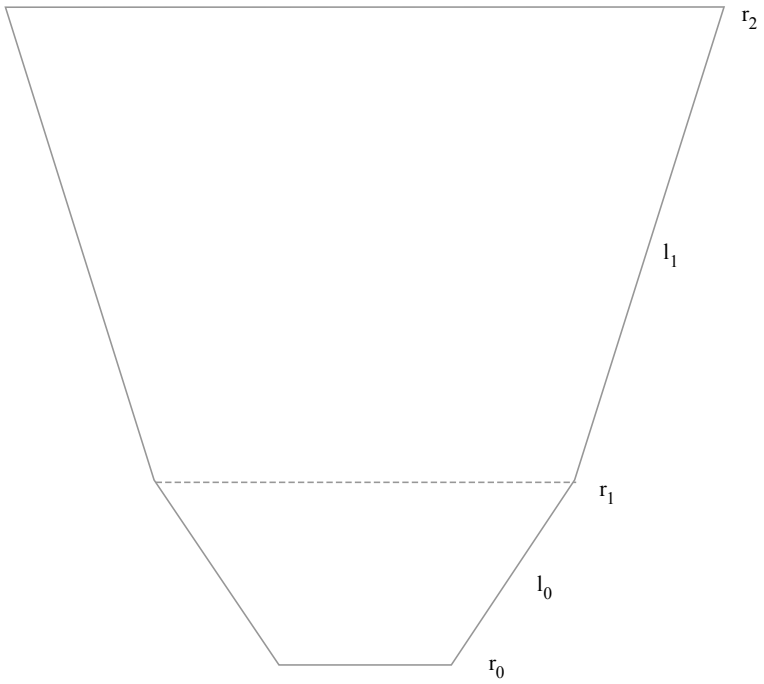


When the tethers and bridle are all fully deployed, they will rest at their end stops. The generator no longer needs to act as a motor, rather the kites drive the rotation using the power of the wind. When the kites apply their nominal high lift C_L, the generator may apply the harvesting moment. From here on, the plant may run for as long as the wind conditions are suitable.



The umbrella shaft modification

Though we have so far opted for the simplest possible shaft. We will have a look at a shaft modification to increase the Λ value, extend the shaft length and/or decrease the size of the cartwheel.



As the value of Λ is defined relative to the kite looping radius r_2, we need to select a matching Λ for the bottom shaft section.

$$\tau_0 = \tau_1$$

Using (13) we get

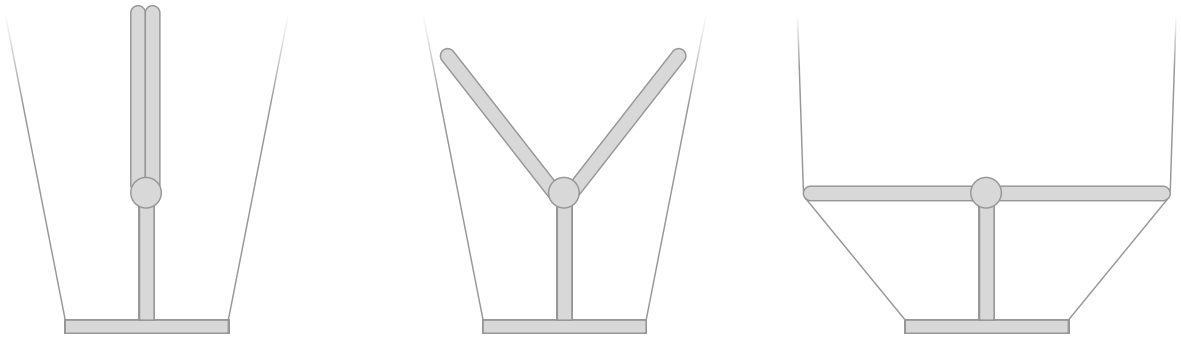
$$\Lambda_0 T_N r_1 = \Lambda_1 T_N r_2$$

$$\Lambda_0 = \frac{r_2}{r_1}\Lambda_1$$

As Λ_1 is the value we normally would use as the Λ value we just write

$$\Lambda_0 = \frac{r_2}{r_1}\Lambda$$

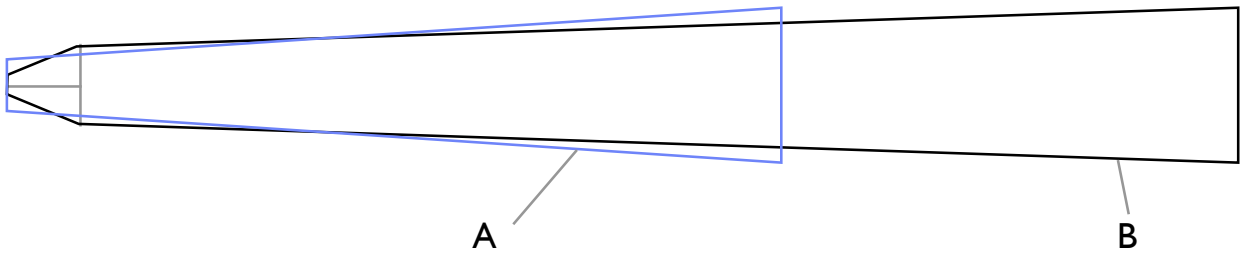
A simple umbrella mechanism to achieve such a segmented shaft without resorting to anything actually hanging freely on the shaft is shown in the sketch below



step 1 step 2 step 3

The figure below shows the effect of the umbrella modification to the shaft. The two geometries depicted both have the same Λ value.

	r_0	r_1	r_2	l_0	l_1	Λ
A	10 m	30 m	N/A	300 m	N/A	0.033
B	3.75 m	15 m	30 m	30 m	450 m	0.033



The umbrella construction should not provide any torsional stiffness, rather be allowed to twist freely. It must withstand the compressive force of the tether and collapsing [ie. closing the umbrella].

All this being said, the simple shaft geometry with only a single section does seem to allow sensible dimensions, making it hard to justify the need for the umbrella modification. If the cartwheel is small relative to the kite wingspan, it may prove difficult to control the kites during takeoff and landing. This may limit the usefulness of decreasing the cartwheel size.

Tether drag

Please refer to the companion note «A simplified drag estimate for a tether with a belly» for a description about tether drag for TRPT. We could use the straight line drag or the improved upon «belly» calculation of tether drag. Both equations can be shown to scale tether drag by x^2 which is also the case for kite lift. So we can assume that the glide ratio of a scaled system remains constant, though we will not prove this here. The «belly» drag calculation is selected here as for this type of rig, we might encounter configurations where the belly of the shaft tethers have some impact on the overall system performance.

We will now do an analysis where we first assume that we have a tether with a given length and diameter, and a kite attached to the tether that provides the maximum allowed pull as defined by a safety factor and the ultimate yield modulus of the UHMWPE tether.

We also need to state at which wind speed such a zero drag kite needs to produce said lift. When that kite is flying at that wind speed, we assume the kites are flying at the speed

$$v_k = G_e w \cos\theta$$

With v_k being the ground speed of the kite (also assumed to be the airspeed as it is flying much faster than the wind), w is the nominal wind speed where the rig is designed to produce full power, and θ is the elevation angle of the centerline around which the shaft rotates.

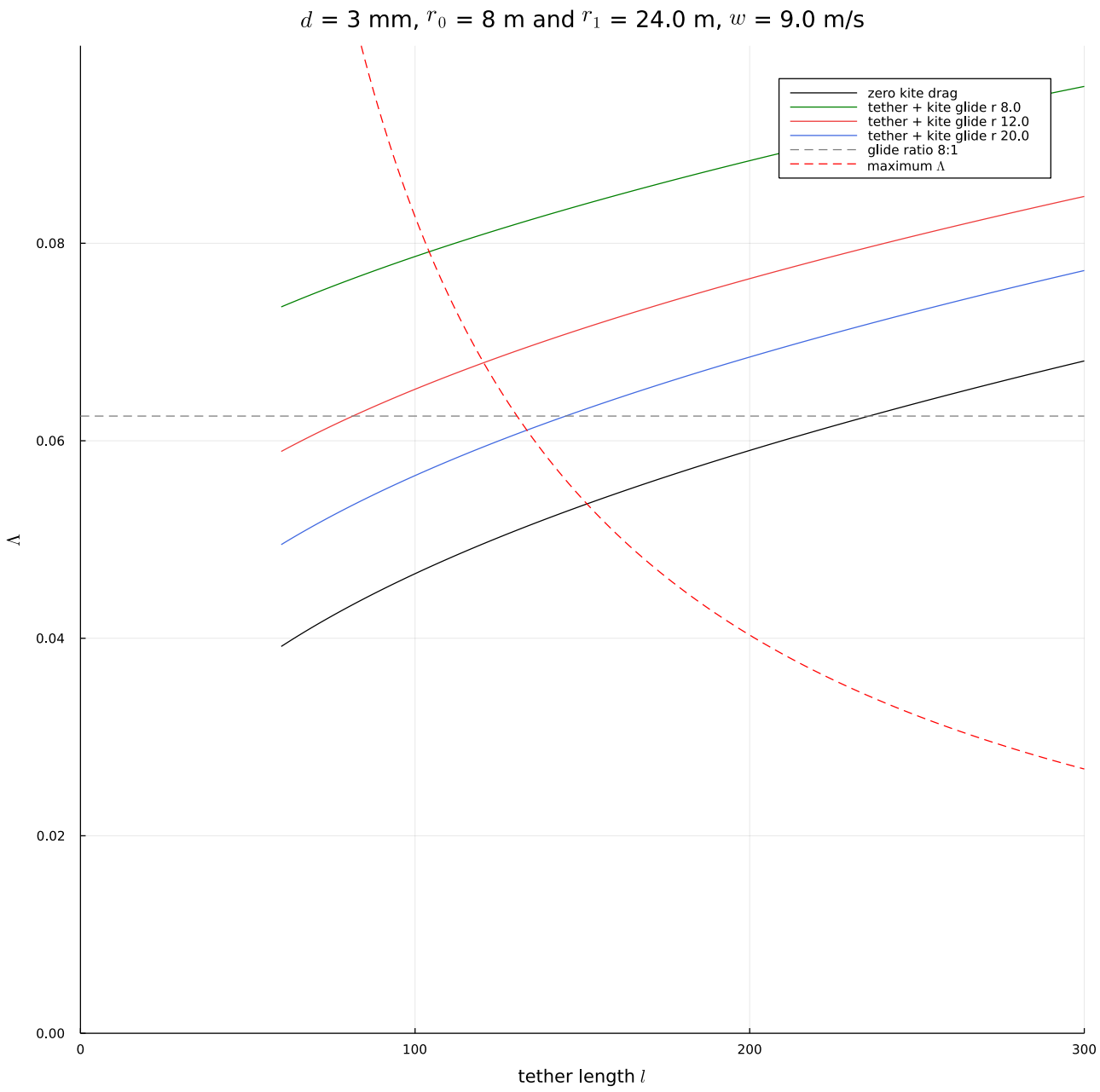
Having now the lift the tether supports L_t and other tether dimensions, we can calculate the drag of the tether D_t and use these to calculate a ceiling glide ratio for the system $G_e = \frac{L_t}{D_t}$. This G_e represents the highest possible overall glide ratio a rig could have if we had a kite without any drag.

We may further extend this model by assuming a kite with a certain glide ratio $G_{e,k}$ and assuming the drag it will produce is close to

$$D_k \approx \frac{L_t}{G_{e,k}}$$

In the following plots, this analysis has been implemented in a Julia script [zero_drag_kite_shaft_length.jl]. We have assumed a safety factor on the tether of 4.0 (usable peak force vs ultimate strength of tether). The first plot assumes a kite dimensioned for using a 3 mm tether providing full power at 9 m/s wind speed. The tether drag coefficient is $C_{D,t} = 1.1$.

Note that no information about the kite other than the glide ratio $G_{e,k}$ went into this model. So the plots are very centered around what is possible with a given tether, regardless of kite design.



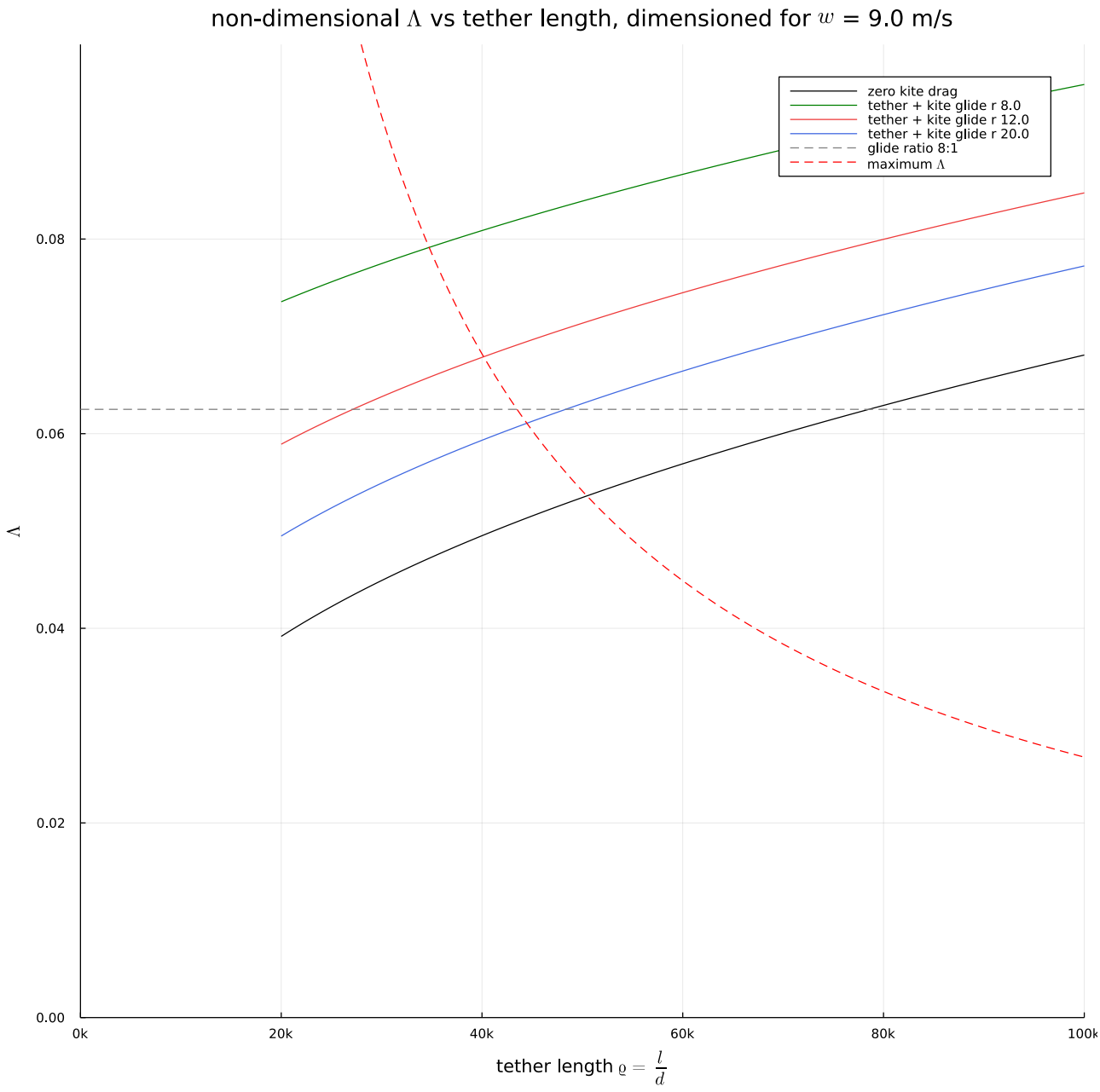
The dashed red line shows the moment-to-tension-to-radius ratio Λ showing the maximum amount of moment that may be transferred by a soft shaft with l of varying lengths on the x-axis. This is an absolute upper limit for that geometry. We see that if we aim for a glide ratio of 8:1 [gray dotted line], we must have a tether l shorter than 120 m approximately [where the red and gray dotted lines meet].

The solid black line represents the optimum Λ induction factor of a zero kite drag curve. The longer the tether, the smaller the possible glide ratio of the rig, and the bigger the value of Λ. We may recall that glide ratio and Λ are inversely related for the optimum power output, related to the induction factor of the rig. For these curves, some reduction of Λ may be acceptable if the shaft is incapable, and some maybe small loss of power can be tolerated.

Looking at the blue, red and green solid curves, the effect of adding drag to the kite has been included, giving the kite different glide numbers 20:1, 12:1 and 8:1. All of these should be feasible to build at large scale using rigid materials such as carbon fiber composites.

We see that is we want to achieve a system overall glide ratio of 8:1, a tether of around 120 m would be the maximum possible for this geometry, and the kite should have a glide ratio of around 20:1. At this point on the curve, the soft shaft is just barely supporting the twisting moment required.

We may introduce the tether aspect ratio $\varrho = \frac{l}{d}$ of the tether to make a non-dimensional version of this plot. We have not shown that it will not change with scale, though numerical analysis shows that it should not.



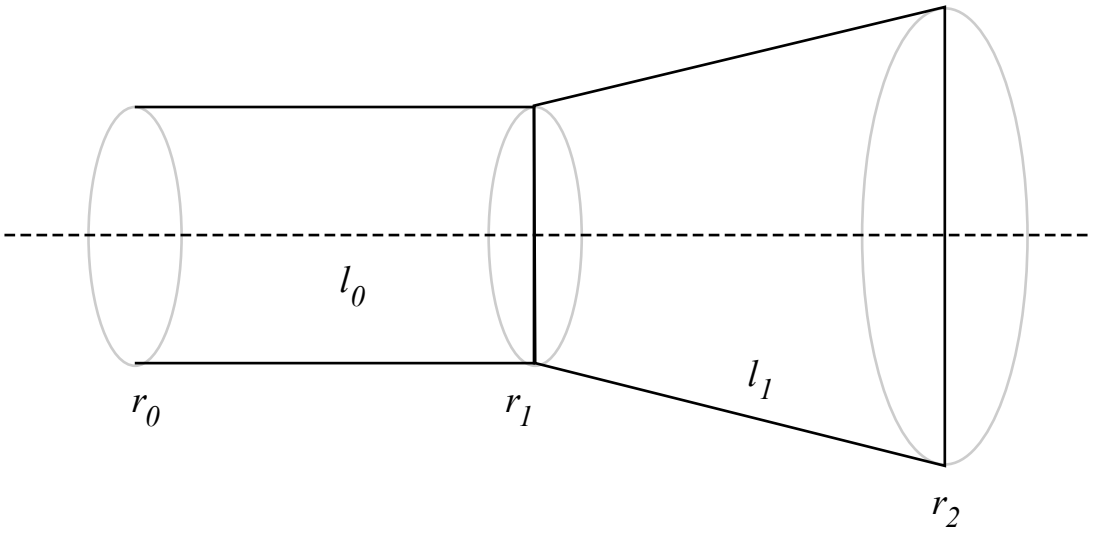
Like before, for a 8:1 overall glide ratio rig, we can read that ϱ must be around 45k or less with a kite glide ratio 20:1.

As a general observation, we see that improving tether drag conditions seem more effective than improving the kite glide ratio.

The Trousers Modification

As tether drag may be a bigger issue than Λ value, we introduce the trouser modification that reduces tether drag but also reduces the mtr value. Because the calculations are becoming more involved, we will select a particular configuration that makes calculations easy, but other configurations may be more optimal if one took the time to perform more detailed analysis.

We will split the shaft into two sections. The top section will be pyramid shaped, the



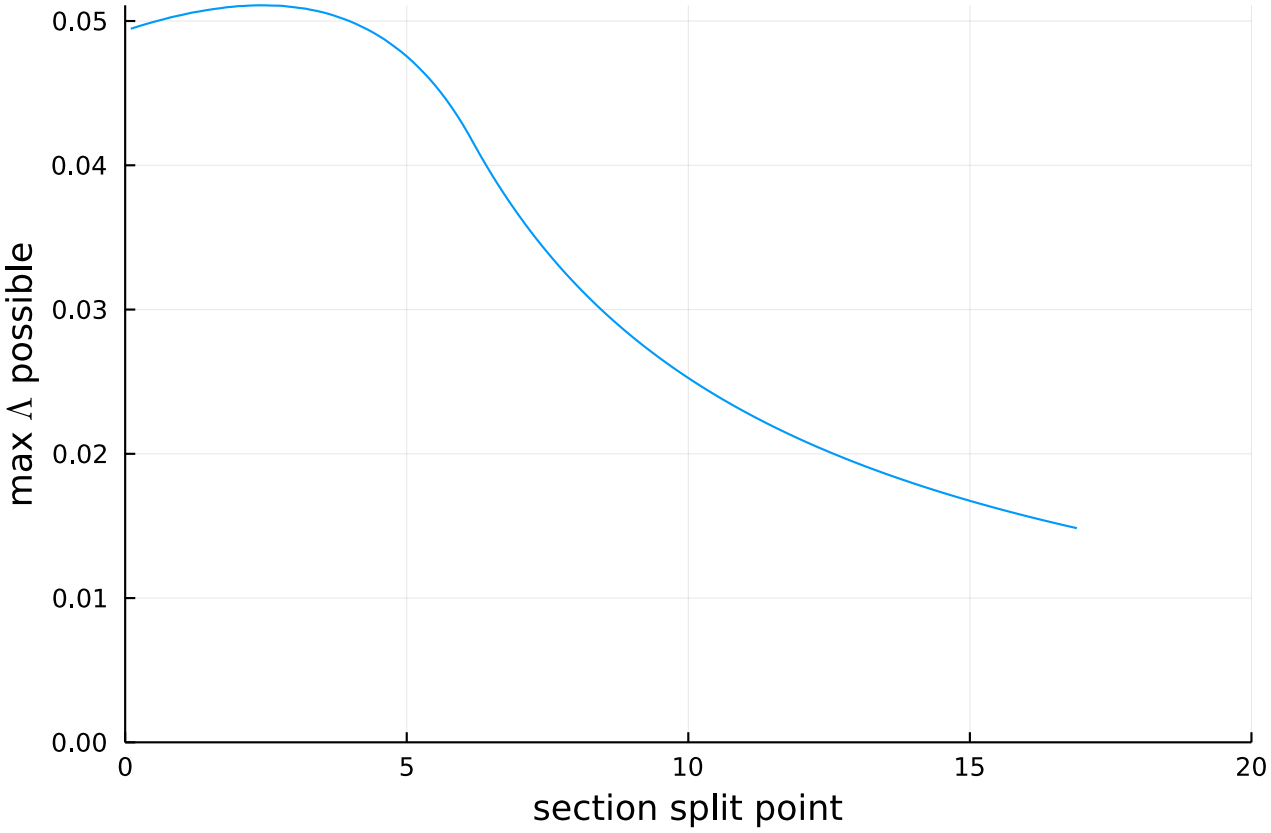
bottom parallel. We will not calculate the Λ value here as it is a bit complicated.
The idea is that the «trousers» is a triangular soft bridle that is attached to the shaft while reeling out. This makes most practical sense if the radius of the bridle matches the radius of the cartwheel. The implications on the Λ value is worse the longer l_0 is. Still, the tether drag may be reduced a lot, and the increase of glide number for the plant will decrease the need for a high Λ value. If tether drag is the limiting factor to extend the shaft length, this is the way to extend the shaft a little more without hurting performance.

The Λ value is approximated coarsely to be between

$$\frac{r_0}{l_0 + l_1}\frac{r_1}{r_2} < x < \frac{r_0}{l_0 + l_1}$$

And the drag would be approximately half if $l_0 = l_1$.

To see more accurately how the Λ value is affected by the trousers modification, we have run a numeric simulation of maximum possible achievable Λ value for varying values of l_0 and a total tether length of $l_0 + l_1 = 20$, and with radii $r_0 = 1, r_1 = 1$ and $r_2 = 4$ [double_section.jl]



The code makes the assumption that the bridle at the join between the two sections should not become compressible. Using trial and error showed that this was not the case for 0 to 90 degrees, though the code may be improved upon to remove such invalid solutions.

Julia code: optimal_force.jl

```
using LinearAlgebra
using Roots
using Rotations
using Plots
using LaTeXStrings

# accoring to document chapter about optimal Lambda (moment to tension to radius)
# coordinate system has x-axis horizontal, z-axis along center of shaft and
# y-axis mostly upwards
#
# psi is the heading angle of the kite in the loop, 0 being facing mostly
# upwards, 90 deg being at the top of the loop

function kite_fwd_vec(psi)
  RotZ(psi) * [0.0; 1.0; 0.0]
end


function kite_right_wing_vec(psi)
  RotZ(psi) * [-1.0; 0.0; 0.0]
end


function wind_vec(wind_speed, elevation_angle)
  RotX(-elevation_angle) * [0.0; 0.0; -wind_speed]
end


function apparent_wind(psi, kite_speed, wind_speed, elevation_angle)
  kite_speed * kite_fwd_vec(psi) - wind_vec(wind_speed, elevation_angle)
end

function direction_of_lift_norm_vec(psi, kite_speed, wind_speed, elevation_angle)
  normalize(cross(kite_right_wing_vec(psi), apparent_wind(psi, kite_speed, wind_speed,
elevation_angle)))
end


function lift_force_vec(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
density_of_air, wing_area)
  v_a = norm(apparent_wind(psi, kite_speed, wind_speed, elevation_angle))
  direction = direction_of_lift_norm_vec(psi, kite_speed, wind_speed, elevation_angle)
  0.5 * density_of_air * lift_coefficient * wing_area * v_a^2 * direction
end


function drag_force_vec(psi, kite_speed, wind_speed, elevation_angle, drag_coefficient,
density_of_air, wing_area)
  v_a_vec = apparent_wind(psi, kite_speed, wind_speed, elevation_angle)
  v_a = norm(v_a_vec)
  direction = -normalize(v_a_vec)
  0.5 * density_of_air * drag_coefficient * wing_area * v_a^2 * direction
end


function shaft_tension(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
  lift_vec = lift_force_vec(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
density_of_air, wing_area)
  drag_vec = drag_force_vec(psi, kite_speed, wind_speed, elevation_angle, drag_coefficient,
density_of_air, wing_area)
  # never mind harvesting force which is perpendicular to the shaft center line
  direction_of_shaft_vec = [0.0; 0.0; -1.0]
```

```
  dot(lift_vec + drag_vec, direction_of_shaft_vec)
end


function lambda(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area, harvesting_force)
  tension = shaft_tension(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
  kite_loop_radius_dummy = 1.0 # just to make the lambda calc easier to understand
  (harvesting_force * kite_loop_radius_dummy) / kite_loop_radius_dummy / tension
end



function find_equilibrium_kite_speed(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area, harvesting_force; initial_speed = wind_speed *
lift_coefficient / drag_coefficient * cos(elevation_angle))
  fwd_vec = kite_fwd_vec(psi)
  harvesting_force_vec = -harvesting_force * fwd_vec
  fun = function(kite_speed)
    lift_vec = lift_force_vec(psi, kite_speed, wind_speed, elevation_angle, lift_coefficient,
density_of_air, wing_area)
    drag_vec = drag_force_vec(psi, kite_speed, wind_speed, elevation_angle, drag_coefficient,
density_of_air, wing_area)
    dot(lift_vec + drag_vec + harvesting_force_vec, fwd_vec)
  end
  try
    find_zero(fun, initial_speed)
  catch
    NaN
  end

end


function maximum_possible_harvesting_force(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
  fun = function(harvesting_force)
    find_equilibrium_kite_speed(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area, harvesting_force)
  end
  max_harvesting_force = find_zero(fun, 0.0)
end


function lambda_power_sweep(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
  max_h_f = maximum_possible_harvesting_force(psi, wind_speed, elevation_angle,
lift_coefficient, drag_coefficient, density_of_air, wing_area)
  harvesting_forces = 0:(max_h_f * 0.90)
  kite_speeds = max.(0.0, find_equilibrium_kite_speed.(psi, wind_speed, elevation_angle,
lift_coefficient, drag_coefficient, density_of_air, wing_area, harvesting_forces))
  shaft_tensions = shaft_tension.(psi, kite_speeds, wind_speed, elevation_angle,
lift_coefficient, drag_coefficient, density_of_air, wing_area)
  lambdas = lambda.(psi, kite_speeds, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area, harvesting_forces)
  (lambdas, kite_speeds .* harvesting_forces)
end


println("plot_power_output_vs_lambda_at_different_windspeeds()")
function plot_power_output_vs_lambda_at_different_windspeeds()
  psi = pi / 2
  wind_speeds = 10.0:5.0:30.0
  elevation_angle = pi / 3
  lift_coefficient = 1.5
```

```
  drag_coefficient = lift_coefficient / 20.0
  density_of_air = 1.225
  wing_area = 3.0

  p = plot()
  for wind_speed = wind_speeds
    (lambdas, powers) = lambda_power_sweep(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
    plot!(p, lambdas, powers .* 0.001, lab = "$wind_speed m/s", xlabel = L"\Lambda", ylabel =
"power [kW]")
  end
  plot!(p, xlims = (0, 0.04), title = L"Power vs $\Lambda$ at wind speeds")
end


println("plot_power_output_vs_lambda_at_different_elevations()")
function plot_power_output_vs_lambda_at_different_elevations()
  psi = pi / 2
  wind_speed = 12.0
  elevation_angles = 0:10:60
  lift_coefficient = 1.5
  drag_coefficient = lift_coefficient / 20.0
  density_of_air = 1.225
  wing_area = 3.0

  p = plot()
  for elevation_angle = elevation_angles
    (lambdas, powers) = lambda_power_sweep(psi, wind_speed, deg2rad(elevation_angle),
lift_coefficient, drag_coefficient, density_of_air, wing_area)
    plot!(p, lambdas, powers .* 0.001, lab = "$(elevation_angle)¬∞", xlabel = L"\Lambda",
ylabel = "power [kW]")
  end
  plot!(p, xlims = (0, 0.04), title = "Power vs lambda at elevation angles")
end


println("plot_power_output_vs_lambda_at_different_glide_numbers()")
function plot_power_output_vs_lambda_at_different_glide_numbers()
  psi = pi / 2
  wind_speed = 12.0
  elevation_angle = pi / 3
  lift_coefficient = 1.5
  glide_numbers = 5:5:30
  density_of_air = 1.225
  wing_area = 3.0

  p = plot()
  for glide_number = glide_numbers
    (lambdas, powers) = lambda_power_sweep(psi, wind_speed, elevation_angle, lift_coefficient,
lift_coefficient / glide_number, density_of_air, wing_area)
    plot!(p, lambdas, powers .* 0.001, lab = "$(glide_number)", xlabel = L"\Lambda", ylabel =
"power [kW]")
  end
  plot!(p, xlims = (0, 0.04), title = L"Power vs $\Lambda$ at different $\frac{C_L}{C_D}$")
end



println("plot_power_output_vs_lambda()")
function plot_power_output_vs_lambda()
  psi = pi / 2
  wind_speed = 10.0
  elevation_angle = pi / 3
  lift_coefficient = 1.5
  drag_coefficient = lift_coefficient / 20.0
  density_of_air = 1.225
```

```
  wing_area = 3.0

  (lambdas, powers) = lambda_power_sweep(psi, wind_speed, elevation_angle, lift_coefficient,
drag_coefficient, density_of_air, wing_area)
  plot(lambdas, powers .* 0.001, lab = "", xlabel = L"\Lambda", ylabel = "power [kW]")
end



nothing
```

Julia code: zero_drag_kite_shaft_length.jl

```
using Plots
using LaTeXStrings
using Roots


println("plot_for_3_mm_tether()")
function plot_for_3_mm_tether(; nondimensional = false, scale = 1.0)
  c_d_t = 1.1;
  rho = 1.225;
  d = 0.003; # 3 mm diameter should yield approx 200 m tether, 5-10 m wingspan
  k_r = 3.0;
  r_0 = 8;
  r_1 = r_0 * k_r;
  safety_factor = 4.0
  glide_ratio_kite0 = 8.0;
  glide_ratio_kite1 = 12.0;
  glide_ratio_kite2 = 20.0;
  nominal_wind = 9.0;
  elevation = 35.0;
  target_glide_ratio = 8


  function belly_drag_shaft(rho, c_d_t, r_0, r_1, mu, d, l, omega, tension)
    -rho * c_d_t * (((r_0 ^ 2 + r_1 ^ 2) * cos(omega * sqrt(mu) * l * tension ^ (-1//2)) - 2 *
r_0 * r_1) * sqrt(tension) * sin(omega * sqrt(mu) * l * tension ^ (-1//2)) + 2 * (r_0 * r_1 *
cos(omega * sqrt(mu) * l * tension ^ (-1//2)) - r_0 ^ 2 / 2 - r_1 ^ 2 / 2) * l * sqrt(mu) *
omega) * mu ^ (-1//2) * d * omega / sin(omega * sqrt(mu) * l * tension ^ (-1//2)) ^ 2 / 4
  end


  function strength_of_uhmwpe_tether(diameter)
    diameter ^ 2 / 0.004^2 * 21_000 # using DSK90 4 mm as reference
  end


  function mass_per_meter_uhmwpe(d)
    DENSITY_OF_UHWMPE_TETHER = 711.4999626069974
    (d / 2)^2 * pi * DENSITY_OF_UHWMPE_TETHER
  end


  function calc_glide_ratio(rho, c_d_t, r_0, r_1, d, l, safety_factor, glide_ratio_kite)
    mu = mass_per_meter_uhmwpe(d)
    lift = strength_of_uhmwpe_tether(d) / safety_factor
    function objective_fun(glide_ratio)
      flying_speed = nominal_wind * glide_ratio * cos(deg2rad(elevation)) * 2 / 3 # 2/3 due to
induction factor optimal
      omega = flying_speed / r_1
      tether_drag = belly_drag_shaft(rho, c_d_t, r_0, r_1, mu, d, l, omega, lift)
      kite_drag = lift / glide_ratio_kite
      glide_ratio - lift / (tether_drag + kite_drag)
    end
    tmp = find_zero(objective_fun, 15.0)
    tmp < 3 ? NaN : tmp
  end


  function plot_nondimensional()
    formatter_fun = x -> "$(Int(round(x / 1000)))k"
    let lengths = 60:300
      lift = strength_of_uhmwpe_tether(d * scale) / safety_factor
      glide_ratios = calc_glide_ratio.(rho, c_d_t, r_0 * scale, r_1 * scale, d * scale,
lengths .* scale, safety_factor, Inf)
```

```
      glide_ratios_k0 = calc_glide_ratio.(rho, c_d_t, r_0 * scale, r_1 * scale, d * scale,
lengths .* scale, safety_factor, glide_ratio_kite0)
      glide_ratios_k1 = calc_glide_ratio.(rho, c_d_t, r_0 * scale, r_1 * scale, d * scale,
lengths .* scale, safety_factor, glide_ratio_kite1)
      glide_ratios_k2 = calc_glide_ratio.(rho, c_d_t, r_0 * scale, r_1 * scale, d * scale,
lengths .* scale, safety_factor, glide_ratio_kite2)
      lambdas = 1 ./ 2 ./ glide_ratios
      lambdas_k0 = 1 ./ 2 ./ glide_ratios_k0
      lambdas_k1 = 1 ./ 2 ./ glide_ratios_k1
      lambdas_k2 = 1 ./ 2 ./ glide_ratios_k2
      lambda_max = r_0 ./ sqrt.(lengths.^2 .- r_0.^2 .- r_1.^2)
      plot(lengths ./ d
           , [lambdas lambdas_k0 lambdas_k1 lambdas_k2]
           , ylabel = L"\Lambda"
           , xlabel = L"tether length $\varrho = \frac{l}{d}$"
           , title = L"non-dimensional $\Lambda$ vs tether length, dimensioned for $w$ = %$
(nominal_wind) m/s"
           , lab = ["zero kite drag" "tether + kite glide r $(glide_ratio_kite0)" "tether + kite
glide r $(glide_ratio_kite1)" "tether + kite glide r $(glide_ratio_kite2)"]
           #, legend = false
           , ylims = (0, 0.10)
           , xlims = (0, Inf)
           , size = (1000, 1000)
           , xformatter = formatter_fun
           , lc = [:black :green :brown2 :royalblue]
          )
      plot!([0; maximum(lengths)] ./ d, [1; 1] ./ 2 ./ target_glide_ratio, lc = :gray, linestyle
= :dash, lab = "glide ratio $(target_glide_ratio):1")
      plot!(lengths ./ d, lambda_max, lc = :red, linestyle = :dash, lab = L"maximum $\Lambda$")
    end
  end


  function plot_3mm()
    let lengths = 60:300
      lift = strength_of_uhmwpe_tether(d) / safety_factor
      glide_ratios = calc_glide_ratio.(rho, c_d_t, r_0, r_1, d, lengths, safety_factor, Inf)
      glide_ratios_k0 = calc_glide_ratio.(rho, c_d_t, r_0, r_1, d, lengths, safety_factor,
glide_ratio_kite0)
      glide_ratios_k1 = calc_glide_ratio.(rho, c_d_t, r_0, r_1, d, lengths, safety_factor,
glide_ratio_kite1)
      glide_ratios_k2 = calc_glide_ratio.(rho, c_d_t, r_0, r_1, d, lengths, safety_factor,
glide_ratio_kite2)
      lambdas = 1 ./ 2 ./ glide_ratios
      lambdas_k0 = 1 ./ 2 ./ glide_ratios_k0
      lambdas_k1 = 1 ./ 2 ./ glide_ratios_k1
      lambdas_k2 = 1 ./ 2 ./ glide_ratios_k2
      lambda_max = r_0 ./ sqrt.(lengths.^2 .- r_0.^2 .- r_1.^2)
      plot(lengths
           , [lambdas lambdas_k0 lambdas_k1 lambdas_k2]
           , ylabel = L"\Lambda"
           , xlabel = L"tether length $l$"
           , title = L"$d$ = %$(Int(d * 1000)) mm, $r_0$ = %$(r_0) m and $r_1$ = %$(r_1) m, $w$
= %$(nominal_wind) m/s"
           , lab = ["zero kite drag" "tether + kite glide r $(glide_ratio_kite0)" "tether + kite
glide r $(glide_ratio_kite1)" "tether + kite glide r $(glide_ratio_kite2)"]
           #, legend = false
           , ylims = (0, 0.10)
           , xlims = (0, Inf)
           , size = (1000, 1000)
           , lc = [:black :green :brown2 :royalblue]
          )
      plot!([0; maximum(lengths)], [1; 1] ./ 2 ./ target_glide_ratio, lc = :gray, linestyle
= :dash, lab = "glide ratio $(target_glide_ratio):1")
      plot!(lengths, lambda_max, lc = :red, linestyle = :dash, lab = L"maximum $\Lambda$")
    end
```

```
  end


  if (nondimensional)
    plot_nondimensional()
  else
    plot_3mm()
  end
end


nothing
```

Julia code: double_section.jl

```
using Roots
using Plots
using Rotations
using LinearAlgebra
using LaTeXStrings


function lambda_helper(r0, r1, l, theta)
  sin(theta) * r0 / sqrt(2 * cos(theta) * r0 * r1 + l^2 - r0^2 - r1^2 + 0im)
end


function lambda(r0, r1, l, theta)
  tmp = lambda_helper(r0, r1, l, theta)
  isreal(tmp) ? real(tmp) : NaN
end


function lambda_error(r0, r1, r2, l0, l1, theta0, theta1)
  lambda0 = lambda_helper(r0, r1, l0, theta0) * r1 / r2
  lambda1 = lambda_helper(r1, r2, l1, theta1)
  (real(lambda0 - lambda1))^2 + imag(lambda0)^2 + imag(lambda1)^2
end


function relax_twist(r0, r1, r2, l0, l1, theta)
  result = find_zero(theta0 -> lambda_error(r0, r1, r2, l0, l1, theta0, theta - theta0), theta *
l0 / (l0 + l1))
end


function find_double_section_lambda(r0, r1, r2, l0, l1, theta)
  # like before, but check if tether force is pointing in or out by using
  # vector math
  #
  # - calc vectors
  # - calc tensions
  # - calc sum of forces
  # - dot product with outward vector
  # - bigger than 0 -> ok, else NaN
  # point are x0, x1, x2
  try
    theta0 = relax_twist(r0, r1, r2, l0, l1, theta)
    theta1 = theta - theta0
    x0 = RotZ(-theta0) * [r0; 0; 0]
    x1 = [r1; 0; 0]
    x2 = RotZ(theta1) * [r2; 0; 0]

    # assume T_N = 1.0
    tension0 = l0 / sqrt(2 * cos(theta0) * r0 * r1 + l0^2 - r0^2 - r1^2)
    tension1 = l1 / sqrt(2 * cos(theta1) * r1 * r2 + l1^2 - r1^2 - r2^2)

    force0 = normalize(x0 - x1) * tension0
    force1 = normalize(x2 - x1) * tension1

    radial_force = dot(force0 + force1, [1.0; 0.0; 0.0])

    if isreal(tension0) && isreal(tension1) && radial_force >= 0
      lambda(r1, r2, l1, theta - theta0)
    else
      # soft bridle acts compressively, or invalid geometry
      NaN
    end
```

```
  catch
    NaN
  end
end


function find_possible_buckling_theta(r0, r1, r2, l0, l1)
  try
    (true, find_zero(th -> lambda(r0, r2, l0 + l1) - find_double_section_lambda(r0, r1, r2, l0,
l1, th), pi/2))
  catch
    (false, NaN)
  end
end


# find the point where the two section shaft buckes. This should be the same
# point where the single segment shaft without the bridle has the same lambda
# value
function max_lambda(r0, r1, r2, l0, l1; accuracy = 0.001)
  (buckling, theta) = find_possible_buckling_theta(r0, r1, r2, l0, l1)
  theta_max = buckling ? theta : deg2rad(180)
  res = if buckling
    lambda(r0, r2, l0 + l1, theta)
  else
    tmp = maximum(filter(x -> !isnan(x), vcat(-1, find_double_section_lambda.(r0, r1, r2, l0,
l1, 0:accuracy:theta_max))))
  end
  res >= 0 ? res : NaN
end


function do_plot(; length = 20, r0 = 1, r1 = 1, r2 = 4, accuracy = 0.001)
  let ll = 0:0.1:length
    plot(ll
         , max_lambda.(r0, r1, r2, ll, length .- ll, accuracy = accuracy)
         , lab=""
         , xlims = (0, length)
         , ylims = (0, Inf)
         , xlabel = "section split point"
         , ylabel = L"max $\Lambda$ possible"
        )
  end
end
```