On the Segmented Helix Formed by Stacking Similar Objects

Robert L. Read [*]

Founder, Public Invention, an educational non-profit.

January 30, 2019

Abstract

In nature, helical structures arise when identical structural subunits combine sequentially, the orientational and translational relation between each unit and its predecessor remaining constant. A helical structure is thus generated by the repeated action of a screw transformation acting on a subunit.[1]

Contents

[*]read.robert@gmail.com

1 Introduction

Note: Much of this paper is pre-dated by work by Peter Kahn[2].

During the Public Invention Mathathon of 2018, software was created to view tetrahedra joined face-to-face in chains. It was noted by the participants that when the rules for which face to add the next tetrahedron to were periodic, the resulting chain was always a helix or a torus. A torus is a degenerate helix.

After some relfeciton, it became clear that any stack of objects of the same length joined at the same angles to each other repetitively form a helix. A specific example is a face-to-face connection of physical objects of the same length where the angles of the two joining faces relative to each other control the angle and rotation of the axis at each joint. We have not yet found this simple idea articulated elsewhere. The purpose of this paper is to prove and to provide formulae for the resulting helix.

In engineering, sometimes the term "special helix"[3] is use for helical curves on non-cylindrical surfaces. This paper use the term "helix" only in the sense of "cylindrical helix".

2 A Warm-up: 2Dimensions

Considering the problem in two dimensions may be a valuable introduction. Suppose that we consider a polygon that that as two edges, called A and B, and that we define the length L of the polygon as the distance between the midpoints of these edges. Suppose that we are only allowed to join these polygons by aligning A of one polygon to B of another polygon, with their midpoints coincident. Let us further assume

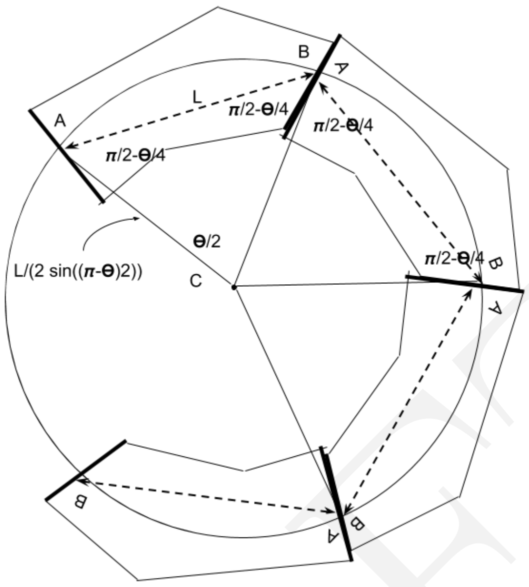


Figure 1: The rotatable prism of three objects

that we disallow inversions of the polygon. Let us imagine that we have a countable number of polygons P_i indexed from 0. Then what shapes can we make by chaining these polygons together?

Each joint J_i between polygons P_i and P_{i+1} will place the axes of at the same angle, θ, since our polygons do not change shape. Let us define θ to be positive if we move anti-clockwise from P_i to P_{i+1} and negative if we move clockwise. If $\theta = 0$, the joints will be collinear.

If $\theta \neq 0$, it seems they polygon joints will always lie on a circle. A proof of this is that each polygon has associated with it an isoceles triangle A, B, C, where $\angle CBA = \angle CAB = \theta/2$, and $\angle ACB = (\pi - \theta)$. AC and BC are not necessarily aligned with an edge of the polygon. The length AB is L, and the lengths AC and BC are $(L/2)/\sin(\pi - \theta)/2$. In any chain of polygons, these triangles all meet at point C, and there all joints are on the circle centered at C with radius $\frac{L}{2\sin\frac{\theta}{2}}$.

An analogous, though far more complicated, result holds in three dimensions.

3 The Segmented Helix

In this section we consider segmented helix, or a helix evaluated only at regular points. Drawing lines between these points create 3D polygon.

Following the Wikipedia artilce `https://en.wikipedia.org/wiki/Helix`, we set up a helix parametrically.

$$P_x(t) = r \sin t$$
$$P_y(t) = r \cos t$$
$$P_z(t) = bt$$

Such a helix has a radius of r and slope (if $r \neq 0$) of b/r,. Note that a helix may be degenerate in two ways. If $r = 0$, these equations become a line. If $b = 0$, these equations describe a circle in the xy-plane. If $r = 0$ and $b = 0$, the figure is a point.

Such helixes are continuous, but we are investigating stacks of discrete objects. We in fact wish to derive the parameters for a continuous helix from such discrete objects which constrain discrete points, so we wish to study a helix evaluated at integral points. We call such an object a *segmented helix* or *shelix (SHELL-lix)*.. A segmented helix may be thought of as function that given an integer gives back a point in three space.

$$P_x(n) = r \sin n\theta$$
$$P_y(n) = r \cos n\theta$$
$$P_z(n) = nd$$

d is the distance along the z-axis between adjacent joints, and θ is the rotation around the z-axis between adjacent points. r is the radius of the shelix. If we think of the shelix as describing a polyline in 3-space, we would like to investgiate the properties of that polyline.

- L is the distance between any two adjacent points.
- c is the length of a chord formed by the projection of the segment between two points into the xy-plane.
- ϕ is the angle in the plane containing two joints which is perpendicular to xy-plane (in other words the plane parallel to the z axis containing two points.)

These quantities are related:

$$c = 2r \sin \frac{\theta}{2} \tag{1}$$
$$L^2 = c^2 + d^2 \tag{2}$$
$$\arctan \frac{c}{d} = \phi \tag{3}$$

Now we are attempting to relate these properties to properties intrinsic to the joint or interface between two segments or objects in the shelix. If given an object, the length between the joint points L is easily measured.

4 The Intrinsic Properties of Stacking Objects

If we are given a physical or mathematical object that stacks, the faces may be specified by a normal vector or by the angles of the face relative to the axis. The normal vector

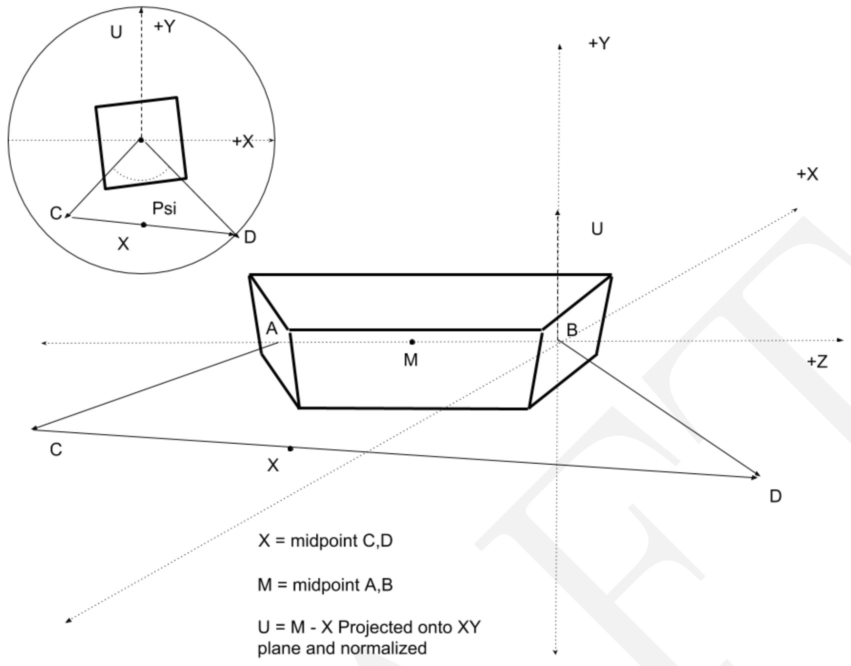


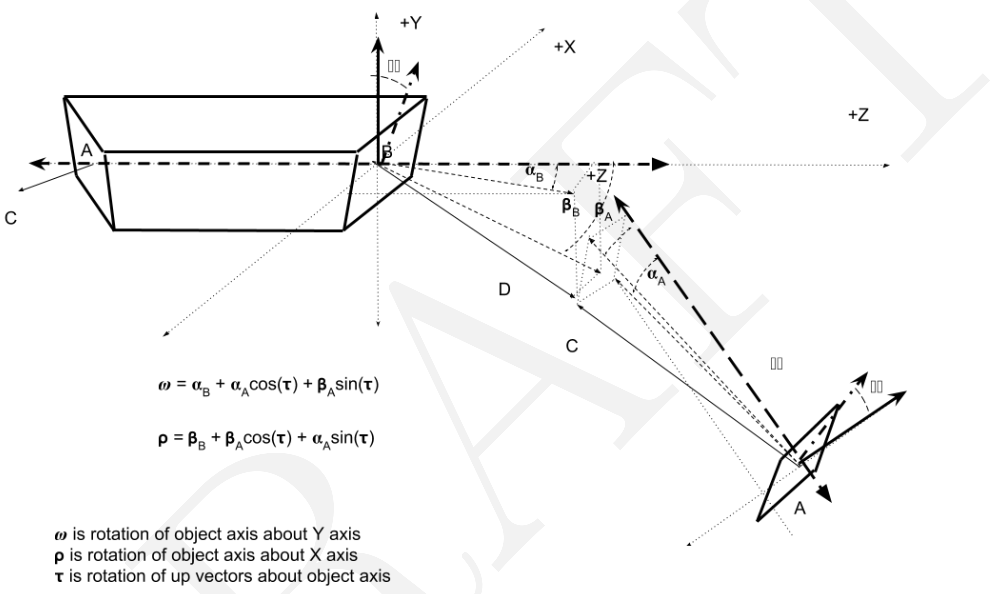
Figure 2: The rotatable prism of three objects

is a more linear-algebraic approach; the face angles may be more natural to chemists and mechanical engineers. Since the two methods are interchangable, it is largely a matter of convenience. However, we seek specifically to develop a formula which allows helices to be designed, and the face-angle approach seems more suited to the carpenter grasping a hand saw or the molecular biologist designing a molecule.

As shown in figures and , We place a joint point on each face, and call these points A and B. We define the axis of the object as $\overline{AB}$. Imagine the object placed on the z axis in a right-handed coordinate system, so that A is in the negative z direction and B is at the origin. Then the cut of the B face can be described by two angles. α is the angle in the XZ plane, and β is the angle in the YZ plane. In other words, if a box or cuboid were drawn with three edges aligned with the axis and at the origin, and the vector defining the face-normal defined the diagonal of the box or cuboid, α is the angle with the z-axis of the face diagonal in the XZ plane (or the projection of the body diagonal into that plane), and β is the angle of the z-axis with the YZ-plane.

The face angles for A are denoted α_A, β_A, and likewise the independent face angles for B are α_B, β_B.

In the angle method, we start with these intrinsic properties of an object, and additionally the rule for how objects are laid face-to-face. That is, knowing the length between two joint points and a vector normal to the faces of the two joints, we almost have enough to determine the unique stacking of objects. The final piece is that we must know the *twist*. That is, when face A of a second objects is placed on face B of a



$$\omega = \alpha_B + \alpha_A \cos(\tau) + \beta_A \sin(\tau)$$

$$\rho = \beta_B + \beta_A \cos(\tau) + \alpha_A \sin(\tau)$$

ω is rotation of object axis about Y axis
ρ is rotation of object axis about X axis
τ is rotation of up vectors about object axis

Figure 3: Joint Geometry

first object so that they are flush (that is, their normals are in opposite directions), it remains the case that the second object can be rotated about the normals. To define the joining rule, we must attach an *up vector* to each object. Then a joining rule is "place the second object against the first, joint point conincident to joint point, and twist it so that its up vector differs by τ degrees from the up vector of the first object."

- An object with two identified faces, labeled F_A and F_b. These faces are in fact planes, with $\alpha_A, \beta_A, \alpha_B, \beta_B$, specifying the angular separation of the normal from the XZ and YZ planes. We assume that normals point out of the object rather than in. The angles in the method may be considered the angles of the face diagonals of the axis-aligned cuboid whose body digaonal is the vector normal.
- The length L of an object, measured from joint point A to joint point B.
- A joint twist τ defining the change in computed up-vector between objects, mesured at the joint face.

4.1 Relating Face Angles to Face Normal

Although intercomputable, we certainly want to be able to compute the coordinates or the point C_1 (the joint point of the second object), or equivalently, the axis vector of the second object.

This depends not solely on one face, but on how they combine, and in particular how they combine via the twist τ. We use ω and ρ to capture this combination.

A basic algorithmic approach to this to compute the unit vectors normal to faces N_B and N_A, and let A, B, C, D be four contigous points. Compute the rotation that aligns N_B in the opposite direction to N_A. Apply this rotation to the vectors AB, and translate it to the point B.

Given these angles and treating them as face diagonals, we can find the position of point A_1 via:

$$z = \frac{L}{\sqrt{1 + \tan^2 \omega + \tan^2 \rho}} \quad (4)$$

$$A[1]_x = z \tan \omega \quad (5)$$

$$A[1]_y = z \tan \rho \quad (6)$$

$$A[1]_z = z \quad (7)$$

Possibly due to my lack of imagination, these coordinates will become the basis of the formulae for r, d, and θ in the segmented helix.

4.2 Observations

Having related the face angles to positions of the points directly connected to an object on the z-axis with a joint at the origin, it now becomes more convenient to think of the object's midpoint as placed at the origin.

In the derivations below, we rely on certain facts about the segmented helix formed by the stack of objects:

- Without loss of generality, we may think of any member whoses faces and twist generate a non-degenerate helix as being "above" the axis of the helix. We furthermore choose to place the object in this figure so that $G_y = H_y$, that is, that the members are symmetrix about the z-axis.
- Every joint (E, F, G, H) is the same distance r from the axis of the helix.
- Every member is in the same angular relation to the axis of the helix.
- Since every member cuts across a cylinder around the axis, the midpoint of every memeber is the same distance from the axis which is general a little a less than r. In particular the midpoint M whose closest point on the helix axis m is on the y-axis and $\overline{Mm} < \overline{Ff}$.
- The points (e, f, g, h) on the axis closest to the joints (E, F, G, H) are eqidistant about the axis and centered about the y-axis. In particular, $\overline{fm} = \overline{gm}$

From the obeservations that $\overline{Ff} = \overline{Gg}$, $\overline{fm} = \overline{mg}$, and we concluded that the helix axis is in a plane parallel to the XZ-plane, it intersects the y-axis, but in general is not parallel to the z-axis. (Note: this assertion requires a formal proof. I can't quite produce a short proof yet.)

From these observations we may state a fundamental fact:

Observation 1. *The angle bisectors of each joint are in general skew and are closest at the axis of helix.*

(This has been asserted by most authors, an id not original to me.)

The conjecture allows us to use the standard linear algebra formula for computing the closest points of two skew lines to find two points on the axis of the helix. The distances between these points is d, and the distances between these points ad the joints is r.

Following the `https://en.wikipedia.org/wiki/Skew_lines`, we calculate:

$$\overline{X} = \overline{F} + \lambda(\overline{F} - \overline{M_{FH}} \tag{8}$$

$$\overline{Y} = \overline{G} + \lambda(\overline{F} - \overline{M_{FH}} \tag{9}$$

Note that in Figure 4 there is great room for confusion in terms of plane ω is actually measured against. The three triangles FfG, FmG, and FgG are all in general not co-planar, that is, they are all at slight angles to each other. We must remain clear that ω and ρ are measured against the FmG. The up-vector of the object lies precisely in this plane. ω must be measured against the up-vector of the object.

If the object in question is a physical macroscopic object, for example made out of wood or plastic, then making physical measurement against it is not likely to be confusing so long as we physically distinguish which was is considered up.

If the object is a mathematical object, there is much greater change for confusion.

Wolfram Alpha provides that:

$$\sin \arctan x = \frac{x}{x^2 + 1} \tag{10}$$

Whe we can reasonably hope to use to produce a simplified form of d.

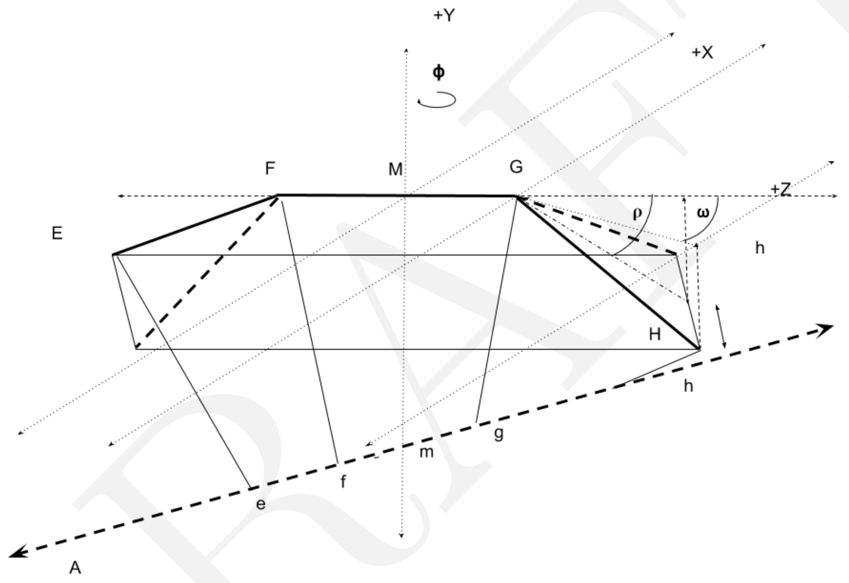


Figure 4: Three Members

From our previous calculations based on face angles, we can concluded that:

$$q = \frac{L}{\sqrt{1+\tan^2\omega+\tan^2\rho}} \tag{11}$$
$$L = \frac{\sqrt{1+\tan^2\omega+\tan^2\rho}}{q} \tag{12}$$
$$H_x = q\tan\omega \tag{13}$$
$$H_y = q\tan\rho \tag{14}$$
$$H_z = q + \frac{L}{2} \tag{15}$$

Combining with the simple fact that:

$$F_x = 0 \tag{16}$$
$$F_y = 0 \tag{17}$$
$$F_z = -\frac{L}{2} \tag{18}$$

We conclude that:

$$q = \frac{L}{\sqrt{1+\tan^2\omega+\tan^2\rho}} \tag{19}$$
$$M^{fh} = \frac{F+H}{2} \tag{20}$$
$$M^{fh}_x = \frac{q\tan\omega}{2} \tag{21}$$
$$M^{fh}_z = q \tag{22}$$
$$G_z = L/2 \tag{23}$$
$$\tag{24}$$

And therefore:

$$\chi = \arctan\frac{\frac{q\tan\omega}{2}}{L/2-q} \tag{25}$$
$$\chi = \arctan\frac{\frac{q\tan\omega}{2}}{\frac{L-2q}{2}} \tag{26}$$
$$\chi = \arctan\frac{q\tan\omega}{L-2q} \tag{27}$$
$$\chi = \arctan\frac{q\tan\omega}{q\frac{L}{q}-2q} \tag{28}$$
$$\chi = \arctan\frac{\tan\omega}{\frac{L}{q}-2} \tag{29}$$
$$\tag{30}$$

Since:

$$d = L \sin \chi \tag{31}$$

$$\tag{32}$$

and Wolfram Alpha provides that:

$$\sin \arctan x = \frac{x}{x^2 + 1} \tag{33}$$

Whe we can reasonably hope to use to produce a simplified form of d.

$$d = L \sin \arctan \frac{\tan \omega}{\frac{L}{q} - 2} \tag{34}$$

$$\tag{35}$$

This needs to be checked—however, if this is true, it is pretty extraordinary. Probably it is wrong.

5 The 4-point Algorithm

In this section, give a good drawing of the 4-point algorithm, and a linear algebra explanation.

6 Applying Symmetry to the 4-point Algorithm

In this section explain how many symmetries in the system allows for significant simplifcations.

The balancing of the face normals opens the possibility of representing the object even more compactly:

- The angle between the face normals,
- The ratio of the projection of face normals of the same length.

This should provide all the information needed to completely specify the object. If this is true, we have acheived a major simplification.

7 Checks and Explorations

In this section add graphs. Also, a check against the BC helix. Possibly software should be used to produce a 3D simulation of the issues.

8 New Section

From the normals, we can compute an *up vector* which is intrinsic to the object. We could think of the up vector as a fine mark made on each face in the same direction (if projected onto a plane perpendicular to the axis.) When face F_A of object $N+1$ is placed against F_B of object N, object $N+1$ is twisted until the mark on F_A angles τ away from the mark on object N, measured anticlockwise looking from A to B on object N.

To define the up vector, place the object so that joint B is at the origin and the axis is aligned with the z-axis, with the point A on the negative z-axis. The projection of the face normals form a minimum angle less than or equal to π radians. Rotate the object about the z-axis so that the projections of the face normals form equal angles with the negative y-axis. Then define the direction straight up the y axis to be the up vector. If we had a physical object, we might mark the face normals with an arrow to mark the up vectors.

If one of the face normals is aligned with object axis, the up-vector is the opposite of the other face normal. (If both face normals are aligned with the axis of the object, then shelix has zero radius and $d = L$, so we need do no further math.)

Mathematically, we will treat the up vector as unit-vector.

$$M = \frac{A+B}{2} \tag{36}$$

$$X = \frac{(A+F_a)+(B+F_b)}{2} \tag{37}$$

$$U = \frac{M-X}{\|M-X\|} \tag{38}$$

The up vector is not defined when the face norms point in precisely opposite directions, in which case $d = L$.

In the placement of the first object described above, it would be nice to know where the position of point B on the second object is. This can be obtained by computing the Euler angles of the face normal N_A at the origin, and then simply adding the negation of the Euler angles N_B, and using the resulting angles to compute a rotation $R(N_A, N_B)$ to be a applied to a vector of length L, and then performing the twist by $T(\tau)$. This process can be repeated at the point B_1 (though no longer at the axis) to generate four or more points. In fact the entire process of constructing the stack could be thought of a series of "rotate then translate" steps by the composition of the rotation matrixes RT applied to a vector $\overline{AB}$ and the up vector. However, it is a tiny bit of work to compute this given only the intrinsic properties of the object and joint.

Note: we have shown we can compute from 4 points. More elegant however, would be to construct a similar construction using the axis-intersecting lines without computing 4 points, perhaps from a pure geometry in some way.

TODO: Actually giving the transformation matrix here would be nice. How would we test?

TODO: Compute the vector angle between axes here. In this positioning, compute where point B of the second object will be.

Note: The twist is not the same as θ. When $\tau = 0$, $\theta = \alpha$, I think. Possible $\theta = \alpha+$ some function of τ.

Note: Possibly theta goes down as tau goes up.

Note: Considering the pitch of the segmented helix may be useful in relating variables.

Note: I now think we can think in terms of the skew lines (the joint lines), and I believe τ is directly the angle formed with the up vector, that the up-vector aims at the mid point of the axis between the intersection skew line points, and that angle bend α (or $(\pi - alpha)/2$) let's us determine the skew line as well. So we have the mid point, and we have another line based purely on τ and α, so that may allow θ to be computed more simply than the skew line intersection construction.

Todo: Try to do my 4-point solution with an Midpoint and upvector solution.

Todo: Try to prove that τ is really either the angle bisector angle or twice it.

Todo: Create a definition of the "sidedness" of shelix, in terms of the number of sides in one pitch, and relate this matehmatically to the other points.

8.1 New Insight

Having arranged the objects as in our diagrams and identified the "prism" formation, it becomes clear that we can operate in the projection plane (XZ). This allows considerable simplication.

Our basic plan is to contruct the midpoint near F, M_F, which is in fact the midpoint of E and G. The line hits the axis of the helix perpendicularly. The midpoint of E adn F also hits this axis line, (and is the origin of the XZ plane.) Therefore, if we find the perpendicular distance from the the $\overline{F, M_F}$ line to the origin, we can easily compute the distance d along the axis, which will be slightly less than L.

9 Computation of Segmented Helix

In this section we attempt to derive formulae for $r, d, /theta$ from the instrinsic properties of the object.

10 Old Work

The most easily measured relationship between the segments is the angle α in the plane containing both of the segments. (If the segments are coincident and therefore parallel α will be π. If $\alpha < \pi/2$, the segment "bends backward". The final measure that is needed is the rotation about a member between the elements attached to it. Let us call this angle ψ. That is, if you look along the axis of an object at the line between the joints, the two members form an angle between 0 and $\pi/2$.

$$A = P(-1)$$
$$B = P(0)$$
$$C = P(1)$$

A, B, C form an isoceles triangle that contains $\overline{AB}$ and $\overline{BC}$, and therefore the angle α is in this plane. Forming the kite $ABCO$, we see that one diagonal is r and the other diagonal is $2r\sin\theta$, where $y = r\sin\theta$ is half of this diagonal.

$$\begin{aligned}
y &= r\sin\theta \\
z^2 &= c^2 - y^2 \\
z^2 &= c^2 - (r\sin\theta)^2 \\
q^2 + z^2 &= L^2 \\
q &= \sqrt{L^2 - z^2} \\
q &= \sqrt{L^2 - (c^2 - (r\sin\theta)^2)} \\
q &= \sqrt{L^2 + (r\sin\theta)^2 - (2r\sin\theta/2)^2} \\
\sin\alpha/2 &= q/L \\
q &= L\sin\alpha/2 \\
\alpha/2 &= \arcsin q/L \\
\alpha &= 2\arcsin\frac{\sqrt{L^2 + (r\sin\theta)^2 - (2r\sin\theta/2)^2}}{L}
\end{aligned}$$

Note y is in the projected kite, q is in the slanted kite. Rearranging these relationships in terms of known quantites L, α, ψ, we have:

$$\begin{aligned}
\sin\alpha/2 &= q/L \\
q &= L\sin\alpha/2 \\
z^2 &= L^2 - q^2 \\
\beta &= \frac{\pi - \theta}{2} \\
\frac{z}{c} &= \cos\beta \\
c &= \frac{z}{\cos\beta} \\
y^2 &= c^2 - z^2 \\
r &= \frac{y}{\sin\theta} \\
d^2 &= L^2 - c^2 \\
\phi &= \arctan c/d
\end{aligned}$$

I think we can compute ψ from θ easily enough. The problem is going the other way.

10.1 An alternate approach

Rather than starting with α and ψ as inputs, we can think of a vector v that moves in the positive $\overline{Z}$ direction, $\overline{Z} = \overline{P}_1 - \overline{P}_0$, and the angle θ which Z rotates about the z-axis at each step. If a rotation matrix M represents rotation about and T represents translation along the z-axis, then $P_n = (M + T)^n P_n$.

11 Reconsidering the BC Helix

In light of Lord's ovservation and the Shelix algorithm, we can now reconsider the BC Helix, and in fact a variety of segmented helices which should perhaps becalled *Platonic delices* or if you prefer *Platonic shelixes*.

This complementary view is to think of the BC Helix not as the helix that intersects the vertices of the tetrahedron as Coxeter did[4], nor a single rail as may be valuable to engneers[5], but rather as a helix through the center point of the faces of the tetrahedron. This is a shelix of very small radius compared to the other two approaches, but it has the advantage that it is far more general. For example, it is clearly defined if one used truncated tetrahedra.

More generally, the same approach gives the formula for the shelix created by placing dodecahedra or icosahedra or octahedra face-to-face in a regular pattern. The table of parameters of such delices is provided below.

Note this also makes clear that in these cases we must also specify the *twist*, even if we insist on perfect face-to-face matching. Thinking of it this way, there are actually 3 tetrahedral delices, depending on which twist is chosen (keeping the faces matching). In the case of the tetrahedron, this creates the clockwise BC Helix, the anti-clockwise BC Helix, and the not-quite-close tetrahedral torus.

In the case of the icosahedron, there are in fact many possibilities, as one need not choose the precisely opposite face as the joining face, and one may choose up to three twists.

All of this is a consequence of Lord's observation that *any* repeated transformation produces a shelix.

12 Three dimensions

12.1 New Approach

We want to compute everything from properties intrinsic to the object and the joint process. These are the length, the face normal vectors, and the joint twist. These are the inputs to our process. From this we can compute several joints by: 1) Placing the object joint-to-joint aligned along their axes; 2) Computing from the face angles the rotation that makes the normal to B the negative of the normal to A, that is, the rotation that moves A into -B. 3) Apply this rotation around the joint. 4) Rotate along the axis until the up vectors match.

The "Up vectors" are computed to be in the line between the midpoints of points pointed to by the face normals, and the mid point of the axis, projected to be normal to that axis.

Using this intrinsic method, we can get as many points as we need of the stacked objects.

From these intrinsic property, the vector angle between axes of different objects is determined.

From these points, we can easily compute a "tilt and twist" approach, which defines α as the tilt and ψ as the twist. Those two parameters now become an output of this process above. Not the "twist" is measured IN THE PLANE OF FACES.

The remaining task is to find an (r, θ) helix representation that gives us what we need.

Conjecture: the twist is the same (or at least monotone in) as θ.

Proof Sketch: Imagine moving so many times around the helix that our (x, y) coordinates are similar to your starting point. The up vector for this object must point in approximately the same direction as the up axis you started with, by symmetry. If θ is anything other an integral factor of the twist, this would be impossible.

Alternative: The up vector at the midpoint of the object must aim at the axis of the helix. This must always be true. Therefore, the

TODO: We are still fundamentally seeking r, θ from these specifications. A formula for that is of the utmost importance.

Note: If the up vectors intersect the axis of the helix, we can find two points on this line using the wikipedia article: `https://en.wikipedia.org/wiki/Skew_lines` and the distance between these two points. This will allow us to find r, I think.

Using the method from skewlines to find two nearest points, we can then solve for the distance between this line and A or B and we have r.

So, the way to test this is to generate a helix. Use the helix to generate the midpoints and up vectors. Then at least check that we can back out and obtain our same radius and theta for the shelix. Then the problem becomes the computation of the points and up vectors from the intrinsic properties of the block, which I believe to be possible. Which should I attack first?

Let us first prove that given 3 points and two up vectors we can recover the helix.

When we have a Mathematica procedure for that, then we can generate these inputs from the object inputs of $L, \overline{A}, \overline{B}$,and twist.

Is twist the same as theta? No, it is not—but they are related by $/phi$.

12.2 Old Approach

Consider a slender cylinder or prism with two faces F_0, F_1 cut at arbitrary angles to the axis of the cylinder. Joining two such cylinders at the axes by placing F_0 angainst F_1 produced a joint with a difference in angle between the axes of α and a rotation of the orienation of the faces about the axis of θ. Note that α and θ are a function of the vectors normal to the faces to be joined, but are not the same as them. α and θ are likely combinations of the face angles.

Theorem 1 (Stacking Helix)**.** *The joints of a sequential stack of objects of length L whose joints axis change by α and orientation rotate by θ are intersected by a helix of radius and pitch easily numerically computable from L, α, and θ.*

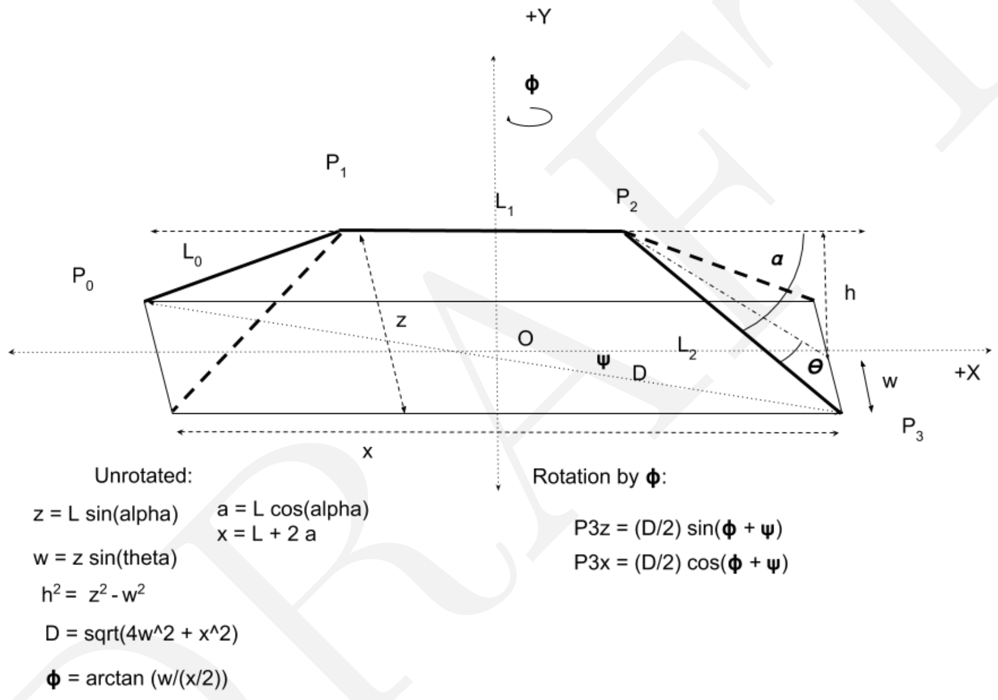


Figure 5: The rotatable prism of three objects

Proof. In Figure , let L_i be the ith instance of the length-L objects. Let α be the change in angle in the axes at a joint (the point were axes meet) measured in the plane containing the axes of both objects. (If these axes are parallel (and therefore coincident) define α to be 0. Let θ be the change in orientation relative to the axis, or, the half-angle formed by L_0 and L_2 when projected onto a plane normal to the axis of L_1. Without loss of generality, choose the measures of these angles in radians so that $0 \leq \theta < \pi/2$ and $0 \leq \alpha < \pi$. Take $\alpha < 0$ to mean that objects L_0 and L_1 bend away from each other, and $\alpha > 0$ to mean that the objects L_0 and L_1 bend toward each other.

If $\theta = 0$ and $\alpha > 0$, the stack will form a circle-like structure or radius $\frac{L}{2\sin\frac{\alpha}{2}}$, as shown in Section 2, where as if $\theta = 0$ and $\alpha < 0$, then stack will form a sawtooth-like structure.

Note that any angle α is possible, if we do not concern oursleves with the self-collision of physical objects. $\alpha > \pi/2$ means the stack "turns back on itself" to some extent.

If we arrange object L_1 so that its axis lies on the x-axis and its midpoint is at the origin, L_0 and L_1 extend from it symmetrically in the projection onto the yz-plane along the x-axis, which is always possible, and define h to be the height of the faces of L_0 and L_2 along the y-axis and w to be the distances between these faces in the z-dimesnion. Then we have:

$$\begin{aligned} z &= L\sin\alpha \\ w &= z\sin\theta \\ h &= \sqrt{z^2 - w^2} \end{aligned}$$

Now we seek the formula for the helix which intersects the joints. To find the radius of this helix, we conceptually place our three objects in a cylinder, with the axis of L_1 along the surface of the cylinder aligned with the axis. We can size this cylinder to include the joints at the extreme ends of L_2 and L_0 as well. However, the L_1 axis lies on the surface, but the axes of L_0 and L_2 in general do not. If there exists a helix which intersects all joints, all axes will cut through this cylinder in the same way, creating chords in the projection into the yz-plane of the same length. Name these chords $c_0, c_1, and c_2$.

Because we will need them later, we work out the geometry of this prism-like stack of three objects completely.

- The three objects are named L_0, L_1, and L_2. L_0 has joints P_0 and P_1, L_1 has joints P_1 and P_2, L_2 has joints P_2 and P_3.
- x is the total length of the prism along the x-axis.
- z is the length of the "face" of the prism, and the length of the chord before any rotation ϕ about the y-axis.
- w is half of the width of the prism in the z-dimension.
- h is the height of the prism in the y-dimension.

- D is the length of the diagonal from P_0 to P_3.
- ψ is the angle of $\overline{P_0P_3}$, $\arctan(w/(x/2))$.
- ϕ is the amount we will have to rotate the prism about the y-axis.

Given these defintions about our assumptions, the cord lengths are:

$$\begin{aligned}
\psi &= \arctan(w/(x/2)) \\
D &= \sqrt{4w^2 + x^2} \\
c_1 &= L\sin\phi \\
c_2 &= \sqrt{h^2 + ((1/2)(D\sin(\psi+\phi) - L\sin\phi)^2} \\
c_0 &= c_2
\end{aligned}$$

We can imagine turing our 3-object stack and simulataneously increasing the size of our intersecting cyliner. If we turn the stack about the y-axis by ϕ degrees and keep the cylinder intersection the two faces of L_1, then the length of the L_1 chord will gradually increase. At the same time, the L_0 and L_2 chords will change their length, possibly increasing or decrasing. When $\phi = 0$, $c_0 = z, c_1 = 0, c_2 = z$.

Equating these quantities to find when $c_2 = c_1$, we have a trigonometric equation with a single unknown, ϕ.

$$L\sin\phi = \sqrt{h^2 + ((1/2)(D\sin(\psi+\phi) - L\sin\phi)^2}$$

Although Mathematica does not appear to be able to solve this equation, it appears to be a smooth equation in the variable ϕ and we believe from the physical structure of the problem that will have only a single solution, so we can solve this numerically with a Newton-Raphson solver easily.

Thus, by rotating our stack of objects ϕ degrees around the y-axis all four faces of our three objects intersect a cylinder on its surface with equal rotational and axial distance. The axial distance between any two joints on the same object is $L\cos\phi$, and the length of the projected chord is $L\sin\phi$.

The points P_0, P_1, P_2, and P_3 now exist on a cylinder or unknown radius parallel to x-axis, and are evenly spaced along and evenly rotated about the axis of the cylinder. The joints points thus coincide with a general helix.

Let us choose our coordinate system so that the x-axis corresponds to the axis of the helix. The general equation for the helix is:

$$\begin{aligned}
P_x(n) &= \kappa t \\
P_y(n) &= r\cos t \\
P_z(n) &= r\sin t
\end{aligned}$$

We seek to discover r and κ based on our knowledge of P_3 and P_2. In particular, we can deduce from the axial spacing there exists some t_0 such that $P_2 = P(t_0)$ and $P_3 = P(3t_0)$. Since we know that after rotation that:

$$\begin{aligned}P_{3z} &= (D/2)\sin\phi + \psi\\ P_{3z} &= r\sin 3t_0\\ P_{2z} &= \sin\phi\\ P_{2z} &= r\sin t_0\end{aligned}$$

We can use symbolic computation to solve this system of 2 equations and 2 unkowns:

$$\begin{aligned}r\sin 3t_0 &= (D/2)\sin\phi + \psi\\ r\sin t &= \sin\phi\end{aligned}$$

Defining symbols:

$$\begin{aligned}E &= (D/2)\sin\phi + \psi\\ F &= \sin\phi\end{aligned}$$

Wolfram alpha solves the system:

$$\begin{aligned}r\sin 3t_0 &= E\\ r\sin t_0 &= F\end{aligned}$$

giving the result ($3F \neq E$ and $F \neq 0$), and ignoring multiples of 2π in t:

$$\begin{aligned}r &= \frac{2F^{\frac{3}{2}}}{\sqrt{3F-E}}\\ t_0 &= -2\arctan x\frac{r+\sqrt{-\frac{F^2(F+E)}{E-3F}}}{F}\end{aligned}$$

From which, using $P_{2x} = L/2 = \kappa t_0$, we conclude:

$$\begin{aligned}r &= \frac{2F^{\frac{3}{2}}}{\sqrt{3F-E}}\\ \kappa &= \frac{L}{4\arctan\frac{r+\sqrt{-\frac{F^2(F+E)}{E-3F}}}{F}}\end{aligned}$$

. Putting this all together we have:

$$
\begin{aligned}
a &= L\cos\alpha \\
x &= L + 2a \\
z &= L\sin\alpha \\
w &= z\sin\theta \\
h &= \sqrt{z^2 - w^2} \\
D &= \sqrt{4w^2 + x^2} \\
\phi &= \arctan\frac{z}{L} \\
E &= (D/2)\sin\phi + \psi \\
F &= \sin\phi \\
r &= \frac{2F^{\frac{3}{2}}}{\sqrt{3F - E}} \\
\kappa &= \frac{L}{4\arctan\frac{r+\sqrt{-\frac{F^2(F+E)}{E-3F}}}{F}}
\end{aligned}
$$

Using these values derived exclusively from the inputs $L, \alpha, and \theta$, we can evalute the formula for the general helix only and integral values of n to obtian a formula for precise the joint points of this any such stack.

$$
\begin{aligned}
P_x(n) &= \kappa t0(1 + 2n) \\
P_y(n) &= r\cos t0(1 + 2n) \\
P_z(n) &= r\sin t0(1 + 2n)
\end{aligned}
$$

$\square$

13 Checking against comprehensible values

Unfortunately, the complexity of these formualae exceed the author's comprehension. However, we may check these formuale by graphing them against comprehensible examples. Obvious examples are extreme solutions, where α and θ are 0 or $\pi/2$, for example. We also have the particular non-trivial example of the Boerdick-Coxeter tetrahelix, formed by regular tetrahedra, which has been studied enough to have a known pitch.

14 Implications

One of the implcations of having a formulaic understanding of the math is that it may be possible to design helixes of any radius and pitch by designing periodic (possibly scalene) segments. Combined with slight irregularites, this means that you have a basis of design molecular helices out of "atoms" which correspond to our objects.

This would mean that if you wanted to build a brace of length exactly 3 meters with bars of exactly 1/2 meter you would be able to come as close to this as mathematically possible.

15 Applied to Periodic Regular Simplex Chains

Corollary 1. *Every regular simplex chain formed by a periodic generator has a helical structure.*

Prove or disprove that *every* periodic 3D Generator generates a figure contained within a cylinder of unbounded length but bounded diameter.

Note: Every example that we have tested exhibits this property. We believe it is a property of any repeated structure, not related to simplices. However, we do not yet know the name of this theorem or principle. We conjecture that every stack of repeated truncated prisms forms a helical aperigon, which is hinted at but not stated in the Wikipedia article`https://en.wikipedia.org/wiki/Skew_apeirogon`.

Note: Rob believes a proof that any periodic structure fits within a cylinder is possible, and that it should be possible to give a formulaic bound on the diameter of this cylinder (under some assumptions.) The key to the proof is to use symmetry and focus on the the center of three such objects, observing that the other two must necessarily bend towards or away from each other in a way describable by two angles. A formula for the cylinder as a function of these angles would convincingly complete the proof.

16 Future Work

We propose that the math developed in this paper can be used to build an exhaustive table of the properties Platonic delices, that is, segmented helices constructed solely out of regular Platonic solids. Such tetrahelices, icosahelices, octahelices and dodechelices have been mentioned in a number of papers[6, 7, 8], but not exhaustively studied in the purely helical form. Because in some cases Platonic delices may be found in nature or related to structures found in nature[9], it would be conveneient to have a table, and images, of all such Platonic delices for reference.

Note: Must read this: `https://www.researchgate.net/profile/Peter_Kahn/publication/220667044_Defining_the_axis_of_a_helix/links/5b86ab1e299bf1d5a730ff2e/Defining-the-axis-of-a-helix.pdf`[2].

Note this extremely important observation: "Since V1 and V2 are both perpendicular ot the axis, their corss product will have the direction of the axis". (Here V1 and V2 are angle bisectors.) This should make my work much simpler! To some extent this suggest that this work is not as original as I had thought.

Note further that Equations 7 and 8 of this paper give BETTER equations for radius r and the distance d than what I have so far given.

Note: Here is an example of a question asked on Math Stack Exchange which is essentially answered by this paper:

`https://math.stackexchange.com/questions/878051/why-does-a-3d-line-of-segments-with-constan`
`878079#878079`

The answerer in fact predicts the linear bisection method which I have outlined.

Note: This must be studied immediately:

`https://math.stackexchange.com/questions/1041780/how-to-prove-the-bisector-vector-of-the-an` `1042231#1042231`

This assumes helical to begin with so not of much use.

Note: Must read this: `https://www.researchgate.net/profile/Peter_Kahn/publication/220667044_Defining_the_axis_of_a_helix/links/5b86ab1e299bf1d5a730ff2e/Defining-the-axis-of-a-helix.pdf`[2].

`https://www.win.tue.nl/~wstomv/publications/mathmitering-final.pdf` `https://www.clinbiomech.com/article/S0268-0033(98)00080-1/abstract` `https://gist.github.com/peteristhegreat/3b76d5169d7b9fc1e333` `https://www.sciencedirect.com/science/article/pii/S0022309303008573` `https://www.sciencedirect.com/science/article/pii/S0022309307005583`

This reference is EXTREMELY IMPORTANT `https://link.springer.com/article/10.1023/A:1015863923728`

This may be worth reading: `https://link.springer.com/article/10.1007/PL00011063`

Some discussion of "screw transformations" `http://dergipark.gov.tr/download/article-file/56483`

CRITICAL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=56653`

This is a valuable reference `http://www.12000.org/my_notes/screw_axis/index.htm`

17 References that need to be studied or reviewed

NOTE: This is a discussion of representing joint angles, it is not obvious how valuable it is: `https://www.clinbiomech.com/article/S0268-0033(98)00080-1/abstract`

NOTE: This needs to be read and understood, it is not clear how related it is: `https://ieeexplore.ieee.org/document/56653`

This is a long, expensive book, but it may be quite relevant[10]: `https://books.google.com/books?hl=en&lr=&id=1LZlSZ7ORrQC&oi=fnd&pg=PP1&ots=0hSEwJvlUB&sig=xNG9UWv_H1OXHwaOiOBJN7TW6xA#v=onepage&q&f=false`

Note: There is another long, deep book that needs to be obtained and studied[11]. `https://books.google.com/books?hl=en&lr=&id=FHPlDWvz1bEC&oi=fnd&pg=PP1&ots=TsOnodavEZ&sig=HO86UUVlqRVWGqY-Tv02nb7x7NA#v=onepage&q&f=false`

Note: This work uses the term "segmented" helix, perhaps that is a known term and I need to switch to it:

`https://www.researchgate.net/publication/236066626_Segmented_helical_structures_formed_by_ABC_star_copolymers_in_nanopores`

This is a discussion of segmened coils in a protein structure:

`https://www.sciencedirect.com/science/article/pii/0022283688903701`

17.1 To Be Done

Need to understand what an "alpha coil" protein structure is. (Done: An alpha coil is one of the most common and studied protein forms, roughly providing distance in the protein, consisting of a helix formed by amino acids, which tend to be cross bonded.

Need to answer the Math stack exchange question.

References

[1] Eric A Lord. Helical structures: the geometry of protein helices and nanotubes. *Structural Chemistry*, 13(3-4):305–314, 2002.

[2] Peter C Kahn. Defining the axis of a helix. *Computers & chemistry*, 13(3):185–189, 1989.

[3] Lizhi Gu, Peng Lei, and Qi Hong. Research on discrete mathematical model of special helical surface. In *Green Communications and Networks*, pages 793–800. Springer, 2012.

[4] HSM Coxeter et al. The simplicial helix and the equation tan(n θ) = n tan(θ). *Canad. Math. Bull*, 28(4):385–393, 1985.

[5] Robert Read. Transforming optimal tetrahelices between the boerdijk-coxeter helix and a planar-faced tetrahelix. *Journal of Mechanisms and Robotics*, 2018.

[6] Michael Elgersma and Stan Wagon. The quadrahelix: A nearly perfect loop of tetrahedra. *arXiv preprint arXiv:1610.00280*, 2016.

[7] Hassan Babiker and Stanisław Janeczko. *Combinatorial cycles of tetrahedral chains.* IM PAN, 2012.

[8] EA Lord and S Ranganathan. Sphere packing, helices and the polytope {3, 3, 5}. *The European Physical Journal D-Atomic, Molecular, Optical and Plasma Physics*, 15(3):335–343, 2001.

[9] EA Lord and S Ranganathan. The γ-brass structure and the boerdijk–coxeter helix. *Journal of non-crystalline solids*, 334:121–125, 2004.

[10] Stephen Hyde, Z Blum, T Landh, S Lidin, BW Ninham, S Andersson, and K Larsson. *The language of shape: the role of curvature in condensed matter: physics, chemistry and biology.* Elsevier, 1996.

[11] Jean-François Sadoc and Rémy Mosseri. *Geometrical frustration.* Cambridge University Press, 2006.

A Mathematica Code

```
The following math in Mathematica may be useful.

Note: Mathematica has build in Vector Angle function that can be used for alpha!!

(* c = 2 r Sin[theta/2] *)

Chord[r_,theta_] := 2 r Sin[theta/2]
 (* 1 == c^2 + d^2 *)
```

```
DZ[r_,theta_] := Sqrt[1 - Chord[r,theta]]
(* Note, this may be a problem --- Chord lenght can be diameter (2r), but I am treating it as 
(* Some radiuses are not possible for some thetas with a length of one *)
(* Maximum radius as a function of theta *)
Mxr[theta_] := MaxValue[1 == 2 r Sin[theta/2],{r}]

Mxr[theta_] := 1/ (2  Sin[theta/2])


 Note: I should go ahead and make these functions....

(* P0 = {0, r, 0} *)
(* P1 = {r Sin[theta], r Cos[theta], d} *)
(* P2 = {r Sin[2 theta], r Cos[2 theta], 2 d} *)
(* P3 = { r Sin[3 theta], r Cos[3 theta], 3 d} *)
(* S0 = P0 - P1 *)
(* S1 = P1 - P2 *)
(* S2 = P2 - P3 *)
P[n_,r_,theta_] := { r Sin[n theta], r Cos[n theta], n DZ[r,theta]}
S[n_,r_,theta_] := P[n+1,r,theta] - P[n,r,theta]

(* Define Ps to be P symmetric; the points P0 and P1 are
symmetric in the XY plane. *)
Ps[n_,r_,theta_] :=
With[{nh = (n - (1/2))},
  { r Sin[nh theta], r Cos[nh theta], nh DZ[r,theta]}]


AlphaC[r_,theta_] := VectorAngle[S[0,r,theta],S[1,r,theta]]


f[r_, theta_] :=
 ArcCos[-1 + (r - r Cos[theta]) (-r Cos[theta] + r Cos[2 theta]) +
   4 r^2 Sin[theta/2]^2 -
   r Sin[theta] (-r Sin[theta] + r Sin[2 theta])]

Plot3D[f[r, theta]/ Degree, {r, 0, 2}, {theta, -Pi/6, Pi/6},
 AxesLabel -> Automatic]


Plot3D[AlphaC[r, theta]/ Degree, {r,0,N[Mxr[Pi]]}, {theta, 0, Pi},
 AxesLabel -> Automatic]

(*

Note: Given that we can compute $\alpha$ from $r,\theta$ in this way, one interesting thing to
```

```
is simply what is the minimum or maximum r (or theta) that matches an alpha. The maximum r is t
largest helix that matches, the minimum is the smallest. But with $\psi$ we will know more.
*)

PlaneNormal[a_, b_] := Normalize[Cross[a, b]]
(* Maybe psi is just theta? *)

Psi[pv0_, pv1_, v_] :=
With[{n = PlaneNormal[pv0, pv1]}, ArcSin[(n.v) /(Norm[v] Norm[n])]]

(* This is computing the angle between planes
PsiC[rx_,thetax_] :=
With[{r = rx,theta = thetax}, Evaluate[Psi[S[0,r,theta],S[1,r,theta],S[2,r,theta]]]]


Plot3D[PsiC[r, theta]/ Degree, {r, 0, N[Mxr[Pi/2]]}, {theta, 0, Pi/2},
 AxesLabel -> Automatic]

(*
Note: A strategy for computation is: given alpha, find maximum or minimum radius.
Then slow change radius until \psi if find.  However, Mathematic has optimization
built in, so we may be able to use an energy function.
*)

Enrg[alpha_,psi_,r_,theta_] := (AlphaC[r,theta] - alpha)^2 + (PsiC[r,theta] - psi)^2


(* These seem to work, but they produce rules, instead of values, what up with that? *)

Rx[alpha_,psi_] := r /. FindMinimum[Enrg[alpha,psi,r,t],{r,t}][[2]][[1]]
Tx[alpha_,psi_] := t /.  FindMinimum[Enrg[alpha,psi,r,t],{r,t}][[2]][[2]]

(* This should return 2! *)
(* Is Psi the VectorAngle of the Normals of each ABC joint? *)
Rx[AlphaC[2, Pi/20], PsiC[2, Pi/20]]
FindMinimum[Enrg[AlphaC[2, Pi/20], PsiC[2, Pi/20],r,t],{r,t}]

Unprotect[$PerformanceGoal]
$PerformanceGoal = ‘‘Speed’’

Plot3D[Rx[a,p]/Degree, {a,0,Pi/10},{p,0,Pi/10}]

(* Now that we can in theory find Rx, Tx as a function of alpha and psi,
It is important that we have a check function (or two.)
The best check would be to use a reconstruction based on transformations
of alpha and psi. I need to think about how this should be done with a
```

```
RollPitchYaw Matrix, and in what order. First you by \alpha, then you rotate by \psi.
I think we are doing ‘‘Pitch, Roll, Yaw’’. First we Pitch by \alpha, then
we Roll into the plane of the joint, then we yaw by \psi.
In fact what we are trying to do is to fly a little airplane in a helical course.
‘‘Pitch down by alpha’’, ‘‘Yaw by psi’’, ‘‘roll so that we are perpendicular
to the the plane of the last joint’’.  Maybe we never need to the roll component.
I want intrinsic angle rotation (by the definition.) We do seem to have a mobile
frame of reference.
So in fact I want EulerMatrices, not RollPitchYaw matrices.

Euluer[{\alpha,\beta,\gamma},{a,b,c}], where a,b,c = 1,2,3, let us specify those.
So we need to define how our mobile frame works. We will want $y$ to always point
‘‘out’’ of the helix, ‘‘z’’ to point along the motion’’, and $x$ to be
tangential to the helix.

If we start with the intrinsic frame aligned with the extrinsic frame, repeated
transofrmations will produce a helix, but not about the $z$ axis.
We have so far described our mostion as ‘‘rotate about y’’ by $\alpha$, then
rotation about $x by $\psi$. Then our $z$ axis would be pointed in the
write direction, we would translate  by $z$, in the intrinsic frame of refrence,
and repeat.

Note, mathematica seems to have a way to render a helix, this would be very useful to me:
*)

ListPointPlot3D[
Table[Table[{t, Cos[t + s Pi/2], Sin[t + s Pi/2]}, {t, 0,
      5 Pi, .2}], {s, 4}], BoxRatios -> Automatic]

Graphics3D[
GeometricTransformation[{Hue[#/Pi], Sphere[{5, 0, 0}, 1]},
  EulerMatrix[{#, Pi/2, #}]] & /@ Range[0, 2 Pi, Pi/16]]

(* My claim is that we can somehow combine an EulerTransformation with a translation
to end up with a helix.

We can call GeometricTransoforamtion on an Euler matrix, can we pass a vector
as an object? Can we create an object of length L which is a vector
and then create a transformation?
*)

Graphics3D[Arrow[{{1, 1, 1}, {1, -1, 2}}], Axes -> True,
  AxesLabel -> {"X", "Y", "Z"}, ImageSize -> Large]

Graphics3D[
GeometricTransformation[{Hue[#/Pi],Arrow[{{1, 1, 1}, {1, -1, 2}}] },
Composition @@ {
     TranslationTransform[{1, 1, 1}],
      EulerMatrix[{0, Pi/2, #}]
}] & /@ Range[0, 2 Pi, Pi/16]]


 EulerMatrix[{Pi/15, Pi/20, 0}] [1,1,1]


 myt = TranslationTransform[{0, 0, 1}] EulerMatrix[{Pi/15, Pi/20, 0}]


 Graphics3D[{
  Opacity[1]
  , Red
  , Arrow[{{0, 0, 0}, {1, 0, 0}}]
  , Green
  , Arrow[{{0, 0, 0}, {0, 1, 0}}]
  , Blue
  , Arrow[{{0, 0, 0}, {0, 0, 1}}]
  , Opacity[0.2]
  , GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
   Composition @@ {
     RotationTransform[Pi/4, {0, 0, 1}]
     , TranslationTransform[{1, 1, 1}]
     }
   ]
 }]

 Graphics3D[{
  Opacity[1]
  , Red
  , Arrow[{{0, 0, 0}, {1, 0, 0}}]
  , Green
  , Arrow[{{0, 0, 0}, {0, 1, 0}}]
  , Blue
  , Arrow[{{0, 0, 0}, {0, 0, 1}}]
  , Opacity[0.2]
  , GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
    Composition @@ {
       RotationTransform[Pi/15,{0,1,2}],
     , TranslationTransform[{1, 1, 1}]
     }
   ]  & /@ Range[0, 5, 1]
 }]
 Graphics3D[
   GeometricTransformation[{Hue[#/Pi],Arrow[{{1, 1, 1}, {1, -1, 2}}] },
       Composition @@ {
       RotationTransform[Pi/15,{0,1,2}],
     , TranslationTransform[{1, 1, 1}]
     }
 ]]


 Composition @@ {
     RotationTransform[Pi/4, {0, 0, 1}]
     , TranslationTransform[{1, 1, 1}]
 }

 Graphics3D[

  (GeometricTransformation[Cuboid[-{1, 1, 1}/4, {1, 1, 1}/4],
   Composition @@ {RotationTransform[Pi/4, {0, 0, #}],
     TranslationTransform[{1, 1, #}]}]) & Range[3]

 ]

double[x_] := 2 x

Nest[double,#,4] & Range[3]

(* Now, what we really want to do here, is  construct
a transform out of \alpha, \pi, ending up with the arrows head to tail
*)

Clear[ComposeN]
ComposeN[0] = ScalingTransform[{1,1,1}]
ComposeN[1] = Composition @@ {
  RotationTransform[ Pi/6, {0, 1, 0}],
  Composition @@ {
    TranslationTransform[{1, 1, 1}],
    RotationTransform[ Pi/6, {0, 0, 1}]
  }
}

ComposeN[n_] :=  Composition[ComposeN[Floor[n/2]],ComposeN[Ceiling[n/2]]]


 myarrows =
 Table[
   GeometricTransformation[{
     Arrow[{{0, 0, 0}, {0, 0, 2}}],
     Arrow[{{0, 0, 0}, {0,1/2, 0}}]},
     ComposeN[i]
   ],{i,1,10}
   ]

 Graphics3D[myarrows]


 (* I think to do what I want, I need to do my own transformationns. *)
 (* An object is a vector and an upvector.
 The 0th object is a z-aligned vector starting at the origin, with a y upvector.
 The n+1th object is:
 A) Take the nth object (AB), find the head B of the vector.
 B) Create a vector of length N pointing in the same direction as the nth object.
 C) Translate it along the nth object to the head.
 D) Relative to
 E) Rotate it in the AB coordinate frame in the Y direction by Alpha.
 F) Rotate it in the X direction by Psi.
 G) Apply these same arguments to the up Vector.

Note that Mathematica has powerful RotationMatrix functions built in to which
 we can use the Nth vectors as input, to rotate in a plane. So in this sense
 we may actually be able to accomplish this.
*)

(* I'm going to try to use F to mean the nth object. *)
(* We will make the first to points computed from the Helix *)
F[0,alpha_,psi_,init_] := init

(* We are forced to pass the initial object in as a starting point *)
F[n_,alpha_,psi_,init_] :=
  With[{Prev = F[n-1,alpha,psi,init]},
      With[{
          A = Part[Prev,1], B = Part[Prev,2], U = Part[Prev,3]},
        With[{BA = B-A, UA = U-A},
        (* now we want to contruct the parts of P_n *)
        Module[{NA,NB,NU,RA,RP,NV},
          NA = B;
          V = BA;
          (* This is in theory spanned by BA, UA *)
          RA = RotationMatrix[alpha,{BA,UA}];
          RP = RotationMatrix[psi,BA];
          NB = RP.RA.V;
          NU = RP.RA.UA;
          NB = NB + B;
          NU = NU + B;
          {NA,NB,NU}
    ]
  ]]]]


  (* F requires an initial object (of three points, A, B Up.) These
  are essentially P[0],P[1], and P[0] + [0,1,0]. *)

  FTest[n_,alpha_,psi_,r_,theta_] :=
  With[{P0 = P[0,r,theta]},
    Part[F[n,alpha,psi,{P0,P[1,r,theta],P0+{0,1,0}}
      ],1]
    - P[n,r,theta]]

  FTest[1,Pi/8,0,1,0]

  FStratTest[k_,alpha_,psi_] :=
  With[
    {r = Rx[alpha,psi], t = Tx[alpha,psi], P0 = {0,0,0} },
    With[{P0 = P[0,r,t], P1 = P[1,r,t]},
      With[{init = {P0,P1,P0+{0,1,0}}},
    Print["alpha, psi, r, t ",alpha,", ",psi,", ",r,", ",t 180 / Pi];
    For[i = 0, i < k, i++,
      Print["point ",i];
      Print["P ",P[i,r,t]];
      Print["F ",Part[F[i,r,t,init],1]]
      ]
    ]
  ]]

  (* Pretty sure this is showing the r is way to small *)
  FStratTest[4, 10 Degree,10 Degree]


  (* now attempting to compute the transformation matrix
  that transforms {A,B,U},{L,N_A,N_B, Tau} into the next A,B,U.
  M = W . R . T , where W is the twist, R is the rotation,
  and T is the translation. We can test this by using a synthetic shelix to generate points an
  *)

  (* I first need to build and test a simple Skew test.
  I want a function that yields the closest points on two skew lines,
  defined by vectors. *)
```

```
  Skew[p1_,d1_,p2_,d2_] := (* p1(2) is a point on line with direction d1(2),
  result is a vector containg C1 and C2, nearest points on those lines.
  from: https://en.wikipedia.org/wiki/Skew_lines *)
  With[{d12 = Cross[d1,d2],
      d21 = Cross[d2,d1]},
(*    Print[N[d1]]
    Print[N[d2]]
    Print[N[d12]]
    Print[N[d21]]
    Print[123123123]
    Print[N[Cross[d2,d12]]]
    Print[444444444]
    Print[N[Cross[d1,d21]]]
    Print[666666666]*)
  With[{n2 = Cross[d2,d12],
      n1 = Cross[d1,d21]},
(*    Print[555555555555]
    Print[N[n1]]
    Print[N[n2]] *)
    With[{C1 = p1 + (((p2-p1) . n2)/ (d1 . n2)) d1,
        C2 = p2 + (((p1-p2) . n1)/ (d2 . n1)) d2},
(*    Print[N[C1]]
    Print[N[C2]]
    Print[999999999999999] *)
    (* For unknown perforance reasons, I have to force evaluation to a number here *)
      {N[C1],N[C2]}]]]

  TestSkew[] :=
  With[{xp1 = {2,0,0}, xd1 = {1,0,0},
      yp1 = {1,5,1}, yd1 = {0,2,0}},
    Skew[xp1,xd1,yp1,yd1]]

  (* Now we could try a 3-point test. We take the
  first 3 points of our shelix and then compute the mid-points,
  and try to find r by treating these as skew lines with
  closes intersections. *)

  M[n_,r_,theta_] := (* midpoint of Pn-1, Pn+1 *)
  (P[n-1,r,theta]+P[n+1,r,theta])/2

  AngelBi[n_,r_,theta_] := (* unit angle bisector *)
 Normalize[M[n,r,theta] - P[n,r,theta]]

 (* This does not return the radius, it returns d, the travel *)
 (* If we made this a function of 4 points instead, then
 this would be independent of the current defintion of our
```

```
test shelix.*)

(* This seems to work. So if we can compute 4 points
by virtue of the transform, we can probably compute d (z travel).
However, the two points should let us go further. Since
they lie on the axis plane, we can compute theta by projecting
onto the plane normal to this line. Or just translate
the points along this angle and then call VectorAngle.


*)
ComputeAxisPointsFrom4[ps_] := (* Input is a vector of four points *)
With[{P0 = Part[ps,1],
    P1 = Part[ps,2],
    P2 = Part[ps,3],
    P3 = Part[ps,4]},
With[{M1 = (P0+P2)/2,
    M2 = (P1+P3)/2},
  With[{B1 = Normalize[M1-P1],
      B2 = Normalize[M2-P2]},
    With[{Sk = Skew[P1,B1,P2,B2]},
      With[{C1 = Part[Sk,1],
            C2 = Part[Sk,2]},
        Print[N[VectorAngle[P1 - C1, P2 -C2]]]
        Sk
    ]]]]]

 (* I think this is supposed to return D *)
TestAxisPoint[k_,r_,theta_] :=
   With[{P0 = P[k,r,theta],
       P1 = P[k+1,r,theta],
       P2 = P[k+2,r,theta],
       P3 = P[k+3,r,theta]},
     With[{CS = ComputeAxisPointsFrom4[{P0,P1,P2,P3}]},
      Norm[Part[CS,1] - Part[CS,2]]]]

   (* IMPORTANT: This seems to be my most successful approach, the only one that is clear. *)
   (* This function returns DZ[r,theta], which seems important. *)
   (* This is the distance along the shelix axis of two points representing angle
   bisetctors. *)
  Test3PointTheory[k_,r_,theta_] :=
  With[{D0 = N[AngelBi[k,r,theta]],
      D1 = N[AngelBi[k+1,r,theta]]},
    Print[D0]
    Print[D1]
    Print[357]
    With[{CS = Skew[P[k,r,theta],D0,P[k+1,r,theta],D1]},
      Print[CS]
      Print[3333333333333]
      Norm[Part[CS,1] - Part[CS,2]]
  ]]

  (* Now beginning simple work on face diagonals *)

  Zf[w_,r_] := 1/Sqrt[1 + Tan[r]^2 + Tan[w]^2]
  Xf[w_,r_] :=
  With[{z = Zf[w,r]},
    z Tan[w]]
  Yf[w_,r_] :=
  With[{z = Zf[w,r]},
    z Tan[r]]

  TestXYZ[w_,r_] :=
  With[{ z = Zf[w,r], y = Yf[w,r], x = Xf[w,r]},
    Print N[Norm[{x,y,z}]]
    ]


  DfromOmega[L_,w_] :=
  With[{ t = Tan[w]},
    (L t) /(2Sqrt[(t^2)/4 + 1])]

  (* One way to test is to generate a synthetic shelix,
  compute rho, omega from it, and then check that this
  formular for the D from omega against this synthetic verison.
  This requires the computation of rho,omega, from r,theta *)

  (* Returns pair rho, omega (rotation about X, rotation about Y) *)
  (* This is currently wrong becasue it relies on the P0_x = P1_x,
  but I am not calculating it that way !!! *)
  (* I could compute these points and compute a rotation about Y I suppose.)
RhoOmegaFromRTheta[r_,theta_] :=
 With[{P0 = Ps[0,r,theta],
       P1 = Ps[0+1,r,theta],
       P2 = Ps[0+2,r,theta]},
   With[{a = ArcTan[Part[P1,1]/Part[P1,3]]},
     With[{rt = RotationTransform[-a,{0,1,0}]},
(*       Print[444]
       Print[N[a] 180 / Pi]
       Print[N[P1]]
       Print[N[rt[P1]]]
       Print[5555555555]
       Print[N[rt[P2]]]
       *)
     With[{v = rt[P2] - rt[P1]},
(*        Print[N[v]]  *)
       With[{x = Part[v,1],
           y = Part[v,2],
           z = Part[v,3]},
         {ArcTan[z,y],ArcTan[z,x]}
         ]]]]]


 Len[r_,theta_] := Norm[Ps[0,r,theta]-Ps[1,r,theta]]

 (* This is a check, we need to be able to compute rho and omega
 and recover the 3rd point from it. Okay, this now works. *)
 HCheck[r_,theta_] :=
 With[{ ro = RhoOmegaFromRTheta[r,theta],
   L = Len[r,theta]},
   With[{ rho = Part[ro,1], omega = Part[ro,2]},
     With[{ q = QFromRhoOmega[L,rho,omega]},
     {q Tan[omega], q Tan[rho], q}]]]


 DfromRTheta[r_,theta_] :=
 With[{ro = RhoOmegaFromRTheta[r,theta],
      L = Len[r,theta]},
   DfromOmega[L,Part[ro,2]]
   ]


 DFromLQW[L_,q_,w_] :=
 L Sin[ArcTan[Tan[w]/((L/q) - 2)]]

 ChiFromLQW[L_,q_,w_] :=
 With[{A = q Tan[w] /2,
     B = (L/2) - q},
   Print[N[A]]
   Print[N[B]]
 ArcTan[A/B]]

 TestDFromLQW[r_,theta_] :=
  With[{ ro = RhoOmegaFromRTheta[r,theta],
   L = Len[r,theta]},
   With[{ rho = Part[ro,1], omega = Part[ro,2]},
     With[{ q = QFromRhoOmega[L,rho,omega]},
       Print[N[L]]
       Print[N[q]]
       DFromLQW[L,q,omega]]]]

  (* I believe I can compute rho, omega from r,theta. I am now attempting to
  compute d from rho, omega, L, based on the projection diagrem. To debug this,
  I need to compute the points F, G, H, and check the projection diagram.
  Since I have already computed the points and a the rotation about Y which
  brings them to Z axis, this would seem to be straightforward. *)



 FGH[r_,theta_] :=
 With[{
     ro = RhoOmegaFromRTheta[r,theta],
     L = Len[r,theta],
     P0 = Ps[0,r,theta],
     P1 = Ps[0+1,r,theta],
     P2 = Ps[0+2,r,theta]},
   With[{ a = ArcTan[Part[P1,1]/Part[P1,3]],
         rho = Part[ro,1],
         omega = Part[ro,2]},
       With[{
           rt = RotationTransform[-a,{0,1,0}],
           q = QFromRhoOmega[L,rho,omega]},
       With[{ x = q Tan[omega],
           y = q Tan[rho],
           z = q },
         With[{ F = {0,0,-(L/2)},
             G = {0,0,(L/2)},
             H = {x,y,(L/2)+z},
           c = {0, Part[rt[P0],2], 0}},
           { {F,G,H},
             {rt[P0]-c,rt[P1]-c,rt[P2]-c}
             }
 ]]]]]

 (* To complete the current approach, I need to take several steps: *)
 (* Modify the code to compute $d$ from 4 arbitrary points.
 Test that this matches the current results by using a helical test pattern. *)
 (* Generate 4 points based on Rho,Omega, by using symmetry and the approach above. *)


 (* This routine generates 4 symmetric points on the Z axis matching the input values. *)
 (* To test this is sensible, I really should be able to invert it---that is,
 To find an r,theta parametrization that matches the L,rho,omega parametrization.
 Of course, in a ense, that is what we are trying to do entirely---so why not back out with my
```

```
computations? *)
Generate4Points[L_,rho_,omega_] :=
With[{
    B = {0,0,-L/2},
    C = {0,0,L/2},
    q = QFromRhoOmega[L,rho,omega]
  },
  With[{ x = q Tan[omega],
      y = q Tan[rho]},
    With[{
        A = {-x,y,-(L/2 + q)},
        D = {x,y,(L/2 + q)}},
        {A,B,C,D}
]]]


Generate4PointsRTheta[r_,theta_] :=
{Ps[-1,r,theta],Ps[0,r,theta],Ps[1,r,theta],Ps[2,r,theta]}

(* Compute d from 4 points *)
Bisector[X_,O_,Y_] :=
Normalize[O - (X+Y)/2]

(* In typical cases Skew is hanging below.
I don't even know a mechanism by which that should be possible. *)
ComputeDR[A_,B_,C_,D_] :=
 With[{Bb = Bisector[A,B,C],
     Cb = Bisector[B,C,D]},
   With[{CS = Skew[B,Bb,C,Cb]},
     {Norm[Part[CS,1] - Part[CS,2]],
      Norm[Part[CS,1] - B]}
 ]]

 ComputeD[A_,B_,C_,D_] :=
 Part[ComputeDR[A,B,C,D],1]


 TestComputeD[r_,theta_] :=
 With[{
     ro = RhoOmegaFromRTheta[r,theta],
     L = Len[r,theta]
   },
   With[{
       rho = Part[ro,1],
       omega = Part[ro,2]
       },
    With[{
        abcd = Generate4Points[L,rho,omega]
        },
    With[{
        A = Part[abcd,1],
        B = Part[abcd,2],
        C = Part[abcd,3],
        D = Part[abcd,4]
      },
(*      Print[N[abcd]] *)
      ComputeD[A,B,C,D] - N[DZ[r,theta]]]]]]

   TestComputeD0[r_,theta_,abcd_] :=
  With[{
      ro = RhoOmegaFromRTheta[r,theta],
      L = Len[r,theta]
    },
    With[{
        rho = Part[ro,1],
        omega = Part[ro,2]
        },
    With[{
        A = Part[abcd,1],
        B = Part[abcd,2],
        C = Part[abcd,3],
        D = Part[abcd,4]
      },
      ComputeD[A,B,C,D]]]]

  TestComputeDagainstHelix[] :=
  With[{abcd = Generate4PointsRTheta[2, Pi/10]},
    With[{
        A = Part[abcd,1],
        B = Part[abcd,2],
        C = Part[abcd,3],
        D = Part[abcd,4]
      },
      ComputeD[A,B,C,D] - DZ[2,Pi/10]]]

  TestComputeDagainstROmega[rho_,omega_] :=
  With[{abcd = Generate4Points[1, rho, omega]},
    With[{
        A = Part[abcd,1],
        B = Part[abcd,2],
        C = Part[abcd,3],
        D = Part[abcd,4]
      },
    ComputeD[A,B,C,D]]]



   (* Full test computation: Given rho,omega, compute r, theta,
   and see that it matches.
   1) Starting with r, theta, generate rho, omega, L.
   2) from rho, omega, L, generate 4 points
   3) Compute D from this.
   5) Use basic formulae to compute r,theta.
   *)

   (* Given the cord length, how do I compute R? *)
   ChordFromLD[L_,D_] := Sqrt[L^2 - D^2]
   ThetaFromRC[R_,C_] :=
   2 ArcSin[C/(2R)]


   (* Return basic helix params: r, theta, d, c *)
 GetHelixParams[L_,rho_,omega_] :=
      With[{abcd = Generate4Points[L, rho, omega]},
         With[{
             A = Part[abcd,1],
             B = Part[abcd,2],
             C = Part[abcd,3],
             D = Part[abcd,4]
           },
           With[{dr = ComputeDR[A,B,C,D]},
             With[{di = Part[dr,1],
                 r = Part[dr,2]},
               With[{
                   c = ChordFromLD[L,di]},
                 With[{ theta2 = ThetaFromRC[r,c]},
                   {r,theta2,di,c}
                   ]]]]]]

   TestComputeDagainstROmega[rad_,theta_] :=
   With[{ L = Len[rad,theta],
       ro = RhoOmegaFromRTheta[rad,theta]},
     On[Assert];
     With[{
         rho = Part[ro,1],
         omega = Part[ro,2]},
       With[{ hp = GetHelixParams[L,rho,omega]},
         With[{ r = Part[hp,1],
          theta2 = Part[hp,2]},
        Print[theta2 180 / Pi]
        Assert[Abs[theta2 - theta] < 0.00001];
        Assert[Abs[r - rad] < 0.00001];
        Off[Assert];
        ]]]]


(* One way to try to solve this is to fold into
one function, then use symmetries and zeroes there
to simplify.)

(* This is an intermediate point with the bisector operations
unfolded *)
UnifiedComp0[L_,rho_,omega_] :=
With[{
    B = {0,0,-L/2},
    C = {0,0,L/2},
    q = QFromRhoOmega[L,rho,omega]
  },
  With[{ x = q Tan[omega],
      y = q Tan[rho]},
    With[{
        A = {-x,y,-(L/2 + q)},
        D = {x,y,(L/2 + q)},
        Bb = {x/2,-y/2,(q-L)/2},
        Cb = {-x/2,-y/2,(L-q)/2}},
      With[{CS = Skew[B,Bb,C,Cb]},
          With[{
          dr = {Norm[Part[CS,1] - Part[CS,2]],
          Norm[Part[CS,1] - B]}},
            With[{di = Part[dr,1],
                r = Part[dr,2]},
              With[{
                  c = ChordFromLD[L,di]},
                With[{ theta2 = ThetaFromRC[r,c]},
                  {r,theta2,di,c}
    ]]]]]]]]


(* Now I attempt to unfold the Skew *)
UnifiedComp1[L_,rho_,omega_] :=
With[{
    B = {0,0,-L/2},
    C = {0,0,L/2},
    q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          Bb = {x/2,-y/2,(q-L)/2},
          Cb = {-x/2,-y/2,(L-q)/2}},
        With[{BbXCb = N[Cross[Bb,Cb]],
            CbXBb = N[Cross[Cb,Bb]]},
          With[{n2 = N[Cross[Cb,BbXCb]],
              n1 = N[Cross[Bb,CbXBb]],
              Bbz = Part[Bb,3],
              Cbz = Part[Cb,3]},
            (* Note: C-B exist only in z dimesion! *)
         With[{CmBz = Part[C-B,3],
             BmCz = Part[B-C,3],
             n2z = Part[n2,3],
             n1z = Part[n1,3]
           },
(*           Print[555555555]
           Print[(C-B) . n2]
           Print[(CmBz)  n2z]
           Print[B-C]*)
(*          With[{CmBz = L,
                BmCz = -L
             },
             *)
            (* Note: this dot product can be replaced with a
            z multiple, since the other terms are zero *)
(*              With[{C1 = B + (((C-B) . n2)/ (Bb . n2)) Bb,
                C2 = C + (((B-C) . n1)/ (Cb . n1)) Cb},*)
              (* But that allows us to cancel out... *)
              With[{C1 = B + ((CmBz  n2z)/ (Bb . n2)) Bb,
                  C2 = C + ((BmCz  n1z)/ (Cb . n1)) Cb},
(*              With[{C1 = B + (L / Bbz ) Bb,
                    C2 = C + (-L / Cbz) Cb}, *)
(* But that can be furher simplied... *)
(*               With[{C1 = B + (2 L / (q-L) ) Bb,
                  C2 = C + (2 L / (q-L)) Cb}, *)
                    (* And then decomposed.... *)
 (*                   With[{S = 2 L / (q - L)},
                      With[{C1 = B + S Bb,
                          C2 = C + S Cb},
                        *)
                        With[{ CS =
                              {N[C1],N[C2]}},
                            With[{
                                dr = {Norm[Part[CS,1] - Part[CS,2]],
                                  Norm[Part[CS,1] - B]}},
                              With[{di = Part[dr,1],
                                  r = Part[dr,2]},
                                With[{
                                    c = ChordFromLD[L,di]},
                                  With[{ theta2 = ThetaFromRC[r,c]},
                                    {r,theta2,di,c}
          ]]]]]]]]]]]]

N[UnifiedComp1[1, Pi/10, Pi/30]]
```

```
N[UnifiedComp2[1, Pi/10, Pi/30]]


 UnifiedComp2[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          Bb = {x/2,-y/2,(q-L)/2},
          Cb = {-x/2,-y/2,-(q-L)/2}},
        With[{BbXCb = Cross[Bb,Cb],
            CbXBb = Cross[Cb,Bb]},
          With[{n2 =   Cross[Cb,BbXCb],
              n1 =  Cross[Bb,CbXBb]
            },
            With[{
             n2z = Part[n2,3],
             n1z = Part[n1,3]
              },
      With[{C1 = B + ((L  n2z)/ (Bb . n2)) Bb,
        C2 = C + ((-L  n1z)/ (Cb . n1)) Cb},
            With[{
              dr = {Norm[C1 - C2],
               Norm[C1 - B]}},
             With[{di = Part[dr,1],
               r = Part[dr,2]},
              With[{
                c = ChordFromLD[L,di]},
               With[{ theta2 = ThetaFromRC[r,c]},
                {r,theta2,di,c}
    ]]]]]]]]]]]


UnifiedComp3[L_,rho_,omega_] :=
With[{
  B = {0,0,-L/2},
  C = {0,0,L/2},
  q = QFromRhoOmega[L,rho,omega]
 },
 With[{ x = q Tan[omega],
   y = q Tan[rho]},
  With[{
    A = {-x,y,-(L/2 + q)},
    D = {x,y,(L/2 + q)},
    (* Apparently we can take out the fact of 1/2 (1/8) here *)
    Bb = {x,-y,(q-L)},
    Cb = {-x,-y,-(q-L)}},
   With[{BbXCb = Cross[Bb,Cb],
     CbXBb = Cross[Cb,Bb]},
    With[{n2 =   Cross[Cb,BbXCb],
      n1 =   Cross[Bb,CbXBb]
     },
     With[{
     n2z = Part[n2,3],
     n1z = Part[n1,3]
      },
      With[{Bn2 = n2z / (Bb . n2),
         Cn1 = n1z / (Cb . n1)},
      With[{C1 = B + (L  Bn2) Bb,
        C2 = C + (-L  Cn1) Cb},
            With[{
              dr = {Norm[C1 - C2],
               Norm[C1 - B]}},
             With[{di = Part[dr,1],
               r = Part[dr,2]},
                              With[{
                                  c = ChordFromLD[L,di]},
                                With[{ theta2 = ThetaFromRC[r,c]},
                                  {r,theta2,di,c}
  ]]]]]]]]]]]


N[UnifiedComp3[1, Pi/10, Pi/30]]

 UnifiedComp4[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      C = {0,0,L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho]},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          (* Apparently we can take out the fact of 1/2 (1/8) here *)
          Bb = {x,-y,(q-L)},
          Cb = {-x,-y,-(q-L)}},
        With[{BbXCb = Cross[Bb,Cb],
            CbXBb = Cross[Cb,Bb]},
          With[{n2 =   Cross[Cb,BbXCb]
(*              n1 =   Cross[Bb,CbXBb] *)
            },
            With[{
             n2z = Part[n2,3]
(*            n1z = Part[n1,3] *)
              },
              With[{Bn2 = n2z / (Bb . n2)
                (*  Cn1 = n1z / (Cb . n1) *)
                },
              With[{C1 = B + (L  Bn2) Bb
(*                  C2 = C + (-L  Cn1) Cb *)
                },
                (* A Key point -- C1.y = C2.y, and C2.x = - C2.x, and C2.z = - C2.z *)
                (* This should simplify the norming operation below! *)
                (* I this case the norm depends only on x and z:
                With[{A = {a, b, c},
       B = {-a, b, -c}},
                  Norm[A - B]] => 2Sqrt[a^2 + c^2] *)
                (* So, in the first place, don't compute C2! *)
                          With[{
                              di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                              r =   Norm[(L  Bn2) Bb]},
                              With[{
                                  c = ChordFromLD[L,di]},
                                With[{ theta2 = ThetaFromRC[r,c]},
                                  {r,theta2,di,c}
  ]]]]]]]]]]
```

```
N[UnifiedComp4[1, Pi/10, Pi/30]]
```

```
 QFromRhoOmega[L_,rho_,omega_] :=
 L/Sqrt[1+Tan[rho]^2+Tan[omega]^2]

  ChordFromLD[L_,D_] := Sqrt[L^2 - D^2]
  ThetaFromRC[R_,C_] :=
  2 ArcSin[C/(2R)]

 UnifiedComp5[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho],
      u = q - L},
      With[{
          A = {-x,y,-(L/2 + q)},
          D = {x,y,(L/2 + q)},
          (* We have removed a factor of 1/2 (1/8) here *)
          Bb = {x,-y,u},
          Cb = {-x,-y,-u}},
        With[{n2 =   Cross[Cb,Cross[Bb,Cb]]
          },
            With[{
             n2z = Part[n2,3]
              },
              With[{Bn2 = n2z / (Bb . n2)
                },
                With[{LBn2 = L Bn2},
              With[{C1 = B + LBn2 Bb
                },
                With[{
                    di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                    r =   Abs[LBn2] Norm[ Bb]},
                  With[{
                      c = ChordFromLD[L,di]},
                    With[{ theta2 = ThetaFromRC[r,c]},
                      {r,theta2,di,c}
  ]]]]]]]]]]



  (* Note, possibly this would be better done with C,D, or A,B *)
  (* Also, this neither takes nor returns tau, so it must be making an assumption *)
  (* How do we compute tau from this? tau should be computed from the points C1, D
  As teh change in the upvector. *)
UnifiedCompTwoPoints[L_,B_,D_] :=
With[{q = D[[3]] - L/2,
      x = D[[1]],
      y = D[[2]]},
    With[{u = q - L},
      With[{Bb = {x,-y,u},
          Cb = {-x,-y,-u}},
        With[{n2 =   Cross[Cb,Cross[Bb,Cb]]
          },
            With[{
             n2z = Part[n2,3]
              },
              With[{Bn2 = n2z / (Bb . n2)
                },
                With[{LBn2 = L Bn2},
              With[{C1 = B + LBn2 Bb
                },
                With[{
                    di = 2Sqrt[Part[C1,1]^2 + Part[C1,3]^2],
                    r =   Abs[LBn2] Norm[ Bb]},
                  With[{
                      c = ChordFromLD[L,di]},
                    With[{ theta2 = ThetaFromRC[r,c]},
                      {r,theta2,di,c}
  ]]]]]]]]]]


(* This does not seem to take tau into effect. Unified Comp needs to return tau *)
UnifiedComp7[L_,rho_,omega_] :=
  With[{
      B = {0,0,-L/2},
      q = QFromRhoOmega[L,rho,omega]
    },
    With[{ x = q Tan[omega],
        y = q Tan[rho],
      u = q - L},
      With[{
          D = {x,y,(L/2 + q)}
        },
        UnifiedCompTwoPoints[L,B,D]]]]


  N[UnifiedComp5[1, Pi/10, Pi/30]]
  N[UnifiedComp7[1, Pi/10, Pi/30]]


Plot3D[Part[UnifiedComp5[1,rho,omega],3],{rho,0,Pi/4},{omega,0,Pi/4},AxesLabel->{rho, omega}]

Plot3D[Part[UnifiedComp5[1,rho,omega],1],{rho,0,Pi/2},{omega,0,Pi/30},AxesLabel->{rho, omega}]

Plot[Part[UnifiedComp5[1,rho,0],1],{rho,0,Pi/2},AxesLabel->{rho}]

Plot[Part[UnifiedComp5[1,Pi/30,omega],1],{omega,0,Pi/2},AxesLabel->{omega}]



(* In this section, I am purely attempting to work out the
coordinates of the two face-to-face tetrahedra in order to
compute rho, omega in this platonic case. *)

(* This is all wrong because the centroid to centroid distance is actually 1/3 *)
(* These are values for a regular tetrahedron, I think. *)
(* These numbers are wrong, I think... *)
alphaA = 0;
alphaB = 0;
betaA = ArcTan[Sqrt[2]/2];
betaB = ArcTan[Sqrt[2]/2];


tau = 2 Pi / 3;

(* I now believe these are wrong, and the approach is probably
wrong. Instead we need to use Rodrigues' rule, rotate by tau,
and get the resulting unit vector back and apply simple trig
to it to get omega and rho. Note this also raises the
question that it might be better to consider the face angles
as unit vectors in the first place, but you still have to
```

```
rotate them. However, possibly unfolding the definition of rho and
omega as unit vectors will let us come closer to a formula.
We will, however, write this as rotating a vector of length L,
becasue that way a valuable point will be produced for our
compuation later. *)

(* I may not need this with built in Mathematica stuff. *)
vRot[k_,v_,theta_] := v Cos[theta] + Cross[k,v] Sin[theta]
+ k (k . v)(1 - Cos[theta])

vRotM[k_,theta_] :=
With[{ x = Part[k,1],
    y = Part[k,2],
    z = Part[k,3]},
  With[{C = {
        {0, -z, y},
        {z, 0, -x},
        {-y,x, 0}
    }},
    Print[C // MatrixForm];
    Print[(1 - Cos[theta])C^2 // MatrixForm];
    Print[IdentityMatrix[3] + (1 - Cos[theta])C^2 // MatrixForm];
    Print[IdentityMatrix[3] + (Sin[theta] C)+ (1 - Cos[theta])C^2 // MatrixForm];
    IdentityMatrix[3] + (Sin[theta] C) + (1 - Cos[theta]) C^2]]

vRotAux[k_,v_,theta_] :=
vRotM[k,theta].v

(* NOTE: This is incorrect, the vector v must be rotated into
k. I is the opposite of k (if symmetric) but must be rotated. *)
testvRot[tau_] :=
With[{alphaA = 0,
    alphaB = 0,
    betaA = -ArcTan[Sqrt[2]/2],
    betaB = ArcTan[Sqrt[2]/2],
    L = 1
  },
  With[{k = Normalize[{Sin[alphaB],Sin[betaB],Cos[betaB]}]},
    With[{
      v = Normalize[-L {Sin[alphaA],Sin[betaA],Cos[betaA]} + k]
    },
    Print[k];
    Print[Norm[k]];
    Print[v];
    Print[Norm[v]];
    With[{ rt = RotationTransform[tau,k]},
      With[{
          (*  vr = vRot[k,v,tau] *)
          vr = rt[v]
        },
      Print[N[vr]];
      Print[Norm[vr]];
      Print[N[Normalize[vr]]];
      With[{ x = Part[vr,1],
          y = Part[vr,2],
          z = Part[vr,3]},
        With[{ ro =
            (* Warning: Mathematic uses x,y order here, Javascript other *)
            { ArcTan[z,y],ArcTan[z,x]}},
          Print[N[ro 180 / Pi]];
          ro]]]]]]]
```

```
N[ArcTan[Sqrt[2]/2]]
```

```
(* Starting new work here. This is an attempt to compute
the point A (and thereby D) from NB and NC (normal vectors)
and tau (and L). This is based on my new understand that
we can compute a vector to be rotated.

I hope to test this with both regular regular objects,
but in the end with the regular tetrahedron, for which
we have a known solution (via Javascript work.)

Variables:

NB = Vector normal of the face containing B
NC = Vector normal of the face containg C
L = distance between B and C
tau = the twist

PA = the projection of BA onto the vector NB
R = The up-pointing rejection of BA onto NB. (perpendicular)
delta = angle between NC and the axis
gamma = angle between the XZ plane and the NB vector.

V0 = Vector to be rotated

AfromParams[v0,taux,NB] -- a function to compute A
```

```
V0fromLNB[L,NB,delta]

TestAWith225s[] = test AfromParams with faces cut and 22.5 degrees

*)

AngleWithXZ[v_] :=
VectorAngle[{v[[1]],0,v[[3]]},v]

(* I now believe this should be done with a rotation.
V0 is a rotation of NB vector by delta in the pure
up direction (that is, in the ZY plane about {1,0,0} *)
V0fromLNB[L_,NB_,NC_,delta_] :=
With[{k = Normalize[NB],
    rt = RotationTransform[-delta,{1,0,0}]},
  With[{ v0 = L Normalize[rt[NB]]},
    v0]
  ]

ADirFromParam[L_,v0_,tau_,NB_] :=
With[{k = Normalize[NB]},
  With[{ rt = RotationTransform[tau,k]},
    L  Normalize[rt[v0]]]]

ComputeBalancingRotation[NB_,NC_] :=
(* our stategy will be to project on the XY plane,
and then compute the midpoint vector *)
With[{ NBN = Normalize[NB],
    NCN = Normalize[NC]},
  With[{Bp = {NBN[[1]],NBN[[2]]},
      Cp = {NCN[[1]],NCN[[2]]}},
    Print[N[{Bp,Cp}]];
    With[{ ba = ArcTan[Bp[[1]],Bp[[2]]],
           ca = ArcTan[Cp[[1]],Cp[[2]]]},
    Print[N[ba] 180 / Pi];
    Print[N[ca] 180 / Pi];
    Print[N[(ba+ca)/2] 180 / Pi];
    With[{m = Bp+Cp},
      With[{mn = m / 2},
        Print[N[ArcTan[mn[[1]],mn[[2]]]] 180 / Pi];
        (* We want to return the angle that makes the negative of this vector point up *)
        Print[N[mn]];
        With[{theta = -(ba+ca)/2 + -Pi/2 },
          Print[70707070];
          Print[N[theta] 180 / Pi];
```

```
         With[{
             (*              rt = RotationTransform[{{mn[[1]],mn[[2]],0},{0,-1,0}}] *)
             rt = RotationTransform[-theta,{0,0,-1}]
           },
           With[{ Br = rt[NB],
               Cr = rt[NC]},
             Print[N[{theta,Br,Cr}]];
             {theta,Br,Cr}
      ]]]]]]]]

TestComputeBalancingRotation[] :=
With[{C = {1,1/2,1},
    B = {-1/2,-1,-1}},
  With[{res = ComputeBalancingRotation[B,C]},
    Print[res];
    With[{ theta = res[[1]],
        Br = res[[2]],
        Cr = res[[3]]},
      With[{Bp = Normalize[{Br[[1]],Br[[2]]}],
          Cp = Normalize[{Cr[[1]],Cr[[2]]}]},
        Print[8888888];
        Print[N[Bp]];
        Print[N[Cp]];
          Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[Bp[[2]] < 0];
        Assert[Cp[[2]] < 0];
        res
  ]]]]

TestComputeBalancingRotation2[] :=
With[{C = {1,3/4,1},
    B = {1,0,-1}},
  With[{res = ComputeBalancingRotation[B,C]},
    Print[N[res]];
    With[{ theta = res[[1]],
        Br = res[[2]],
        Cr = res[[3]]},
      With[{Bp = Normalize[{Br[[1]],Br[[2]]}],
          Cp = Normalize[{Cr[[1]],Cr[[2]]}]},
         Print[8888888];
        Print[N[Bp]];
        Print[N[Cp]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[N[Bp[[1]]] == -N[Cp[[1]]]];
        Assert[Bp[[2]] < 0];
        Assert[Cp[[2]] < 0];
        res
  ]]]]

(* Warning: I may not have the sign of this correct. *)
AfromLtauNBNC[L_,tau_,NBunb_,NCunb_] :=
With[{ res = ComputeBalancingRotation[NBunb,NCunb]},
  With[{NB = res[[2]],
    NC = res[[3]]},
    With[{delta = VectorAngle[{0,0,1},NC]},
      With[{v0 = V0fromLNB[L,NB,NC,delta]},
        Assert[N[Norm[v0]] == L];
        With[{ Ad = ADirFromParam[L,v0,tau,NB],
            B = {0,0,-L/2}},
          Assert[Abs[Norm[Ad] -L ] < 0.00001];
          Ad + B]]]]]

(* For testing, I create a block of size 1
with NB and NC based on half of a 45 degree angle. *)
(* I need to write a function that takes two face normals,
and computes the balancing of them, returning two new normals
that are balanced around the up vector. Then I need
to test that the same point is computed from both directions
even when we have unbalanced normals. *)

TestAfromLtauNBNC[tau_] :=
With[{
    L0 = 1,
    NB0 = {0,-Sin[Pi/8],-Cos[Pi/8]},
    NC0 = {0,-Sin[Pi/8],Cos[Pi/8]},
    tetAngle = ArcTan[Sqrt[2]/2]},
  With[{
      NB1 = {0,-Sin[tetAngle],-Cos[tetAngle]},
      NC1 = {0,-Sin[tetAngle],Cos[tetAngle]},
      B = {0,0,-L/2}
    },
    Print[N[NB1]];
    Print[N[NC1]];
    Print[8888];
    With[{
        AA = AfromLtauNBNC[L0,tau,NB1,NC1]
      },
      Print[333333333333];
      Print[N[AA]];
      Print[Norm[AA - B]];
      AA
  ]]]

(* I need to write a test that sends the face normals
into UnifiedComp. Probably it would be better
to separate out a version of UnifiedComp that takes
points (the rho/omega version can implement that. *)




On[Assert];

TestRegularTets[] :=
With[{
    L0 = 1/3,
    tetAngle = ArcTan[Sqrt[2]/2]},
  With[{
    NB1 = {0,-Sin[tetAngle],-Cos[tetAngle]},
    NC1 = {0,-Sin[tetAngle],Cos[tetAngle]},
    tau = 2 Pi /3
    },
    With[{
        A = AfromLtauNBNC[L0,tau,NB1,NC1]},
      Print[N[A]];
      With[{ B = {0,0,-L0/2},
          D = {-A[[1]],A[[2]],-A[[3]]}
        },
        Print[4444];
        Print[L0];
        Print[N[B]];
        Print[N[D]];
        With[{ res = UnifiedCompTwoPoints[L0,B,D],
            theta = ArcCos[-2/3]},
          Print[N[res]];
          Print[33333];
          Print[N[res[[2]]] 180 / Pi];
          Assert[Abs[res[[2]] - theta] < 0.00001];
]]]]]

TestRegularTets2[] :=
With[{
    L0 = 1,
    tetAngle = ArcTan[Sqrt[2]/2],
    (* Rotation by an aribitraty amount should give us the same number. *)
    testRot = Pi/17
  },
  With[{ rt = RotationTransform[testRot,{0,0,1}]},
    With[{
        NB1 = rt[{0,-Sin[tetAngle],-Cos[tetAngle]}],
        NC1 = rt[{0,-Sin[tetAngle],Cos[tetAngle]}],
        tau = 2 Pi /3
      },
      With[{
          A = AfromLtauNBNC[L0,tau,NB1,NC1]},
        Print[N[A]];
        With[{ B = {0,0,-L0/2},
            D = {-A[[1]],A[[2]],-A[[3]]}
          },
          Print[4444];
          Print[L0];
          Print[N[B]];
          Print[N[D]];
          With[{ res = UnifiedCompTwoPoints[L0,B,D],
              theta = ArcCos[-2/3]},
            Print[N[res]];
            Print[33333];
            Print[N[res[[2]]] 180 / Pi];
            Assert[Abs[res[[2]] - theta] < 0.00001];
]]]]]
```