Johann C. Rocholl (Rostock) Style

Delta Robot Kinematics

by Steve Graves

PREFACE

First I would like to thank Johann Rocholl for his contributions. While I have not met him, much of what is written here is thanks to him. For example, all of the Rostock images were generated using modified versions of his OpenSCAD model for the Rostock. His Marlin code was also a source of my verifying the concepts. And then of course I would like to thank him for creating the best DIY 3D printer in existence. When I wrote this I didn't own a delta printer. I am now working with the Kossel Clear. I am using the principles discussed here to make improvements to the bed leveling algorithms. Since my first version I have rewritten the forward kinematics to use trilateration as recommended by Michael Paauwe from New Zealand. Thanks! Michael. I have also added references to the Marlin variables to make this a more useful document when tuning your printer. And I added some discussion of how the shape of the printing area is the intersection of the arcs of the arms around the virtual columns. I have found this concept useful for graphically playing around with arms of varying lengths and columns at positions other than the 120 degree spacing.

INTRODUCTION

This is my analysis of the style of delta robot used in the Rostock printer. I have yet to see a published analysis of this style. There are a number analyzing the original delta robot, the Clavel. I realize that the inverse kinematics for Rostock is already available in firmware and is probably described some where on the Internet if I look long enough, but I like to understand things myself. And it is my philosophy that if the problem appears to be something I can figure out myself, then I want to do that before reading about it. It gives me more insight and my brain is not biased by the accepted solution. I may come up with different way of looking at the problem than the accepted solution.

Once you break it down, this is a reasonably simple geometry problem. In fact, I now find myself saying to people that it is simply the Pythagorean theory applied to each column. And the math can be simplified down to that very point. So it is important to describe the geometry and define the formulas that are derived from that geometry. Part of describing the geometry is to also establish naming conventions. The delta robot has three columns (The Rostock uses a pair of rods, gray in the fig. 1).. We will call the three columns A, B, and C. Each column has a carriage (yellow in fig. 1) that runs up and down the column. Each carriage has two parallel arms (blue in the fig. 1) that connect to the effector platform (green in the fig. 1). Each pair of arms are the same length and the connections on each carriage are exactly parallel to the the bottom surface of the robot. The bottom surface is called the bed (red in the fig. 1). To make the arms parallel, the connection points on each carriage and the effector platform are the same distance apart.

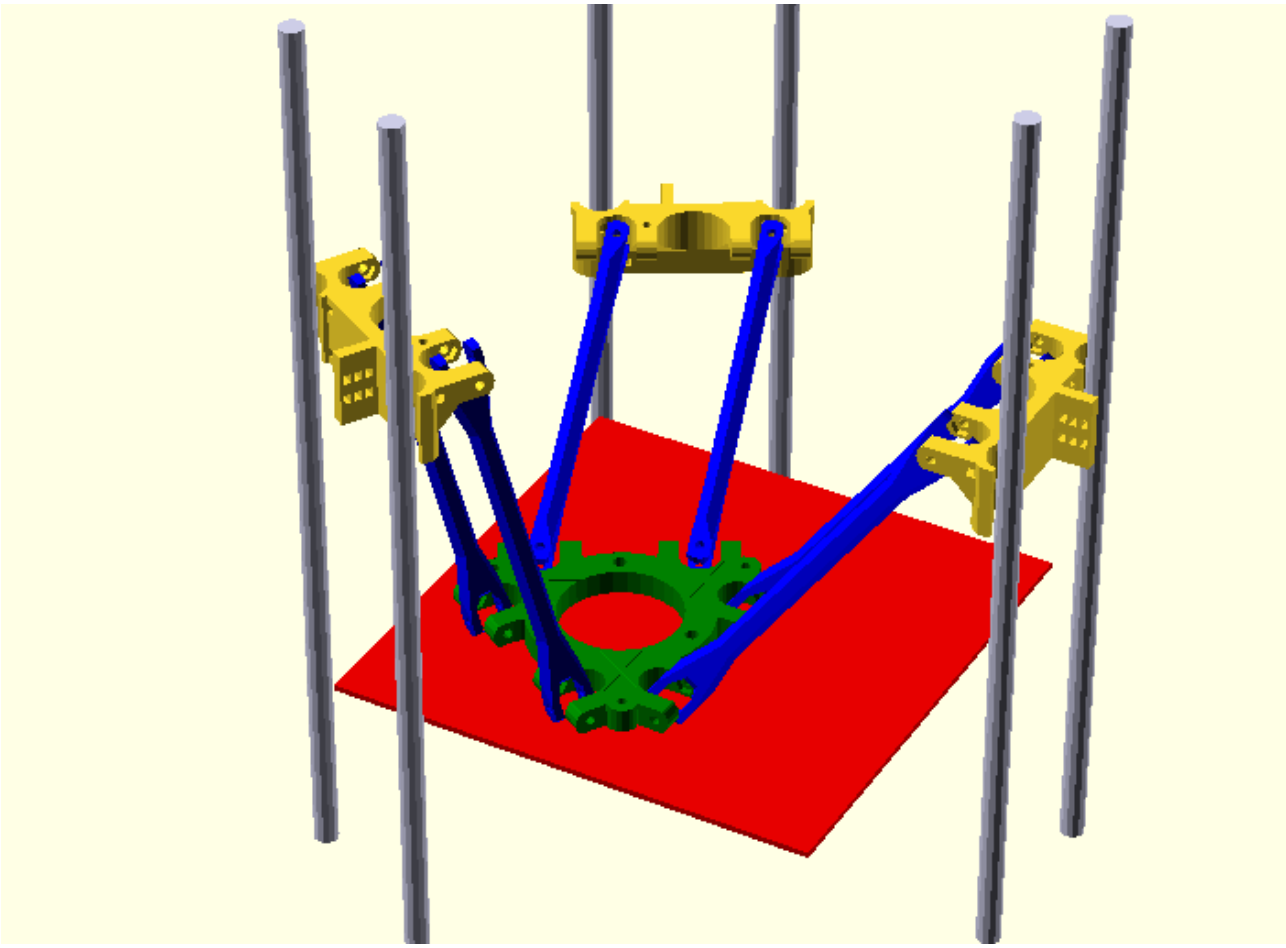


Fig. 1

The first thing I see is that part of the geometry follows the principle of the pantograph. The parallel arms are not only parallel to each other but they also force the edge of the effector platform to be parallel to the connection points on the carriage. We can see this in Fig. 2 with a view directly above the robot. Since there are three carriages and all three have connection points parallel to the bed, the plane of the effector platform must always be parallel to the plane of the bed. To repeat, **the parallel arms from the carriages to the effector platform force its plane to be parallel to the plane of the bed**. I have seen posts questioning the parallel arms, here you have one reason why they exist. To my thinking, making the effector platform be parallel to the bed is the main benefit of the parallel arms.

And since the edge of the effector platform is parallel to the connection points on each carriage, the distance between corresponding points on the effector platform and the carriage is the same. So for the math we can choose the line that connects the point half way along the edge of the effector platform and half way between the connection points of the corresponding carriage. For most calculations we can use this line. We will call this our line of action. The end points of this line are exactly between the pivot points on the effector platform and exactly between the pivot points on the carriage. When we refer to the carriage or the effector platform these are the points of reference. For example, the height of the carriage above the bed is the distance from the bed to this point. And the height of the carriage above the effector platform is the vertical distance from the point on the effector platform and the point on the carriage. To summarize, the math can be based on a single line of action connecting the carriage and the effector platform with the knowledge that the parallel arms maintain a geometric orientation around this line of action.

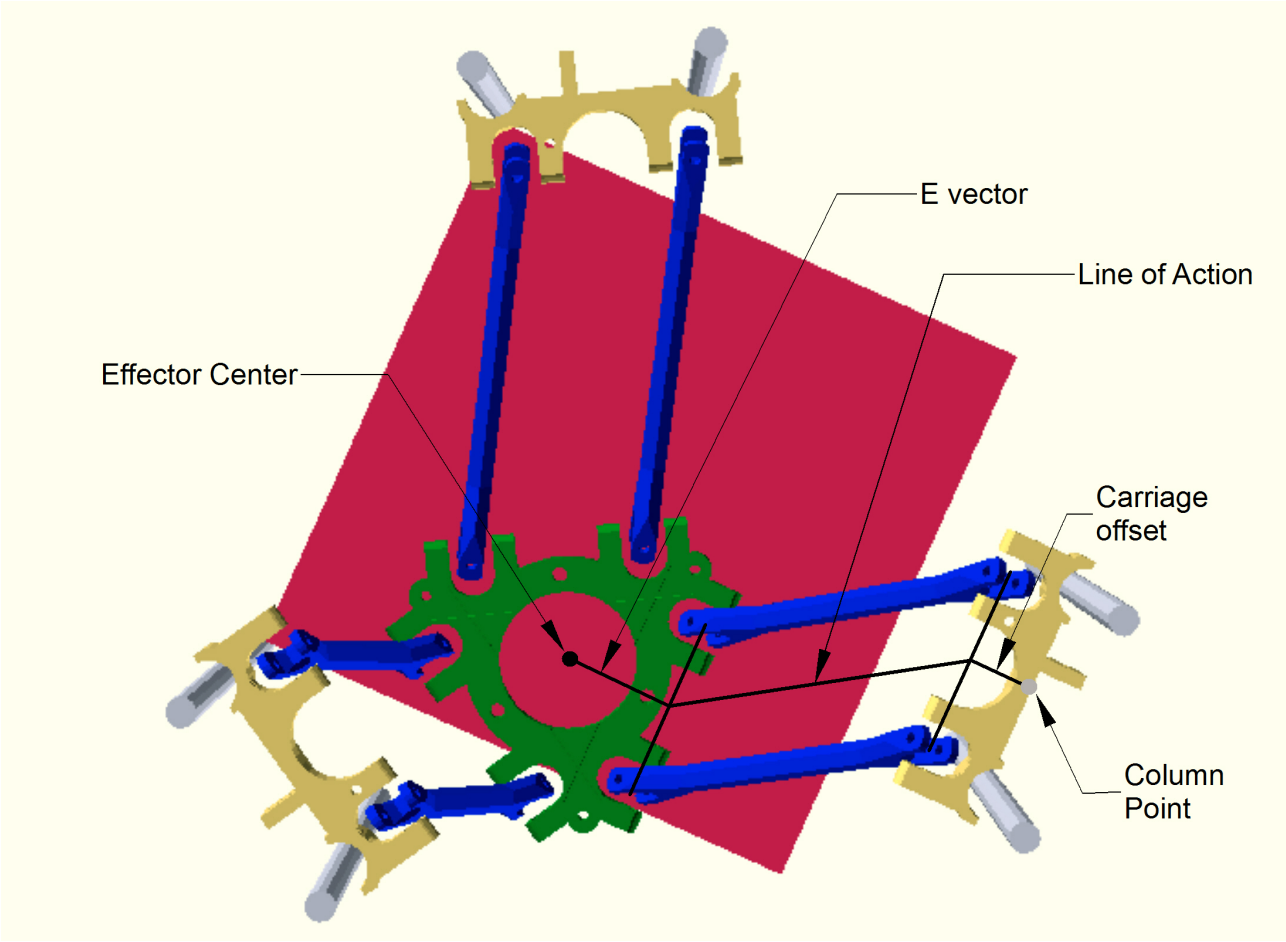


Fig. 2

There are some simplifying assumptions to make. First we will assume that the columns form an equilateral triangle. This is not necessary, one can have a non-equilateral triangle, but this assumption simplifies the math by allowing us to use one variable for the distance between columns. That is, the distance between any two columns is the same as any other two columns. We will call that distance S (side of triangle). It also allows us to choose a coordinate system where the columns are all at the same distance from the origin. Note that the columns can be represented by points. In the case of the Rostock each column point is at the midpoint between the two rods. The distance from the center to each column point is `DELTA_SMOOTH_ROD_OFFSET` in Marlin

Next we will assume that all three pairs of arms are the same length. We will call that length `L` (`DELTA_DIAGONAL_ROD` in Marlin). I don't believe it is necessary that the distance between columns be the same or that the arms are all the same length. The math can be worked out with different lengths between columns and/or with one or more arms of different lengths. It changes the shape of the area that can be covered. This might be beneficial for a robot that prints specialized objects (a house?). The next assumption is that the head is mounted exactly in the center of the effector platform. We will call the distance from where the line of action meets the edge of the effector platform to the center of the effector platform `E` (`DELTA_EFFECTOR_OFFSET`). The coordinates `X`, `Y` and `Z` are the coordinates of the extrusion point on the bottom of the hot head. The center of the effector platform is exactly above the extrusion point and it has the same `X`, `Y` coordinates as the head.

As discussed, we will measure the height of each carriage where the line of action terminates in between the connection points. We will call the height of each carriage above the effector platform

Acz, Bcz and Ccz. We will call the height of the carriages above the bed Az, Bz and Cz. We will call the distance the head extends below the effector Hez. The height of the head above the bed is Z. So Az = Z + Acz + Hez, Bz = Z + Bcz + Hez and Cz = Z + Ccz + Hez. So this leads us to another simplification of the problem. We can derive the formulas for a given X,Y plane and remove Z from the equations. Then we can translate that plane to any Z location. All we need to concern ourselves with is Acz, Bcz and Ccz. Since they are all relative to the effector platform, we can calculate X,Y coordinates in the plane of the effector platform.

```
Az = Z + Acz + Hez
Bz = Z + Bcz + Hez
Cz = Z + Ccz + Hez

Solving for Z
Z = Az - Acz - Hez
Z = Bz - Bcz - Hez
Z = Cz - Ccz – Hez
```

This brings us to a discussion of the Z axis. One can see by the geometry that if one moves all the carriages up by an equal amount then one will change only the Z component of the head. So the Z component is directly related to the carriage heights (Az, Bz and Cz). This is also shown in the above formula. Now implied by the observation about the Z component and the formula above is that Acz, Bcz and Ccz are only dependent on X, Y. This leads to the following discussion for the inverse kinematics.

INVERSE KINEMATICS

The inverse kinematics are the basis for the formulas used in software for controlling the printer. They take the desired X,Y,Z coordinates and calculate the carriage positions (Az, Bz, Cz) that will cause the robot to move to those coordinates.

We will review this below, but here are how the Marlin constants relate to the variables given here. The variable L is represented in Marlin by DELTA_DIAGONAL_ROD. Ae, Be and Ce is DELTA_EFFECTOR_OFFSET. Aco, Bco and Cco is DELTA_CARRIAGE_OFFSET. DR, Ad, Bd and Cd are DELTA_RADIUS. The variable R is DELTA_SMOOTH_ROD_OFFSET.

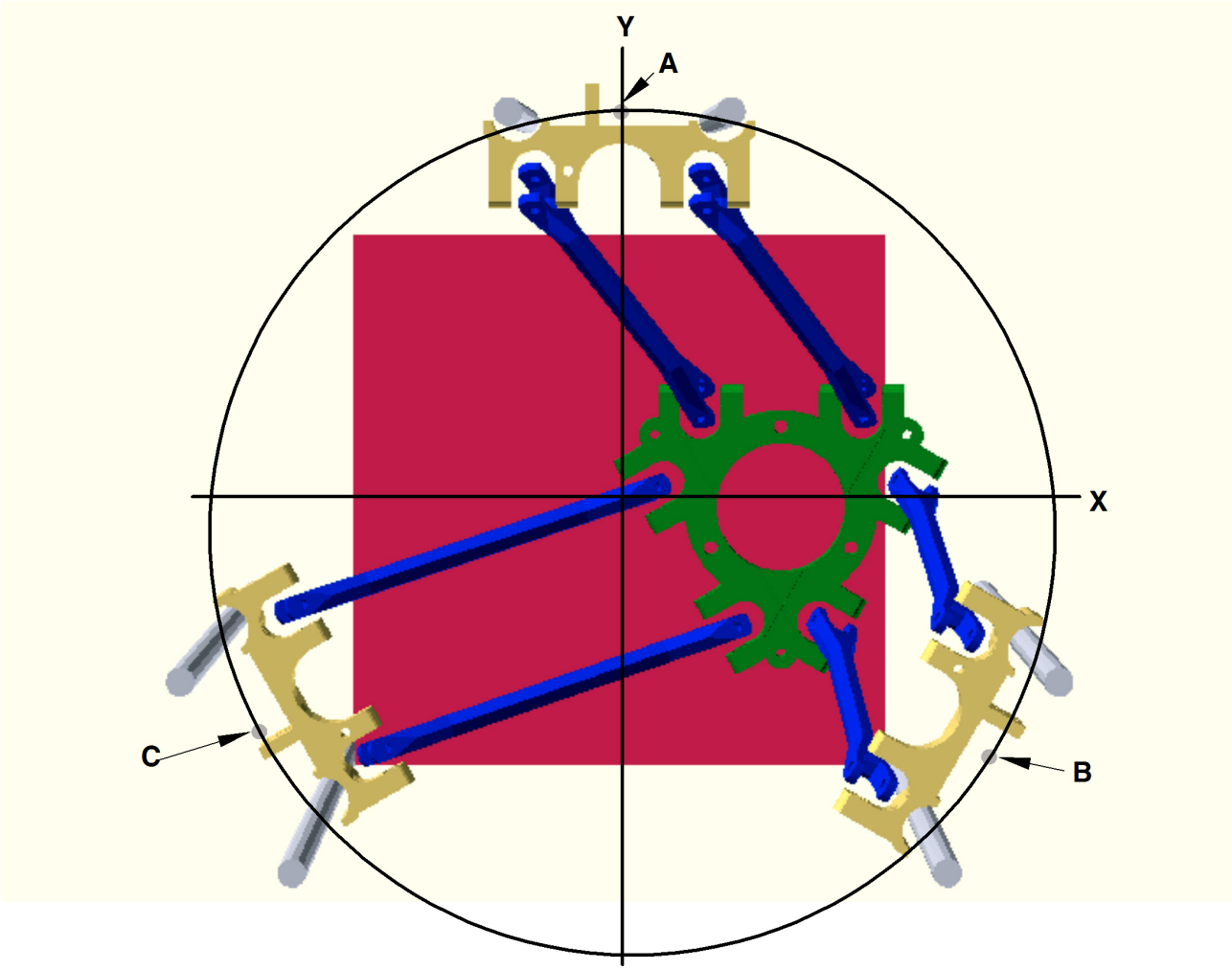


Fig. 3

Now we need to define a coordinate system. We are going to choose a Cartesian system with the origin at the center of the bed. We are going to place column `A` on the positive `Y` axis at a distance `R (DELTA_SMOOTH_ROD_OFFSET)` from the origin. The other columns will lie on a circle at that distance at 120 degrees. Column `B` will be 120 degrees clockwise when viewed from above. Column `C` is another 120 clockwise from `B` or 120 counterclockwise from `A`.

One important set of formulas can be derived by observing that the arms and the vertical rise form a right angle triangle where the third leg is in the plane of the effector platform. Now we need to understand how the the third leg relates to our `X, Y` point in the middle of the effector platform. My first thought was that our line of action passes through the center of the effector platform. But this is not true. The line from the center of the effector platform to edge of the effector platform where the line of action meets the effector platform is perpendicular to the edge. Since the edge is parallel to connection points on the carriage the line on the effector platform will maintain the same orientation in the `X,Y` plane. It is a fixed vector. We will call these vectors `Ae, Be` and `Ce`. By our definition, they are all the same length and the direction is determined by the orientation of the corresponding column. Since vector addition is commutative, we will show that these vectors can be moved from the end of the arc and applied to the column positions. This changes the pivot point to a virtual column position that the points of interest form an arc around, simplifying the math when doing kinematic calculations. So the distance from the edges to the center of the effector platform become a constant (`DELTA_EFFECTOR_OFFSET`) applied once when initializing the software.

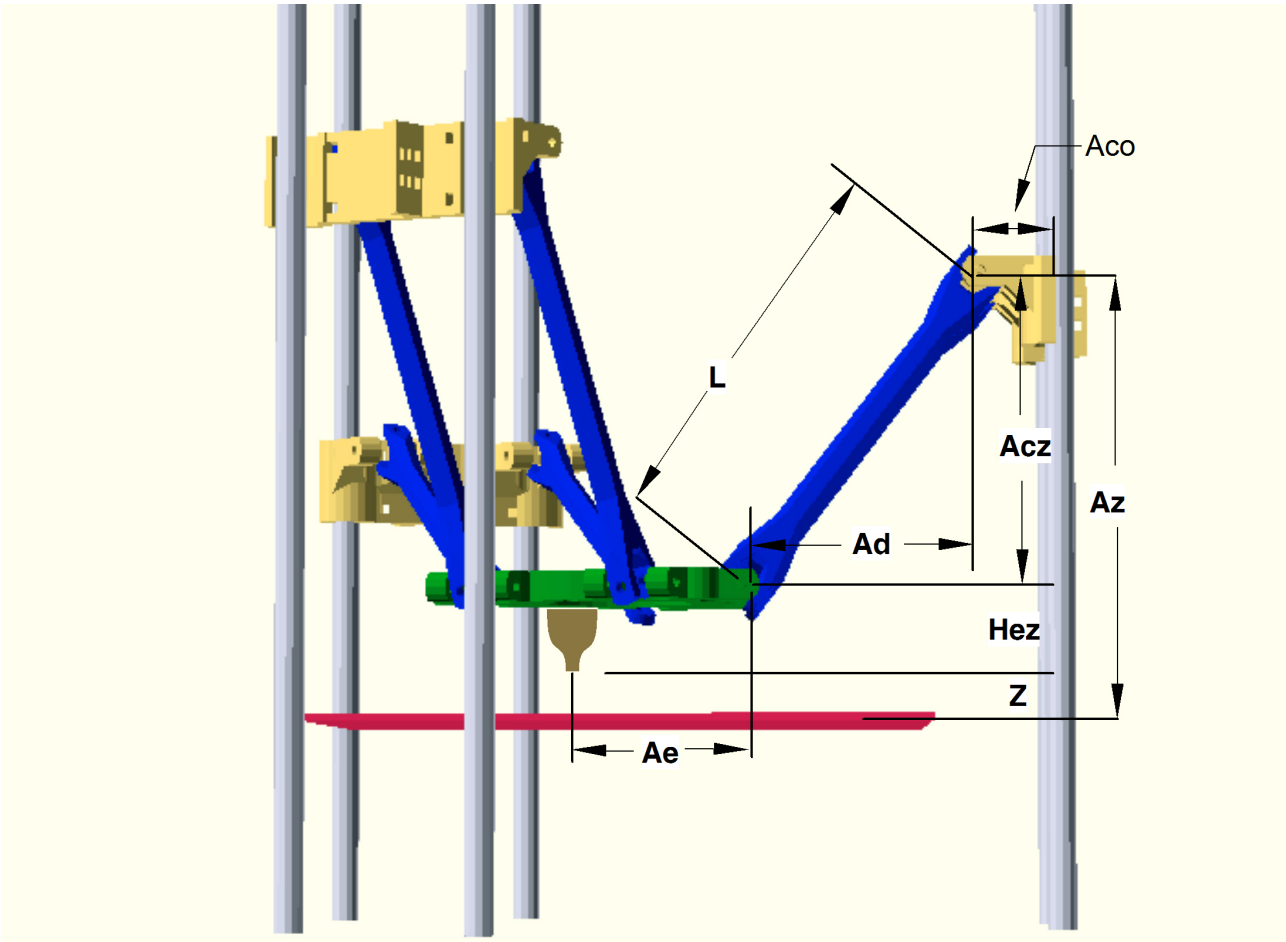


Fig. 4

Now to determine what formulas we can derive from the right angle triangles formed by each arm, the vertical rise (virtual column) and leg in the X-Y plane. We will call the distances of the triangle legs (effector platform edge to the point below the carriage pivot point) in the `XY` plane `Ad, Bd`, and `Cd`. Using the Pythagorean theorem we get the formulas

```
Ad^2 + Acz^2 = L^2
Bd^2 + Bcz^2 = L^2
Cd^2 + Ccz^2 = L^2
```

Now let's relate our columns to the `X, Y` plane. First important thing is to describe the locations of the base of the columns `A, B` and `C` in the `X, Y` plane. We will call the physical locations `Ax, Ay; Bx, By` and `Cx, Cy`. By observation we can see that the actual pivot points are the points exactly below where each line of action terminates on its carriage. We can calculate these locations by subtracting a vector from each column which represents the carriage offset. We will call these pivot locations `Apx, Apy; Bpx, Bpy` and `Cpx, Cpy`. We will use these values for now, but later we can reduce the number of unknowns by moving these locations to virtual column positions. For this analysis we hold `Z` constant, so we are restricting our effector platform to a given `XY` plane. The line of action for each column pivots around the point on the carriage. When the carriage is at a given height above the effector platform, the possible locations in the `XY` plane where the edge of a effector platform meets a given line of action is an arc around the pivot point. If the vector `Ae, Be` and `Ce` were zero and the lines of action met at the center of the effector platform our `X, Y` coordinates would be the intersection between these arcs. But since `Ae, Be` and `Ce` are not zero, the problem is a little more complicated.

Let's make some equations to relate Ae, Be and Ce to our X, Y coordinates. Let's call the coordinates where the lines of action meet the effector platform Acx, Acy ; Bcx, Bcy and Ccx, Ccy. We know that a vector can be represented by a delta X and delta Y, so we will break the vectors down to Aex, Aey; Bex, Bey and Cex, Cey. We will define the vectors as pointing from the edge to the center of the effector platform. So we have the following

```
X = Acx - Aex = Bcx - Bex = Ccx - Cex
Y = Acy - Aey = Bcy - Bey = Ccy - Cey
```

Based on our discussion above the line of action travels on an arc. The formula for a circle is (X - CX)^2 + (Y - CY)^2 = CR^2. Where CX, CY is the center of the circle and CR is the radius. We should be able to solve the equations for one column and determine the other two by similarity. So looking at column A.

```
(Acx - Apx)^2 + (Acy - Apy)^2 = Ad^2
```

Solving for Acx and Acy above and substituting we get

```
Acx = X + Aex
Acy = Y + Aey

(X + Aex - Apx)^2 + (Y + Aey - Apy)^2 = Ad^2
```

We see a simplification, we can treat the pivot points of the circle as Apx - Aex, Apy - Aey. We will call this point Avx, Avy. We will define the following:

```
Avx = Apx - Aex
Avy = Apy - Aey

Bvx = Bpx - Bex
Bvy = Bpy - Bey

Cvx = Cpx - Cex
Cvy = Cpy - Cey
```

These are the locations of our "virtual" columns. These locations can be calculated in software at initialization and then used for all subsequent calculations. These are at radius DELTA_RADIUS in the Marlin firmware written by Johann. The virtual columns are the DELTA_TOWER variables in Marlin. We show the virtual columns in **fig. 5 & 6**. We also show single arms on the lines of action. So **fig. 6** is a physical representation of our mathematical model. The arms meet at the X,Y coordinates and travel on virtual columns at locations at a distance of the carriage offset plus the effector offset away from the physical column locations. This is the model we will use in firmware.

This is good time to once again relate the other Marlin constants to the variables given here. The variable L is represented in Marlin by DELTA_DIAGONAL_ROD. Ae, Be and Ce is DELTA_EFFECTOR_OFFSET. Aco, Bco and Cco is DELTA_CARRIAGE_OFFSET. DR, Ad, Bd and Cd are DELTA_RADIUS. The variable R is DELTA_SMOOTH_ROD_OFFSET.

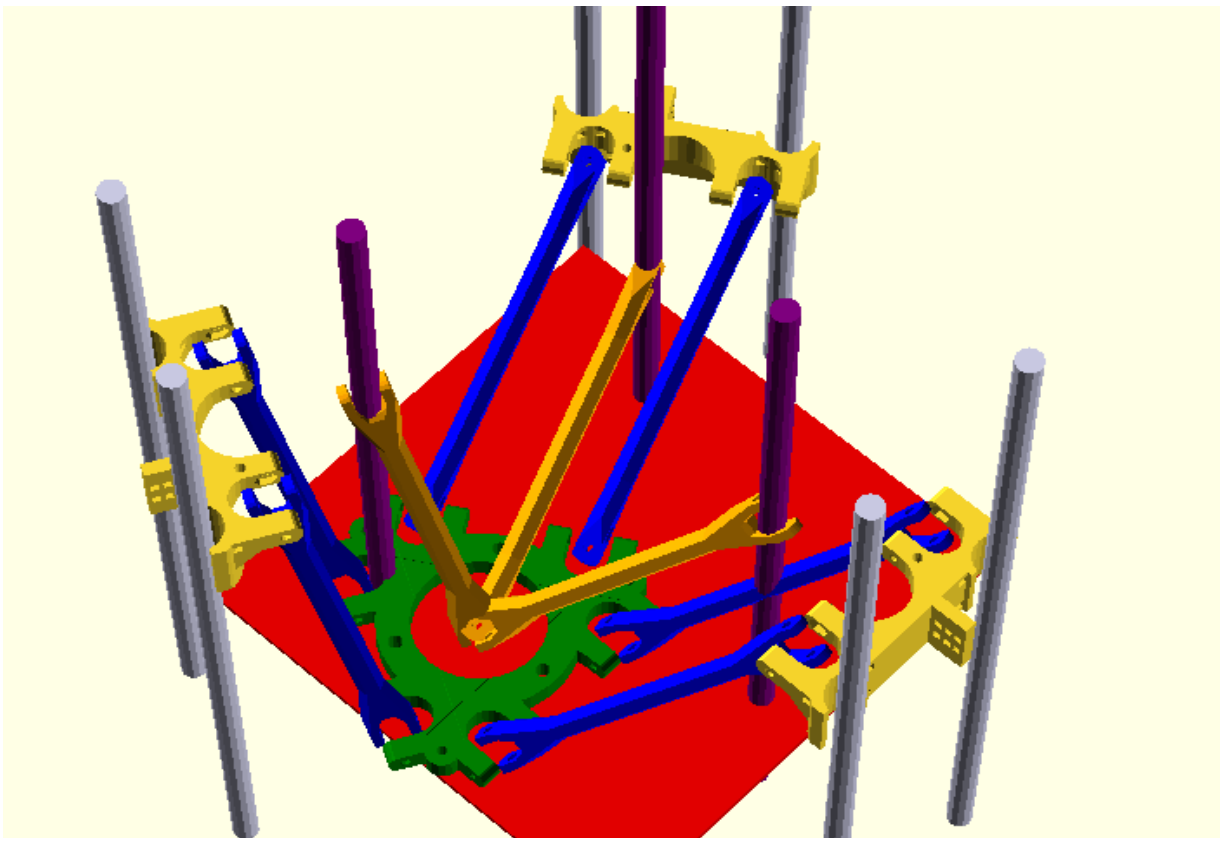


Fig. 5

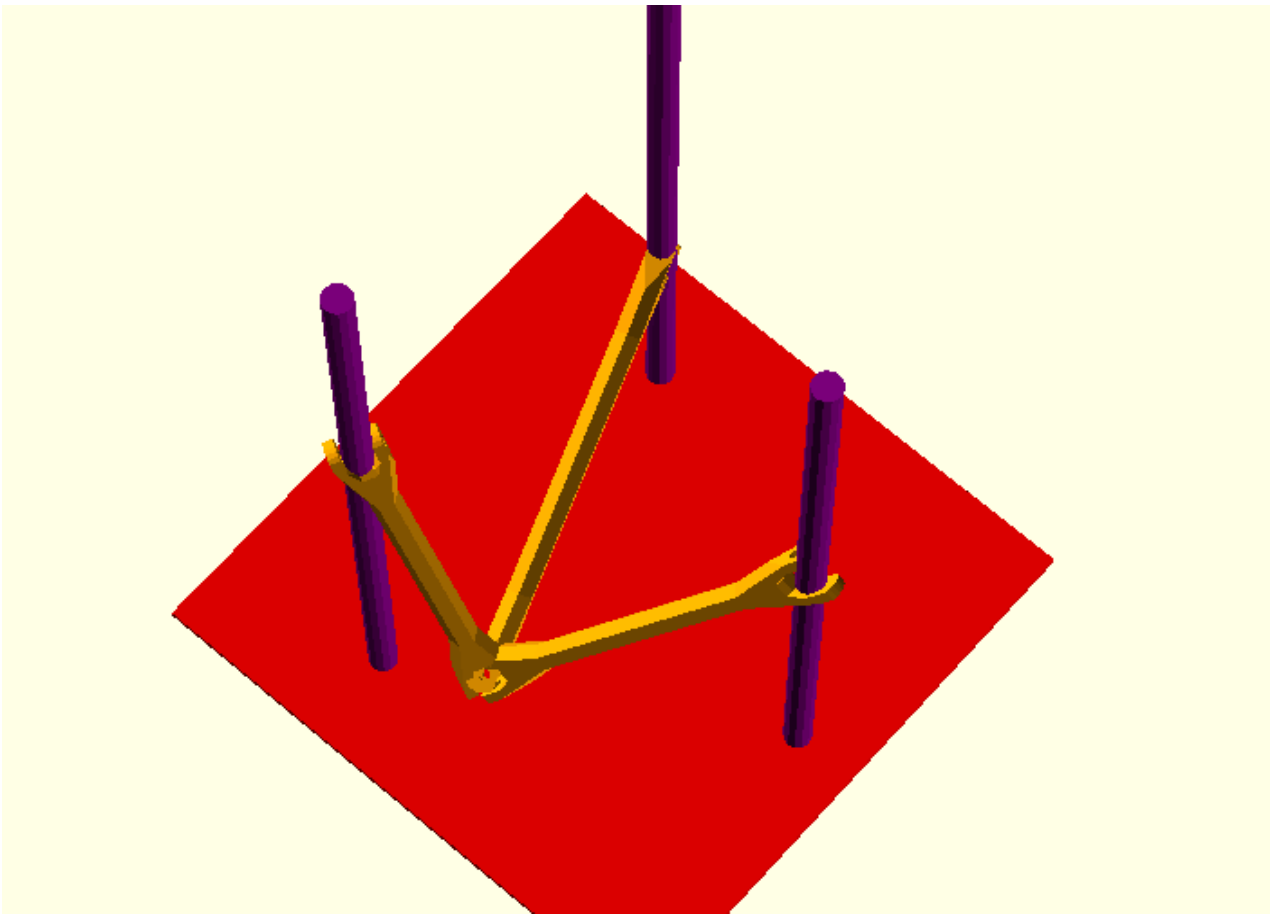


Fig. 6

Now with our simplified model we have the following formulas.

```
(X - Avx)^2 + (Y - Avy)^2 = Ad^2 = L^2 - Acz^2
solve for Acz
Acz^2 = L^2 - (X - Avx)^2 - (Y - Avy)^2
Acz = sqrt(L^2 - (X - Avx)^2 - (Y - Avy)^2)
```

By similarity the formulas for the carriage height above the effector platform `Bcz` and `Ccz` are:

```
Bcz = sqrt(L^2 - (X - Bvx)^2 - (Y - Bvy)^2)
Ccz = sqrt(L^2 - (X - Cvx)^2 - (Y – Cvy)^2)
```

The real values we want is the distance of each carriage above the bed. These formula were given above. Here they are again:

```
Az = Z + Acz + Hcz
Bz = Z + Bcz + Hcz
Cz = Z + Ccz + Hcz
```

These are the formulas used for the inverse kinematics in software.

Now let us check our formulas for validity. If we have columns that are arranged on a circle so that the virtual columns are at radius `DR` from the center 0, 0 and one leg (`A`) is on the `Y` axis we get the following:

```
Avx, Avy => 0,  DR
Bvx, Bvy => DR*cos(30), -DR*sin(30) = DR*sqrt(3)/2, -DR/2
Cvx, Cvy => -DR*cos(30), -DR*sin(30) = -DR*sqrt(3)/2, -DR/2

So
Acz = sqrt(L^2 - (X - Avx)^2 - (Y - Avy)^2) =
sqrt(L^2 - (X - 0)^2 - (Y - DR))^2)

Bcz = sqrt(L^2 - (X - Bvx)^2 - (Y - Bvy)^2) =
sqrt(L^2 - (X - DR*sqrt(3)/2)^2 - (Y + DR/2)^2)

Ccz = sqrt(L^2 - (X - Cvx)^2 - (Y - Cvy)^2) =
sqrt(L^2 - (X + DR*sqrt(3)/2)^2 - (Y + DR/2)^2)
```

`L` and `DR` are known constants, You plug in these and the desired `X` and `Y` and you get `Acz, Bcz` and `Ccz.`

So let's determine the `Acz, Bcz` and `Ccz` for `X,Y => 0,0` they should be equal

The only difference between the three equations is

```
-(X - Avx)^2 - (Y - Avy)^2
-(X - Bvx)^2 - (Y - Bvy)^2
-(X - Cvx)^2 - (Y - Cvy)^2
```

If we set `X` and `Y` to 0 and multiply by -1, these can be rewritten as

```
Avx^2 + Avy^2
Bvx^2 + Bvy^2
Cvx^2 + Cvy^2
```

If these terms are equal the equations are equal

```
Avx^2 + Avy^2 = 0 + DR^2 = DR^2

Bvx^2 + Bvy^2 = (DR*sqrt(3)/2)^2 + (-DR/2)^2 = (DR^2)/4 + (DR^2 * 3)/4
```
factor out the /4 and combine terms
```
(4*DR^2)/4 = DR^2

Cvx^2 + Cvy^2 = (-DR*sqrt(3)/2)^2 + (-DR/2)^2 = (DR^2)/4 + (DR^2 * 3)/4
=
```
factor out the /4 and combine terms
```
(4*DR^2)/4 = DR^2
```

The equations are equal, our formulas are valid for this scenario.

FORWARD KINEMATICS

Forward kinematics are the formulas which determine the `X-Y-Z` coordinates of the effector platform when given the carriage positions. They are a little more difficult than the inverse kinematics. Our mathematical model using the virtual column positions is still useful. If we use the virtual column positions the center of the effector platform is the point that is L away from each carriage position on the virtual column. This gives the following equations.

```
(X - Avx)^2 + (Y - Avy)^2 + Acz^2 = L^2
(X - Bvx)^2 + (Y - Bvy)^2 + Bcz^2 = L^2
(X - Cvx)^2 + (Y - Cvy)^2 + Ccz^2 = L^2
```

Where `Acz, Bcz` and `Ccz` are the height of each carriage above the plane of the effector platform and relate to `Z` as follows

```
Acz = Az - Z - Hez
Bcz = Bz - Z - Hez
Ccz = Cz - Z - Hez
```

`Az, Bz` and `Cz` are the `Z` components of the given carriage positions.

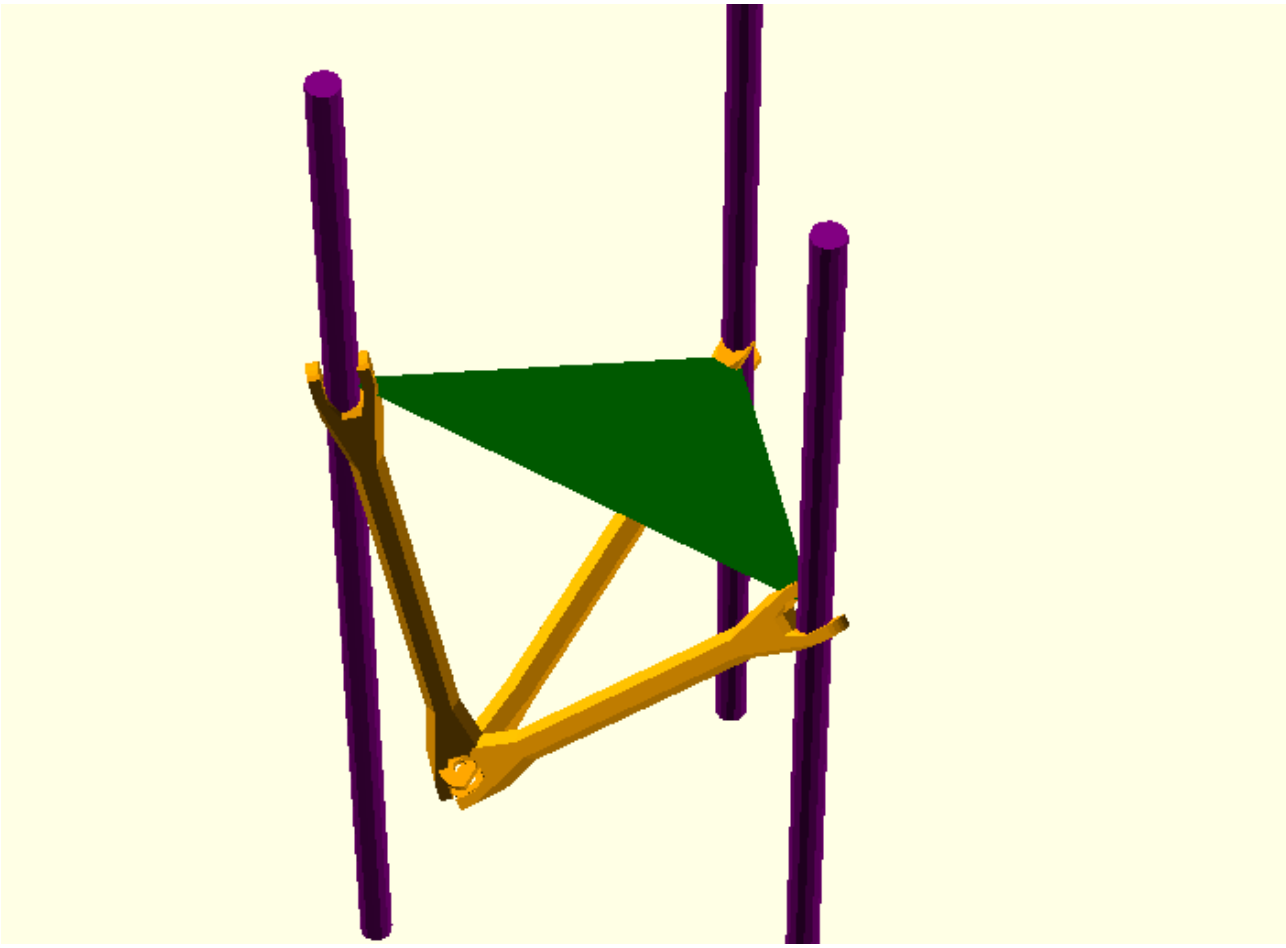


Fig. 7

We have three equations and three unknowns but the squares make these difficult equations to solve.

I had originally used a solution for the forward kinematics based on the circumcenter, but thanks to Michael Paauwe from New Zealand I have now used a better method based on Trilateration. (see Trilateration in Wikipedia to see the details).

Trilateration means finding a point in 3D space based on the distances from 3 known points. This is exactly how GPS works.

Our three known points are the carriage positions. Below is code from the Java program I wrote to do analysis of the delta kinematics. It shows how one can use the Wikipedia discussion to derive code to do forward kinematics.

```
double[] forwardKinematics(double[] dZ)
{
     double[][] dColP = new double[3][3];
     for(int iIdx = 0; iIdx < 3; iIdx++)
     {
          dColP[iIdx][0] = dCol[iIdx][0];
          dColP[iIdx][1] = dCol[iIdx][1];
          dColP[iIdx][2] = dZ[iIdx];
     }
     //dColP has the three points on the columns
     //As discussed in Wikipedia "Trilateration"
```

```
//we are establishing a new coordinate
//system in the plane of the three carriage points.
//This system will have the origin at dColP[0] and
//dColP[1] is on the x axis.  dColP[2] is in the X-Y
//plane with a Z component of zero.  We will define unit
//vectors in this coordinate system in our original
//coordinate system.  Then when we calculate the
//Xnew, Ynew and Znew values, we can translate back into
//the original system by moving along those unit vectors
//by the corresponding values.  We try to use the
//variable names from the Wikipedia article.

//Create a vector in old coords along x axis of new coord
double[] p12 = vectorSub(dColP[1], dColP[0]);

//Get the Magnitude of vector.
double d = vectorMag(p12);

//Create unit vector by dividing by magnitude.
double[] ex = vectorMult(p12, 1/d);

//Now find vector from the origin of the new system to
//the third point
double[] p13 = vectorSub(dColP[2], dColP[0]);

//Now use dot product to find the component of this
//vector on the X axis
double i = vectorDotProd(ex, p13);

//Now create a vector along the x axis that represents
//the x component of p13
double[] iex = vectorMult(ex, i);

//Now subtract the X component away from the original
//vector leaving only the Y component.  We use the
//variable that will be the unit vector after we scale
//it.
double[] ey = vectorSub(p13, iex);

//The magnitude of Y component
double j = vectorMag(ey);

//Now make vector a unit vector
ey = vectorMult(ey, 1/j);

//The cross product of the unit x and y is the unit z
double[] ez = vectorCrossProd(ex, ey);

//Now we have the d, i and j values defined in Wikipedia.
//We can plug them into the equations defined in
//Wikipedia for Xnew, Ynew and Znew
//Xnew = (r1^2 - r2^2 - d^2)/2*d
//     = (L^2 - L^2 - d^2)/2d = d/2;
```

```
        double Xnew = d/2;

        //Ynew = (r1^2 - r3^2 + i^2 + j^2)/2*j - i * Xnew/j
        //     = ((L^2 - L^2 + i^2 + j^2)/2 - i * Xnew)/j
        //     = ((i^2 + j^2)/2 - i * Xnew)/j
        double Ynew = ((i * i + j * j)/2 - i * Xnew)/j;

        //Znew = sqrt(r1^2 - Xnew^2 - Ynew^2)
        //     = sqrt(L^2 - Xnew^2 - Ynew^2)
        double Znew =
            Math.sqrt(dArmSqrd - Xnew * Xnew - Ynew * Ynew);

        //Now we can start from the origin in the old coords and
        //add vectors in the old coords that represent the
        //Xnew, Ynew and Znew to find the point in the old system
        double[] cartesian =
           vectorAdd(dColP[0], vectorMult(ex, Xnew));
        cartesian = vectorAdd(cartesian, vectorMult(ey, Ynew));
        cartesian = vectorAdd(cartesian, vectorMult(ez, -Znew));
        return cartesian;
    }
```

More Delta Analysis

Now that we have the formula for forward and inverse kinematics, we can answer some often asked questions. Two of the most common questions are: “What is the shape of the printing area?” and “How do errors in the carriage positions affect the effector position?” To that end I have written a program that creates a color topological map showing the print shape and the possible error. Based on the above discussion, we can analyze an X-Y plane and the results can be translated to any other X-Y plane.

I have written a program that steps through the X-Y space and uses the inverse kinematics to calculate the carriage positions needed for that X-Y location. I then vary each column from the target value by the error value. There are a couple of modes. In the multiple column mode I try all combinations of error (+, 0, -) with all three columns (26 combinations, the 27th is all 0s) and look for the maximum error. In single column mode I vary each column between – and + holding the other two columns at 0 (6 combinations) and look for maximum error. I also have some other modes they are: Error in X, error in Y, error in Z, error in X-Y and error in X-Y-Z. A note about Z. Even though we are working in one given Z plane, the resulting carriage positions could cause the effector to leave that plane. Hence Z error.

A further discussion of error is in order. I have tried to do some research on this but I haven't found anything that makes me feel like I have a handle on the subject. So I don't feel qualified for this discussion, but here goes. As an engineer, I take any case of making measurements and think about a probability distribution. In many cases one would approximate a probability distribution with a Gaussian function. But to my thinking, the type of mechanism determines the error. An acme screw, for example, must have some backlash. One is almost sure to have some error. So the error distribution is not necessarily centered on 0 and is probably not Gaussian. On the other hand, the belt drives used in the DIY 3d printers probably have an error that is more distributed around 0 and the distribution is more likely to be Gaussian. I mention this because when we are analyzing the error in the delta, one must take into account that there are three sources contributing to the error.

One can calculate the maximum error at a given point based on the maximum error in all three columns, but one must be aware that the error distribution has changed. For example, if there is a 10% probability for the maximum error in a given column, then the chances of the maximum error in all three is 10% * 10% * 10% (0.1 * 0.1 * 0.1 * 100) or 0.1%. But in the case of the acme screw where that error is more likely, then the multiple column error has more meaning. So I have created two modes for measuring error. As always, the truth is somewhere in between.

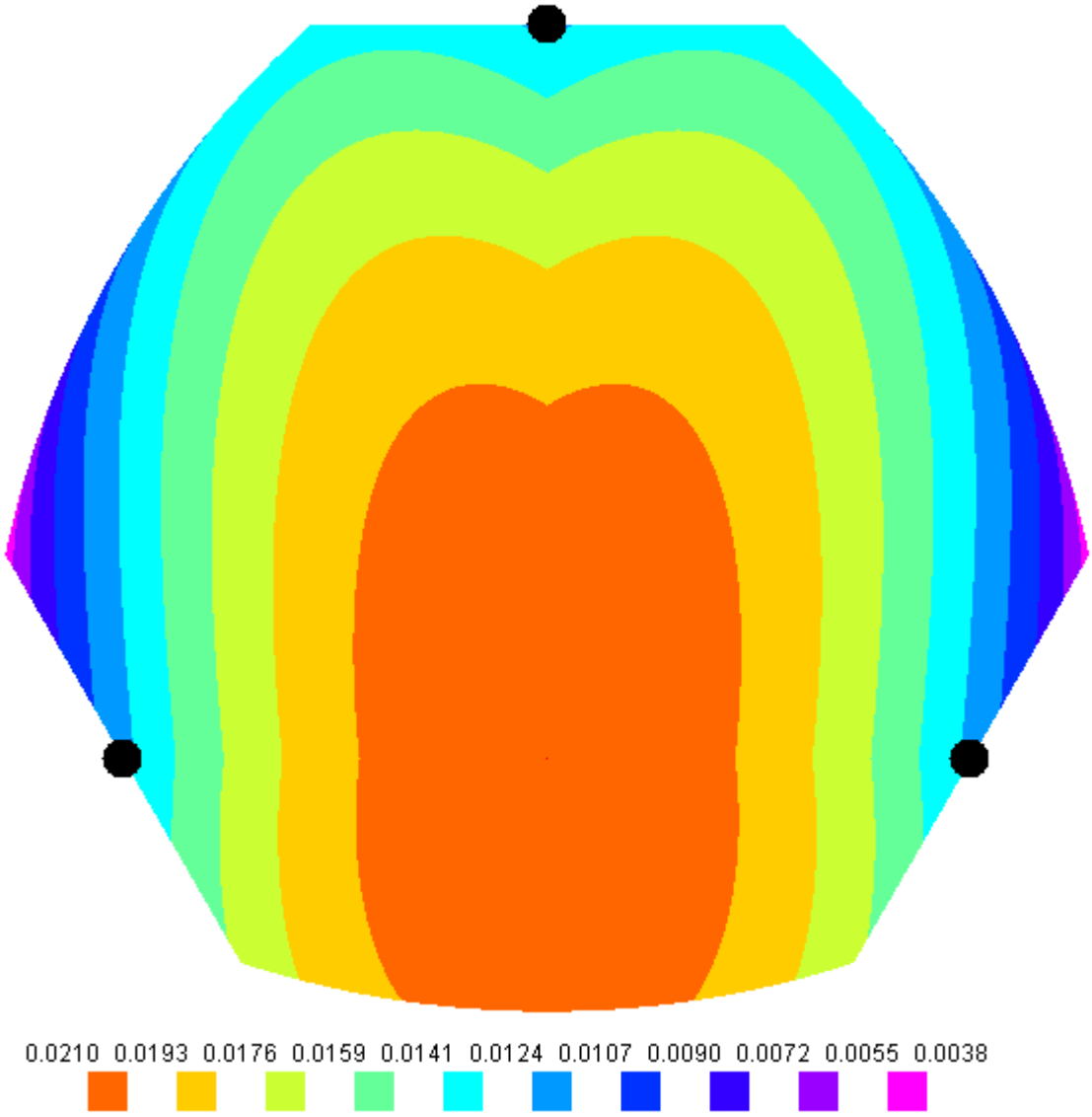
Note that each image has its own color legend. For each set of calculations the range of errors is divided into 10 regions and is assigned the colors shown in the legend at the bottom. This first set of images uses the values I found in Johann's Marlin code. The virtual radius is 124mm and the arm length is 250mm.

Following the first set of images which show various error modes are some images showing the printing area for different arm and radius combinations. These patterns would scale based on the ratio between arm and radius.

The printing area shown is based on the intersection of the circle around each virtual column. It assumes that the arms can swing 180 degrees. There is a further discussion about this below.

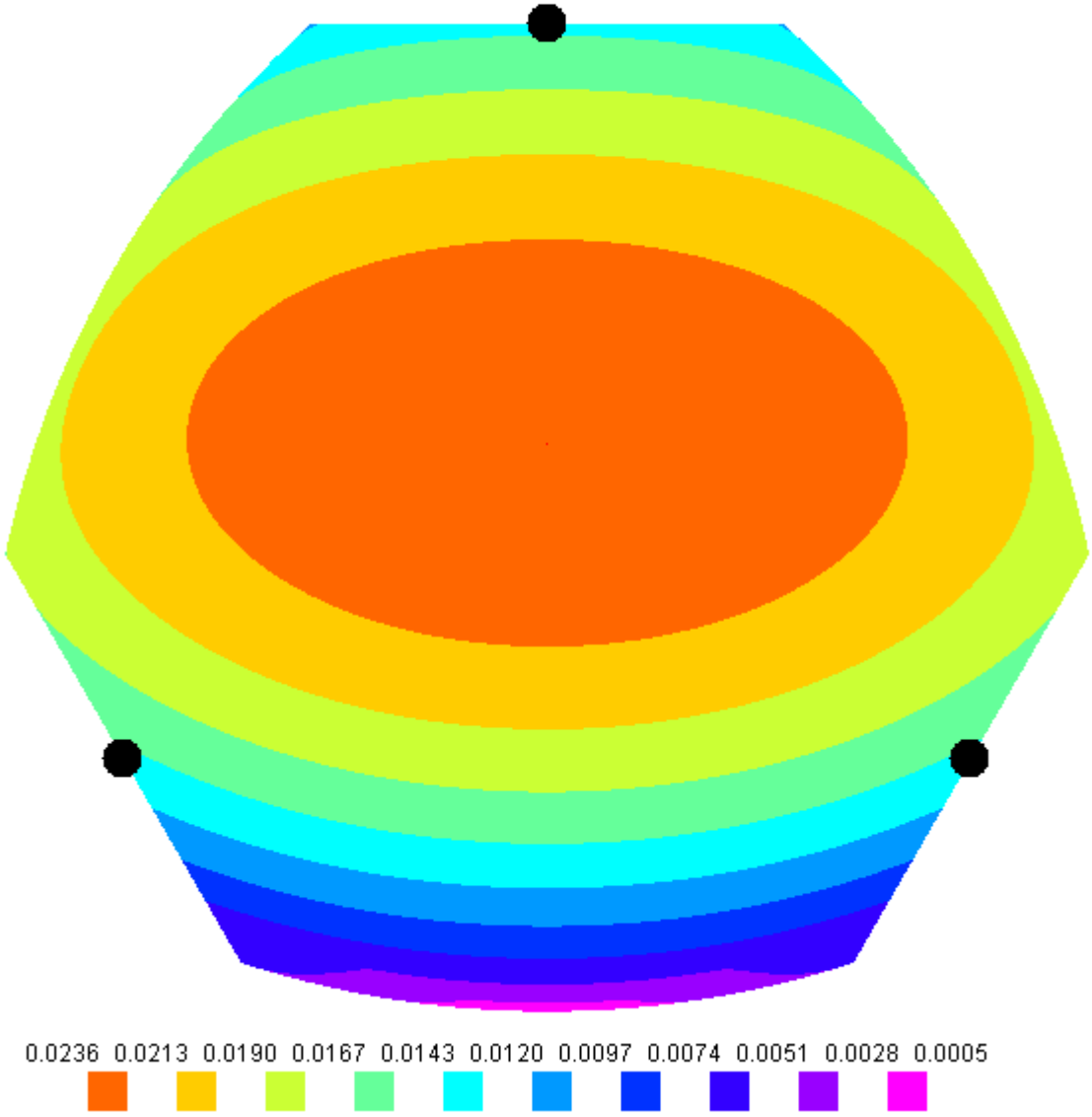
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in X



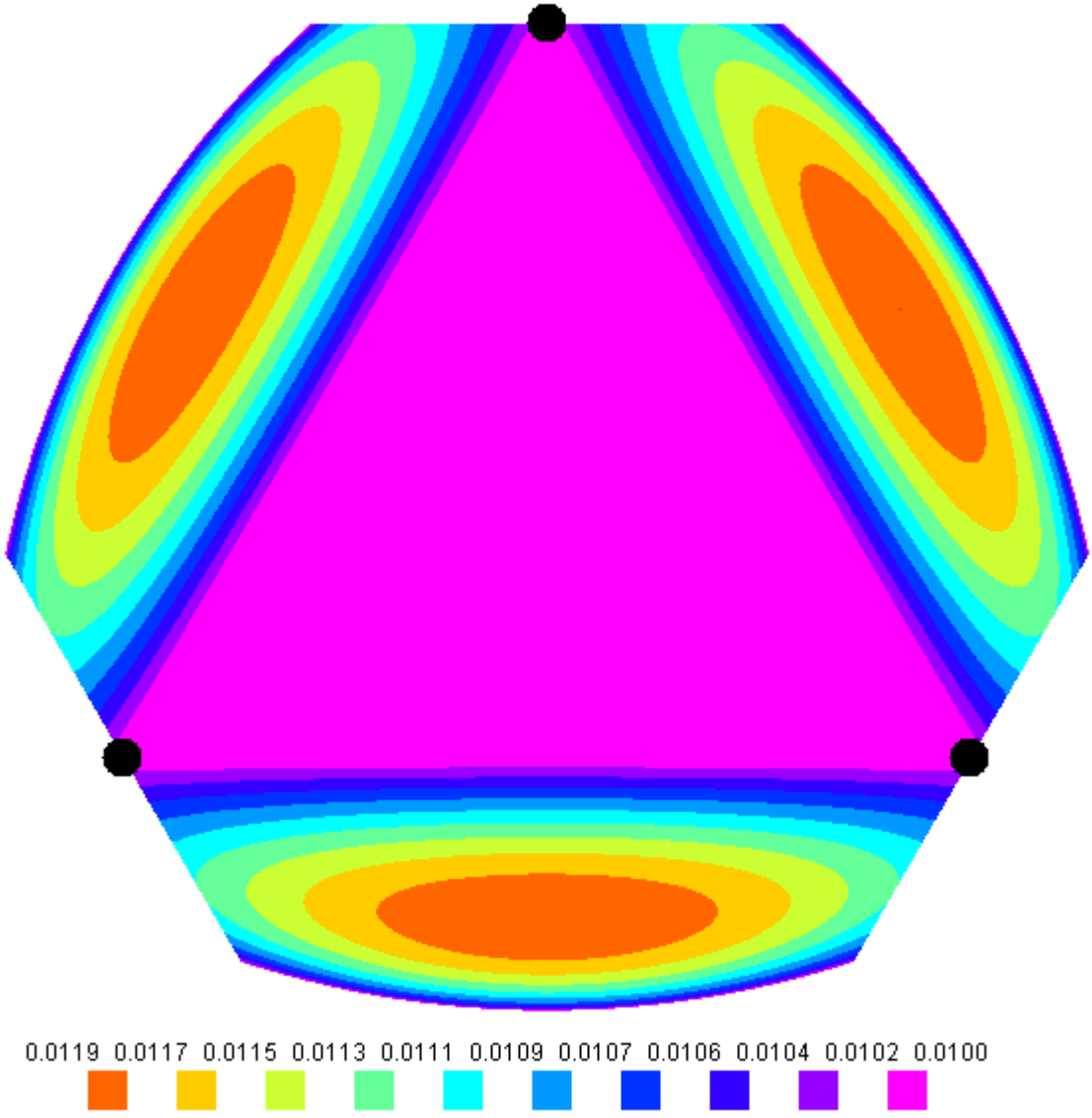
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in Y



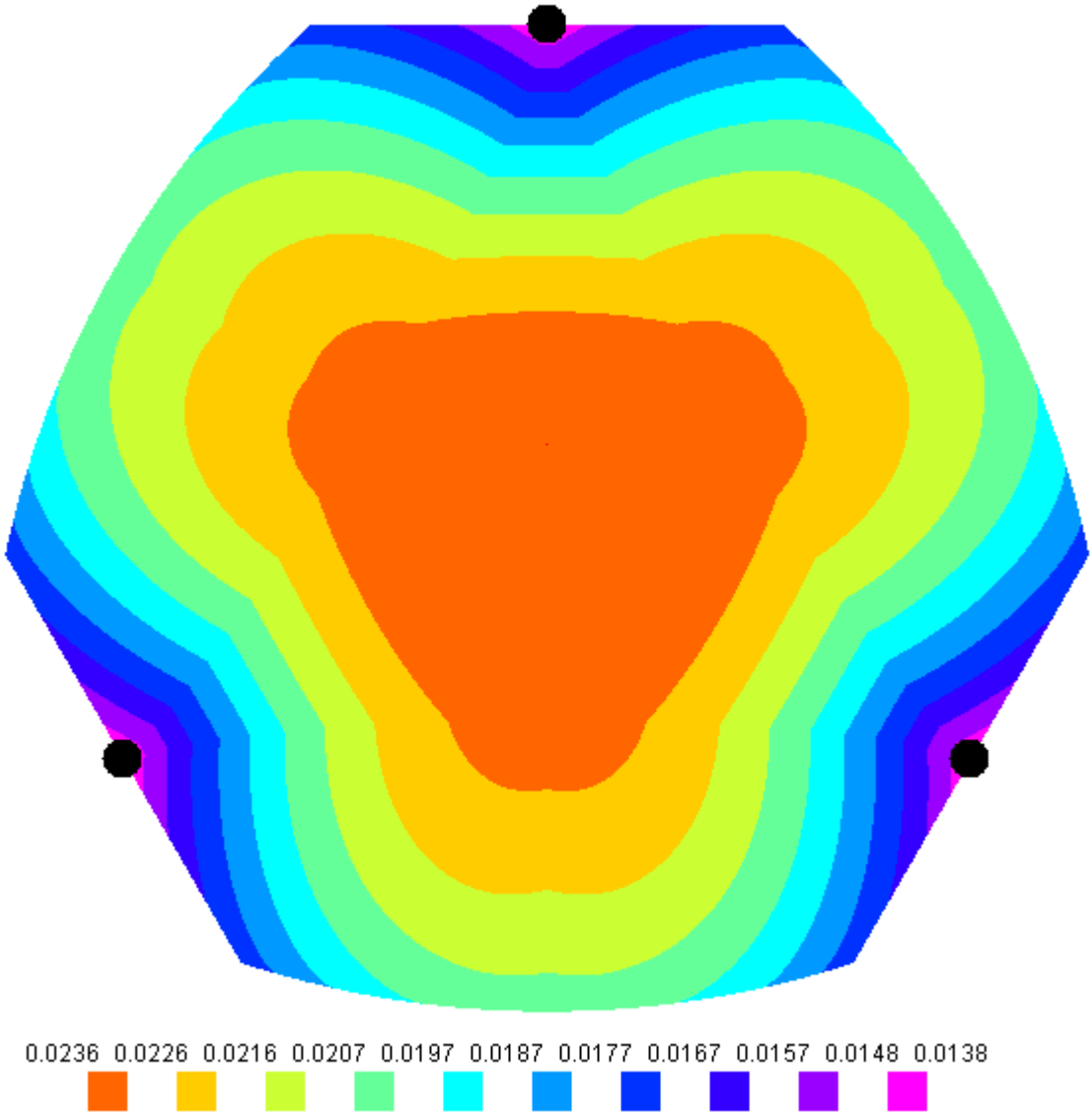
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in Z



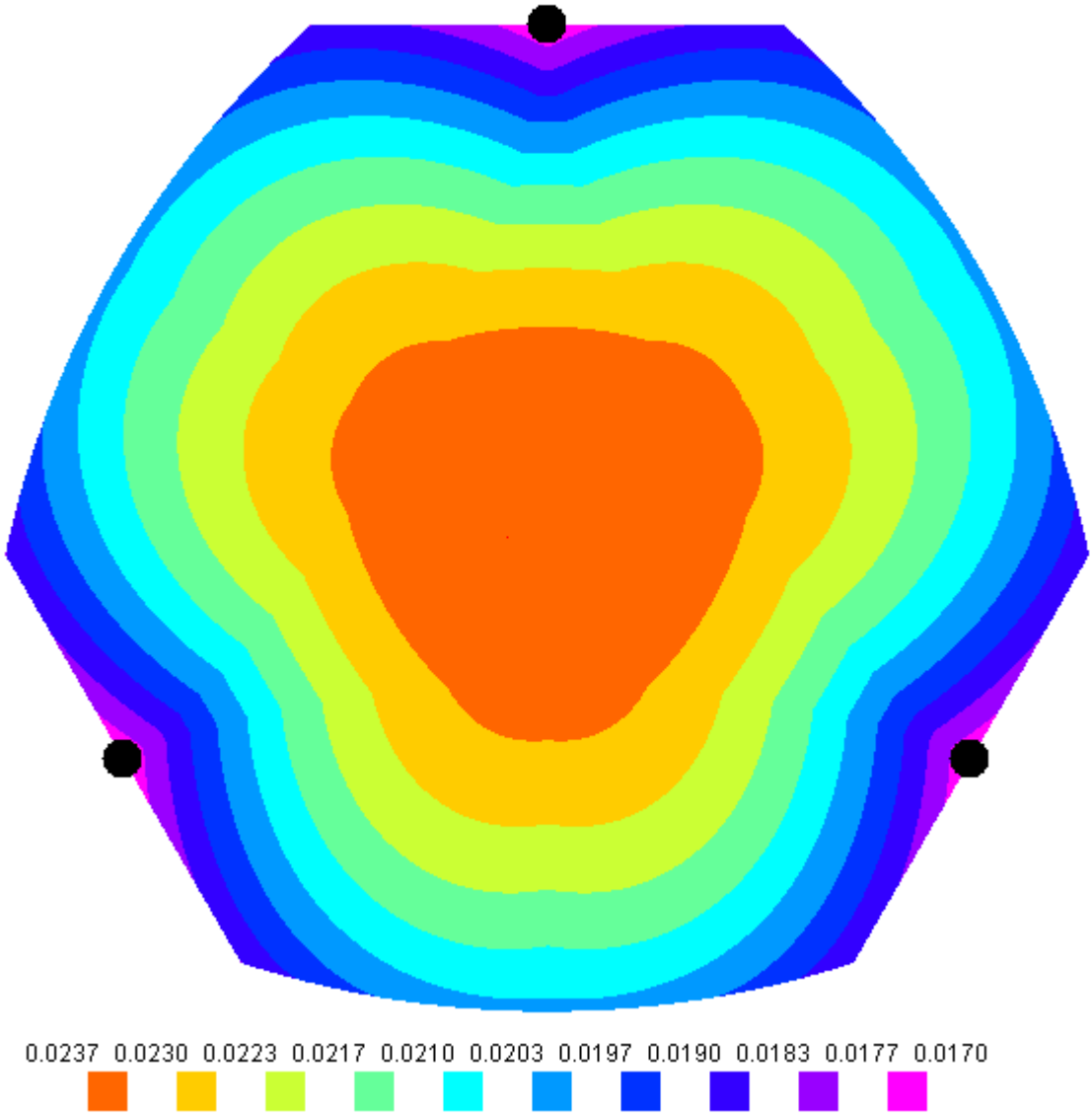
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in X-Y



Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

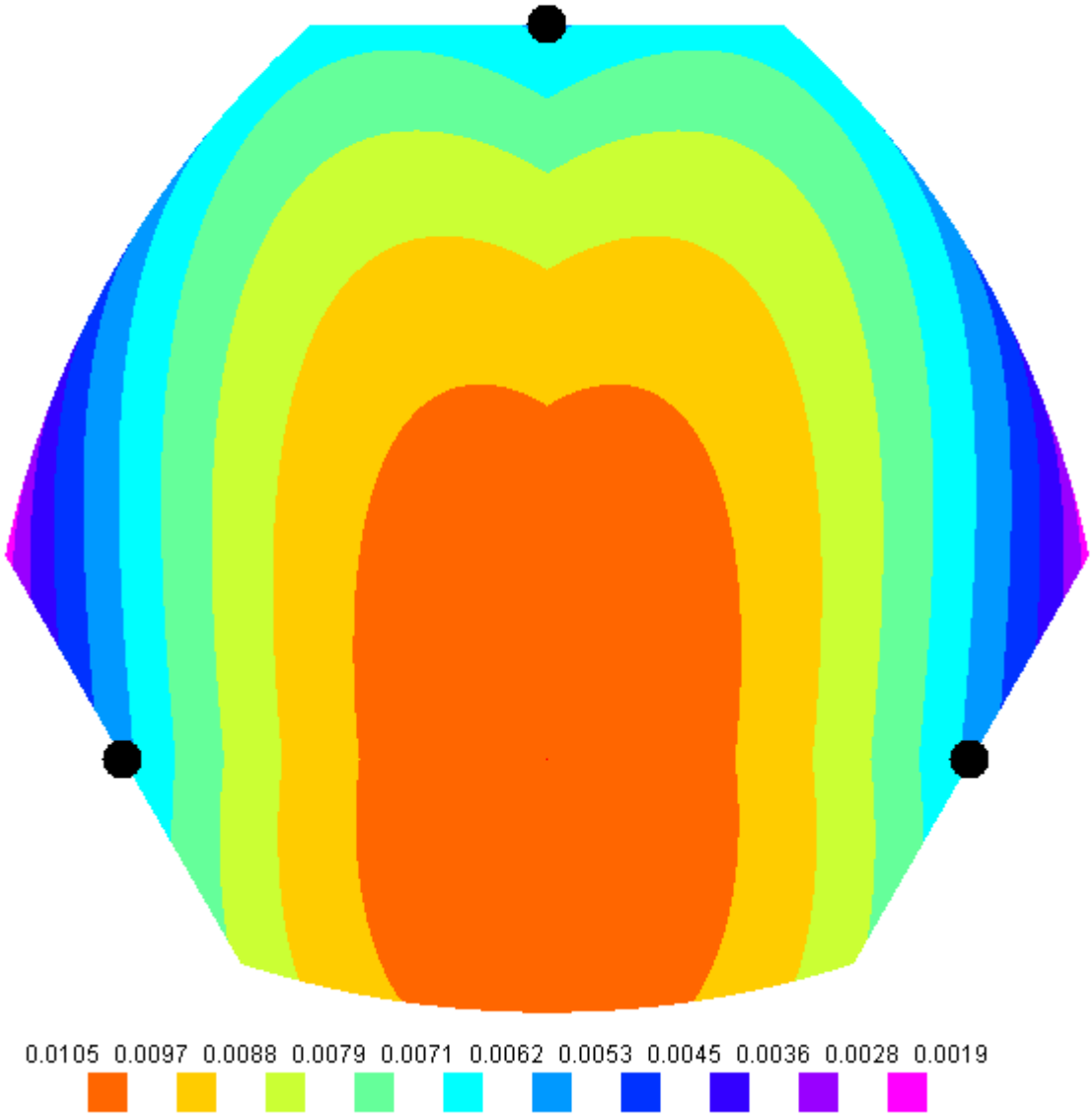
Max Error in X-Y-Z



0.0237 0.0230 0.0223 0.0217 0.0210 0.0203 0.0197 0.0190 0.0183 0.0177 0.0170

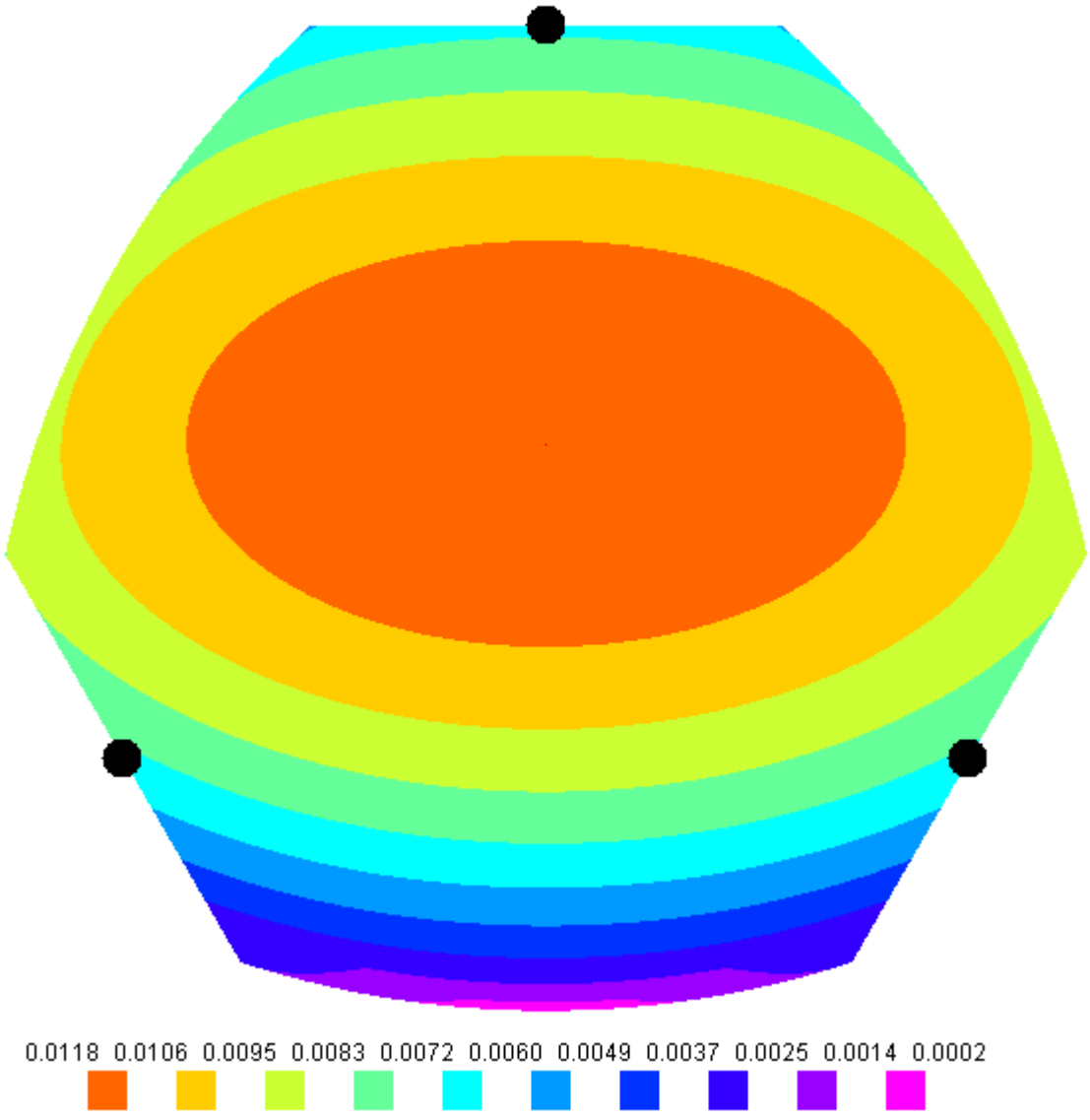
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Single Column

Max Error in X



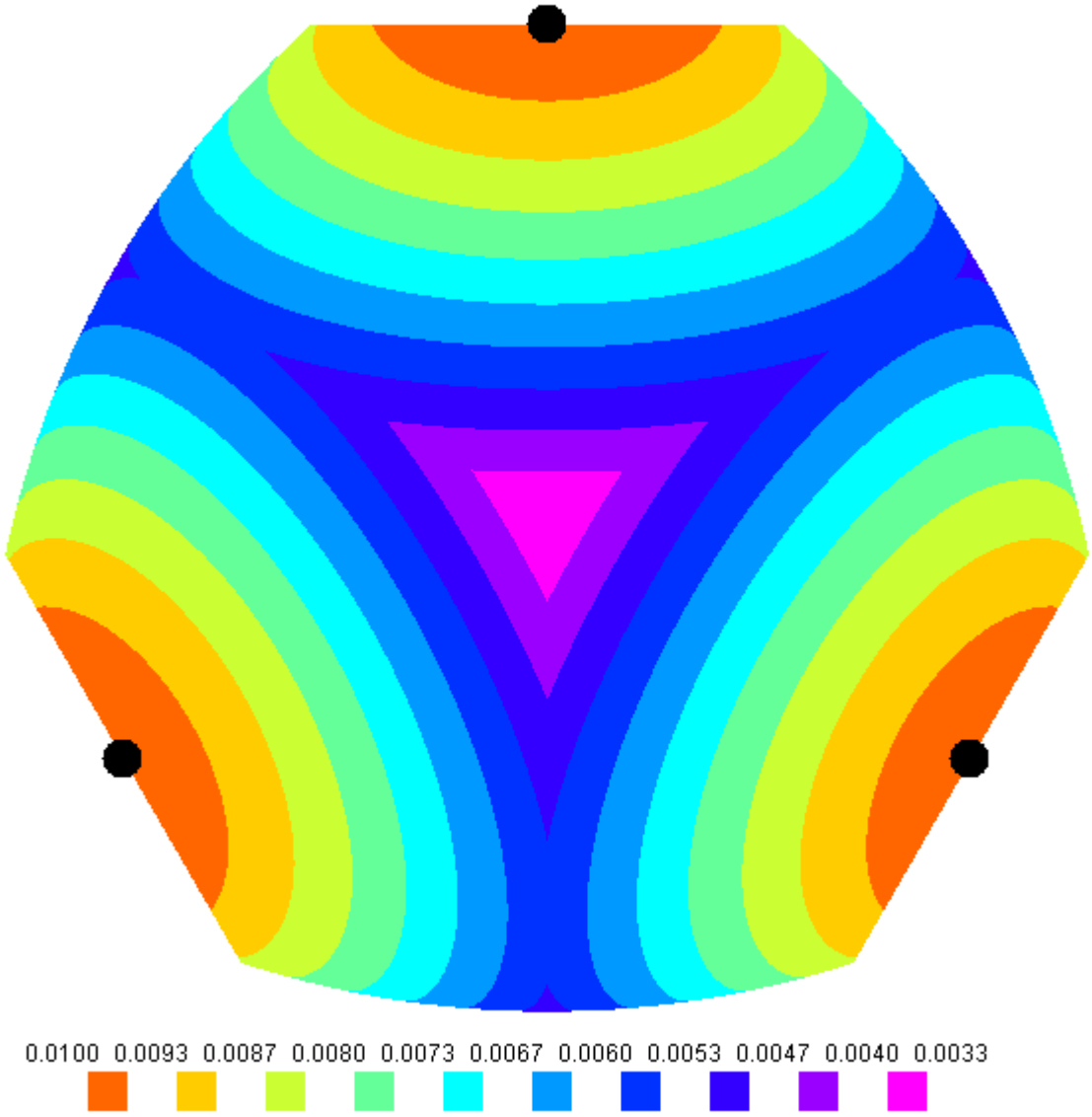
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Single Column

Max Error in Y



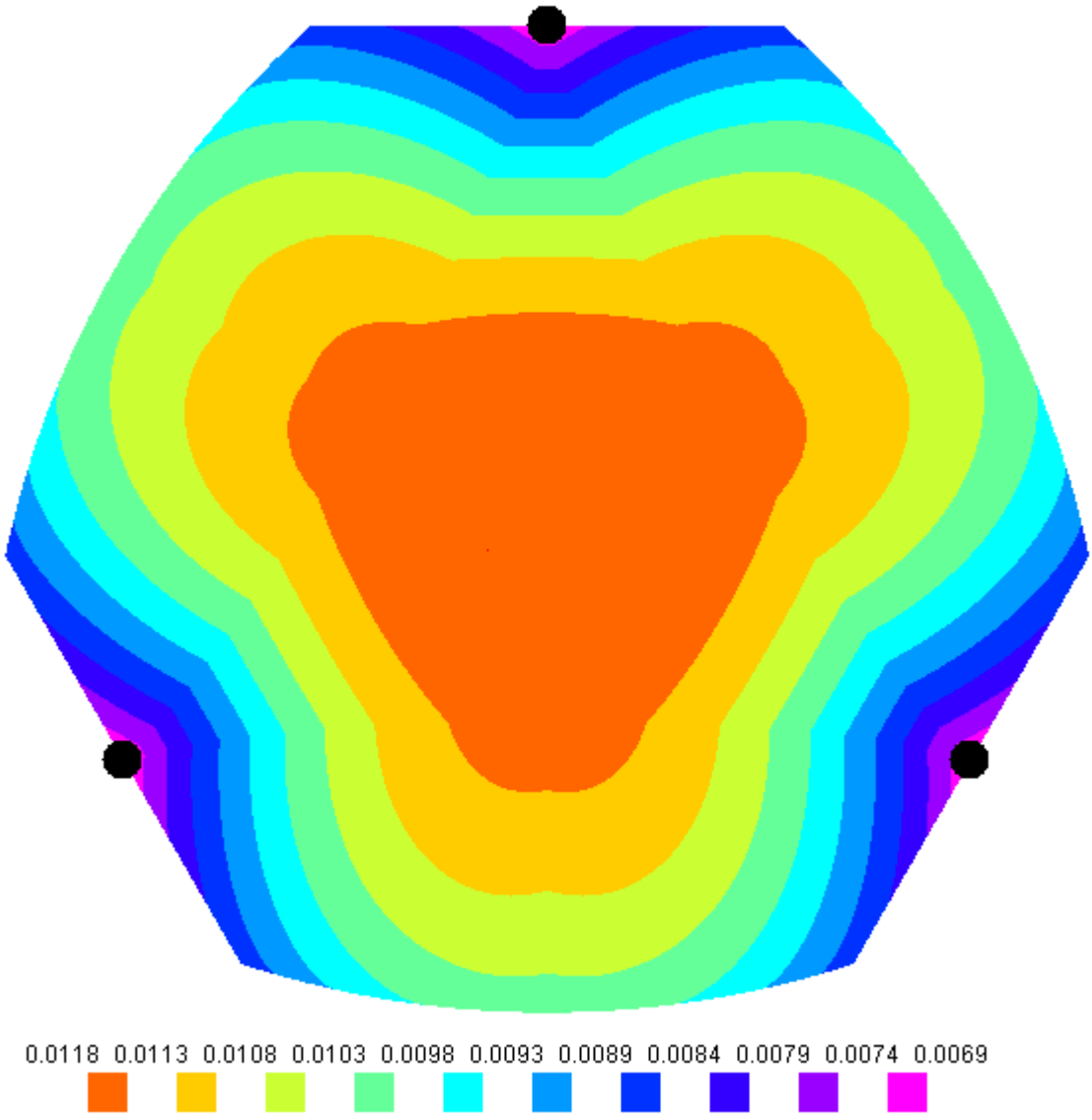
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Single Column

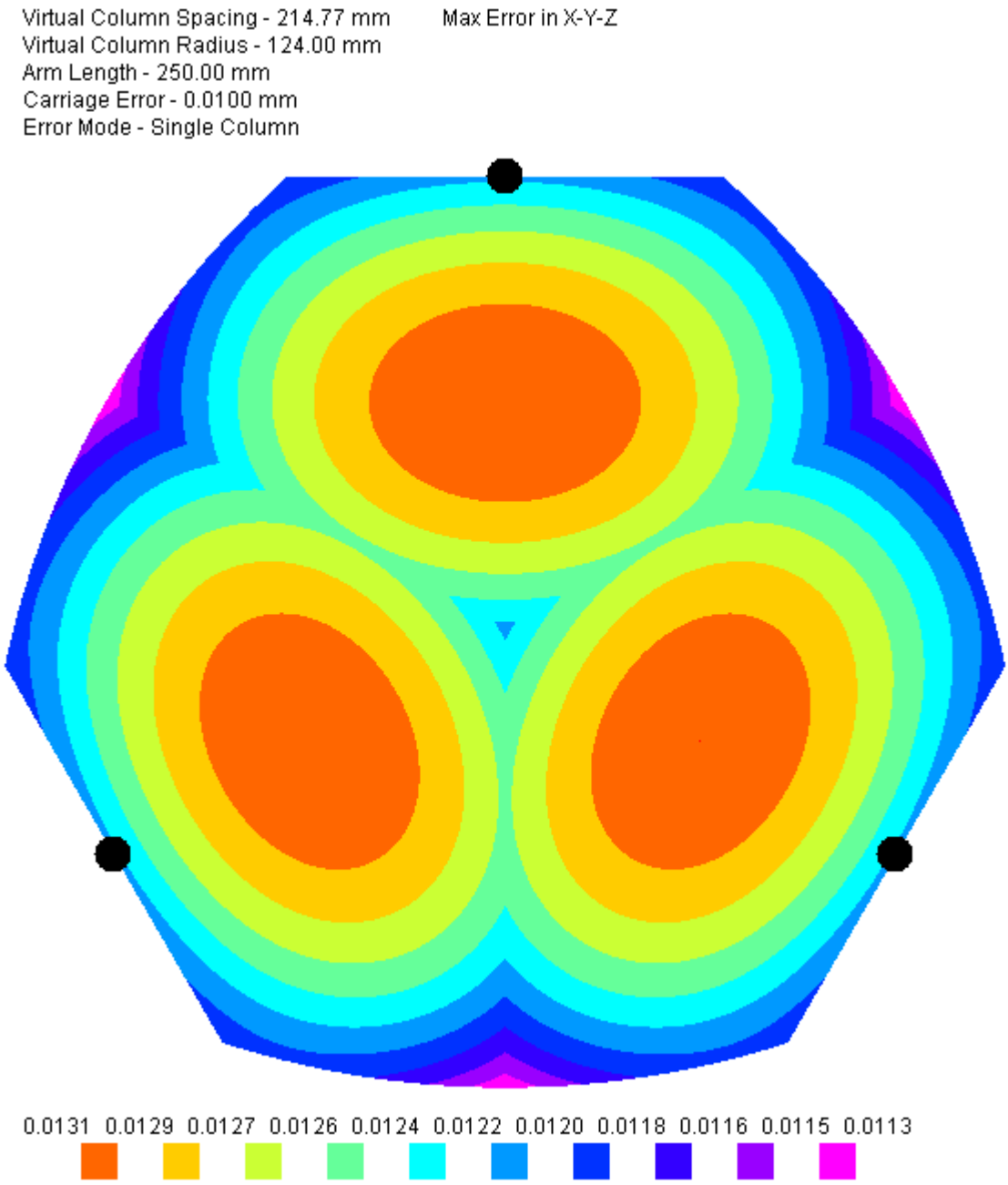
Max Error in Z



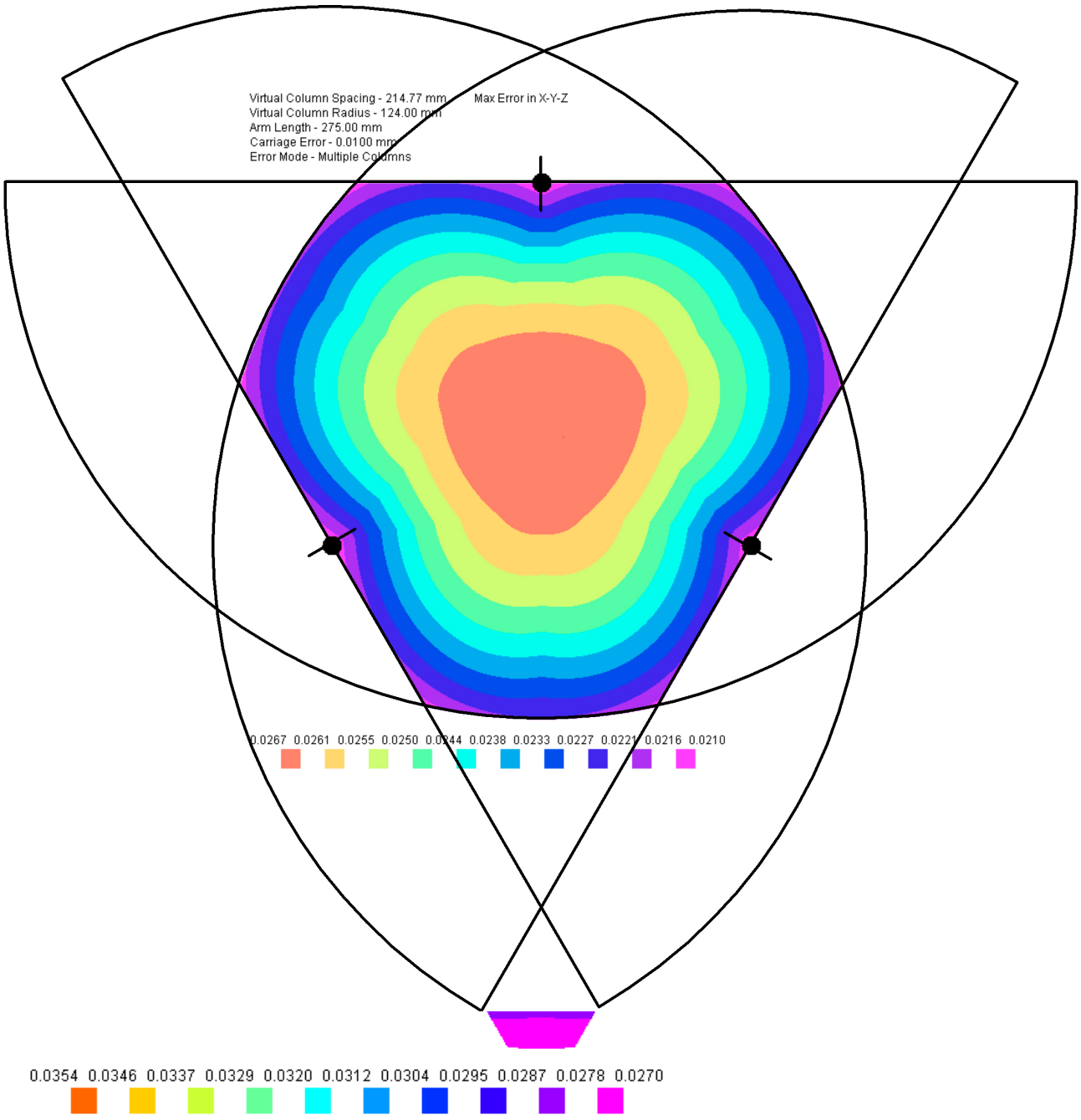
Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 250.00 mm
Carriage Error - 0.0100 mm
Error Mode - Single Column

Max Error in X-Y



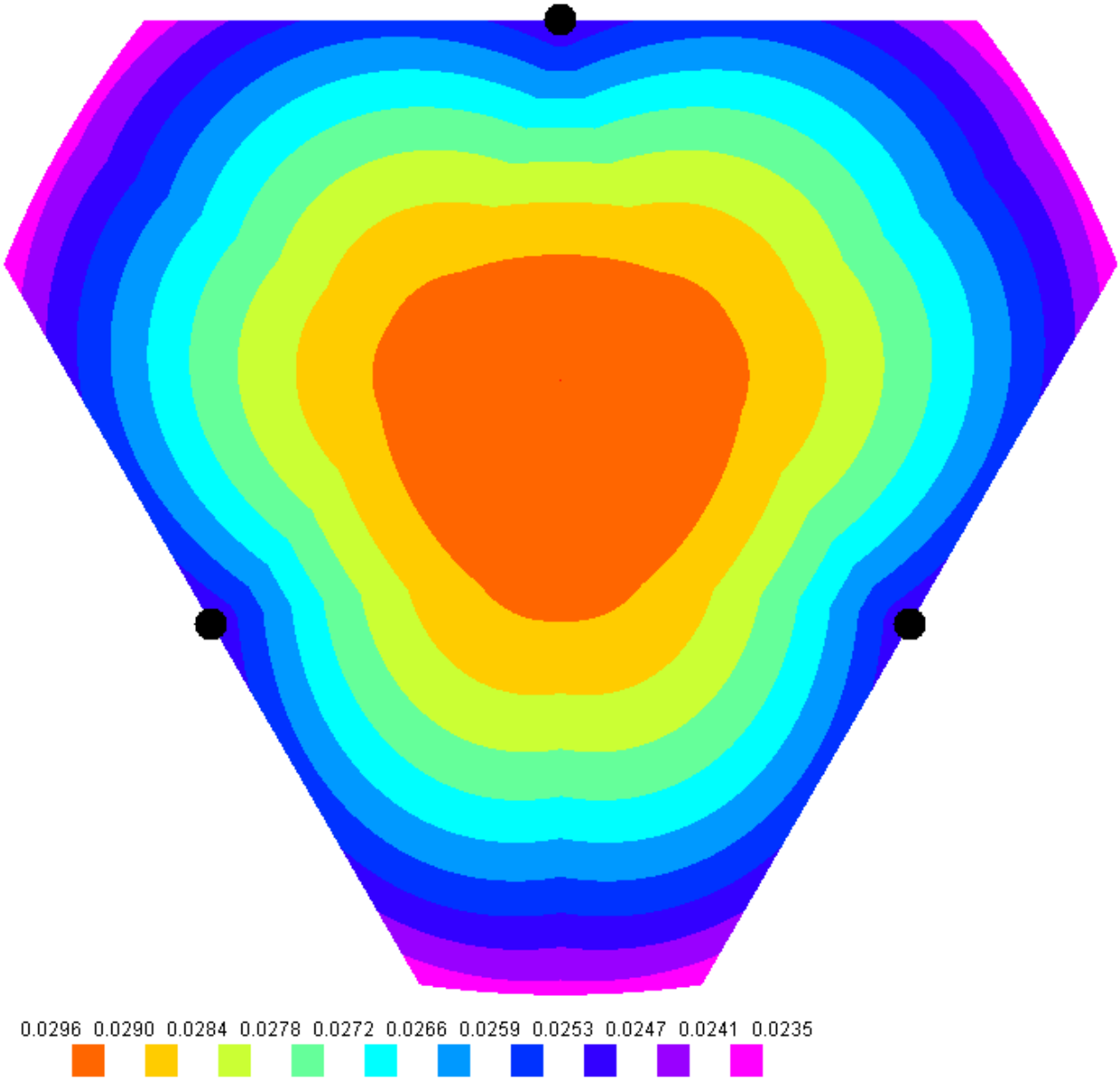


The virtual columns make a lot of analysis simplier. This includes analysis of the print area. To determine the print area one just takes the overlap of the possible reach for each arm. The following pictures show an assumed 180 degree reach. In reality, most delta designs can not make a 180 degree swing, so these are theoretical shapes. In most designs (the Rostock can) it is possible for the arms to go vertical, so the shape does include the virtual column. From there the shapes would angle at the limits of the arm swing forming a pie shape at less than 180 degrees. These intersect the circles around the other two columns. The cool thing here is that one can use graphic methods to play around with different column positions and arm lengths.


Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 275.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns
Max Error in X-Y-Z
0.0267 0.0261 0.0255 0.0250 0.0244 0.0238 0.0233 0.0227 0.0221 0.0216 0.0210
0.0354 0.0346 0.0337 0.0329 0.0320 0.0312 0.0304 0.0295 0.0287 0.0278 0.0270

Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 300.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in X-Y-Z



0.0296 0.0290 0.0284 0.0278 0.0272 0.0266 0.0259 0.0253 0.0247 0.0241 0.0235

Virtual Column Spacing - 214.77 mm
Virtual Column Radius - 124.00 mm
Arm Length - 200.00 mm
Carriage Error - 0.0100 mm
Error Mode - Multiple Columns

Max Error in X-Y-Z