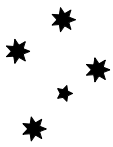


PRIME OPTICS

Extracting Asphere Vertex Radius of Curvature from ROMER Arm Measurement Point Cloud

2021 July 29

Telephone :	+61-7-5442 8813	*Local Time :*	UT + 10
SMS :	+61-429-02 8831	*EMail :*	damien@primeoptics.com.au
Contact :	**Damien Jones**	*Web :*	www.primeoptics.com.au

1 Introduction

The impetus for this note is the need to measure the profile of an aspheric surface and find the vertex curvature.

Specifically, point clouds of the asphere are generated by a ROMER Arm. It is understood that even though the point clouds can be generated with great precision; the surface cannot be centred exactly, or have zero tilt, so that these parameters also need to be found.

Another note ("Coordinate System Alignment for Optical Surface Modelling") describes the geometrical transformations that are required and how to do them.

The unknown parameters are not an orthonormal set and are vastly outnumbered by the data point population, so no analytic solution is feasible. Thus, optimisation methods must be used.

2 ROMER Measurement Ball Geometry

There are 2 coordinate systems in play.

In the following sketch the ROMER reports point **S** by projecting a radius from the centre of the [small] measurement ball to the virtual centre of curvature (**CoC**) of the reference sphere in the global coordinate system [machine space]. **S** is transformed to **S'** in the local coordinate system [local space] of the aspheric surface.

The measurement ball is in contact with the aspheric surface at an unknown point **A**, in local space. We need to calculate the vector **S'A** via iteration; the first of which will define $\mathbf{S'A_1}$.

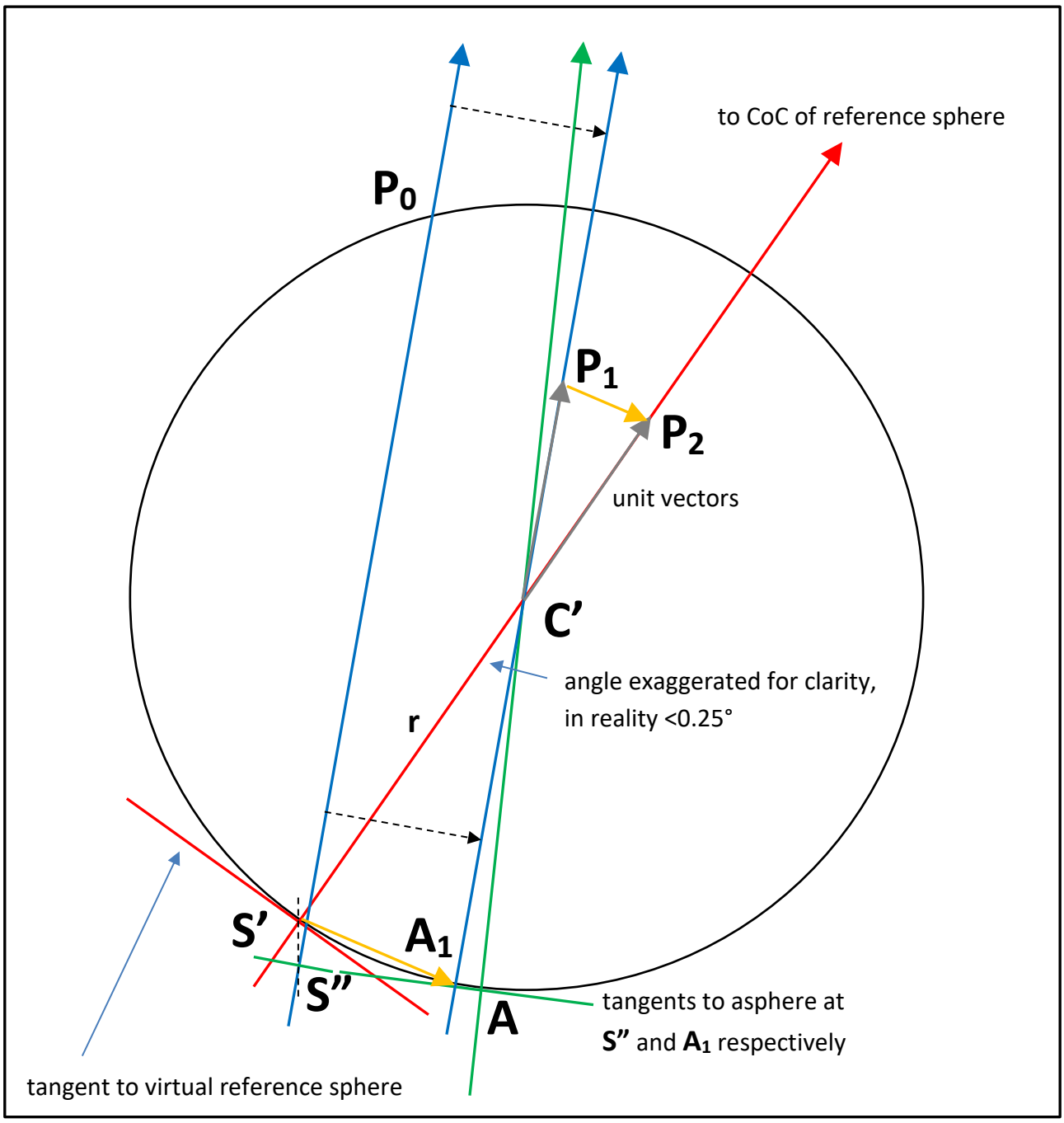
The normal vector to the ball surface at **S** is simply the line joining **S** with the **CoC**. It passes through the centre of the ball (point **C** in the machine space, **C'** in the local space) by definition. This vector is transformed into the local space, normalized, and translated, forming the vector $\mathbf{C'P_2}$ in the sketch.

For the first iteration, the normal to the asphere is calculated at the **X** and **Y** coordinates of **S'** to form $\mathbf{S''P_0}$. It is translated so as to pass through **C'**, and then normalised to form the unit vector $\mathbf{C'P_1}$.

Then, by similar triangles:

$$\overrightarrow{S'A_1} = r \cdot \overrightarrow{P_1P_2}$$

The normal to the asphere is then calculated at the **X** and **Y** coordinates of $\mathbf{A_1}$ and the process is repeated until **A** is approached to the desired precision. 2 - 3 iterations are usually sufficient to attain a precision of <1E-6 mm.


to CoC of reference sphere
P0
P1
P2
unit vectors
C’
angle exaggerated for clarity,
in reality <0.25°
r
S’
A1
S”
A
tangents to asphere at
S” and A1 respectively
tangent to virtual reference sphere

3 Fitting a Measurement Point Cloud

There follows a DELPHI/PASCAL code-kernel for matching the vertex curvature of an aspheric profile with a point cloud generated by measurement instrumentation. Note that there is no z-rotation; but in principle this would be straightforward to include.

The variables passed to the optimiser are the origin translations, theta_x, theta_y and the vertex curvature, c.

The final surface is quite close to nominal so there is no real advantage to be gained by allowing the aspheric coefficients and the conic constant to be used by the optimiser. We are only interested in the vertex curvature.

The point cloud is also passed to the optimiser as the fixed reference.

It has been observed that when the origin x- and y-translations are included in the optimiser's variable space, the global minimum is rather broad and the end result is not reproducible from random inputs of the optimiser variables. It is often better to run the optimiser a few times and obtain best guesses of the translations. These can then be set by hand and "walked in" to the global minimum. It is observed that minima achieved in this manner are highly reproducible.

Note that the code kernel returns a figure of merit for the entire point cloud.

In this instance the code kernel was used with a downhill SIMPLEX optimiser. But in principle, any suitable optimiser could be deployed.

4 Measurement Scatter and Reliability

The figure below shows a typical measurement cloud scatter from a large concave aspheric surface.

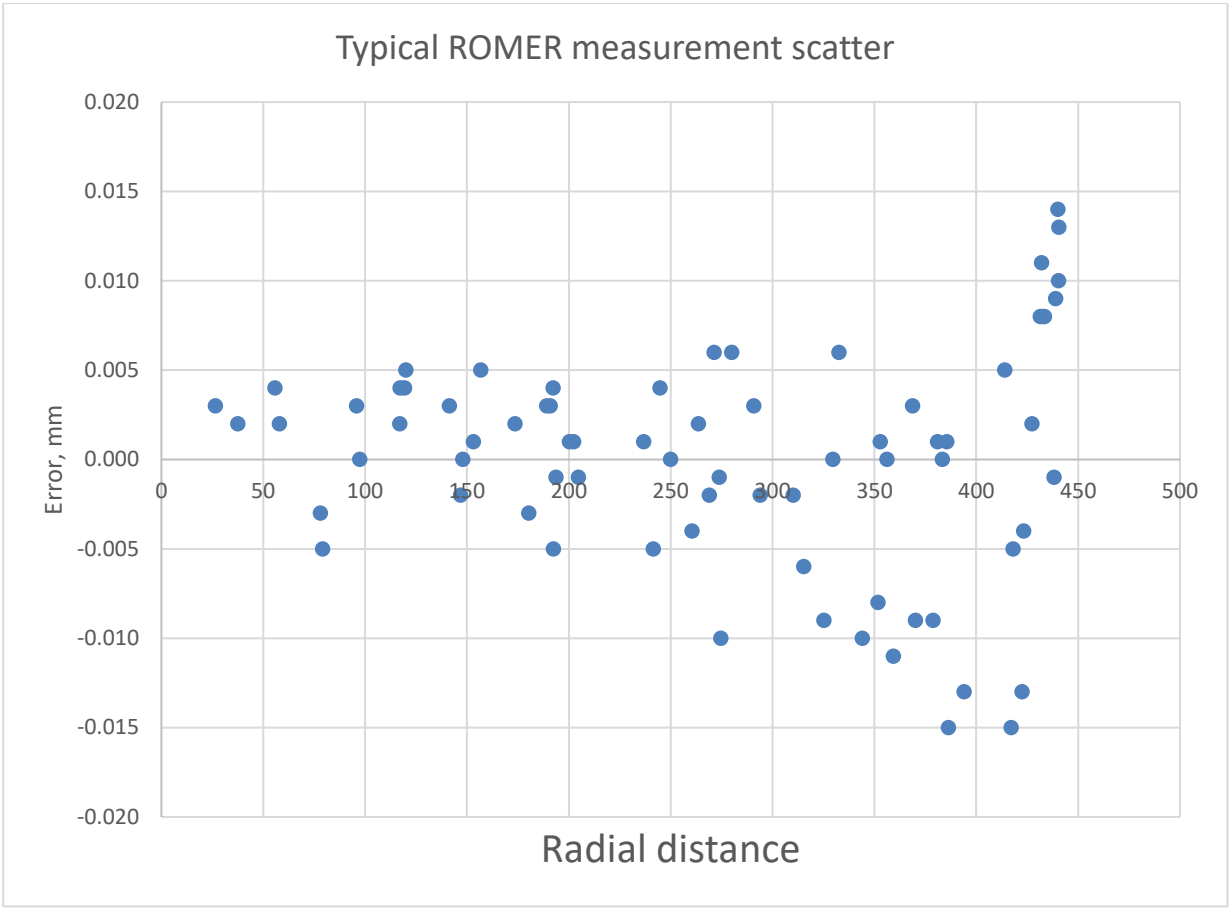
A second measurement cloud produced a very similar scatter.

The RMS scatter in both plots was around 0.0065 mm.

Putting aside the magnitude of the scatter for the moment it is quite clear that there is very little systematic error, which validates the decision to exclude the conic constant and aspheric coefficients from the optimisation.

Now if we consider the "outer reaches" of the surface, so to speak, it is clear that this ROMER ARM might be struggling a little at the limits of its measurement range.

Notwithstanding this, the morphological similarity of 2 complete sets of measurements gives a pretty high level of confidence in the calculated vertex RoC.


Typical ROMER measurement scatter
0.020
0.015
0.010
0.005
0.000
-0.005
-0.010
-0.015
-0.020
Error, mm
0
50
100
150
200
250
300
350
400
450
500
Radial distance

```
// Point cloud data structure and functions

TYPE
  TSurfData = class
    Public
      CMax : INTEGER ; // number of data points
      x0, y0, z0, theta_x, theta_y, TErr: DOUBLE ;
      c, sgn_c, cc, a4, a6: DOUBLE; // aspheric coefficients
// reported data points cX, cY, cZ, and calculated variables
      cX, cY, cZ, h, vZ, dvZ: ARRAY[1..1000] OF DOUBLE ;
// Romer reference sphere CoC in global system
      x_c, y_c, z_c: DOUBLE;
// reference sphere & measurement ball curv/radii
      c_r, r_r, R_b: DOUBLE;
    FUNCTION CalcSurf(): DOUBLE;
    PROCEDURE Load3D_PCData(FileName: STRING);
    Private
  end;
//=======================
FUNCTION TSurfData.CalcSurf(): DOUBLE;

VAR
  J: INTEGER;
// intermediates used frequently
  sin_theta_x, cos_theta_x, sin_theta_y, cos_theta_y, y_int: DOUBLE;
// Romer reported points after translation [only] into local system
  x_s, y_s, z_s: DOUBLE;
// intermediates used more than once
  denom, h_2, h_4: DOUBLE;
// reference sphere normal components from reported coordinates
  n_s_i, n_s_j, n_s_k: DOUBLE;
// fully transformed reported point and normal to Romer sphere
  s_s, n_s: Vector;
// normal to asphere, correction vector, point on asphere
  n_a, d_a, s_a: Vector;
// successive values of asphere parameter "h" or "u"
  h_old, h_new: DOUBLE;

BEGIN
// intermediates used more than once
  sin_theta_x := SIN(theta_x);
  cos_theta_x := COS(theta_x);
  sin_theta_y := SIN(theta_y);
  cos_theta_y := COS(theta_y);

  sgn_c := Sign(c);
// initialize the figure of merit
  TErr := 0;

  for J := 1 to CMax do begin
// intermediates
    x_s := cX[J];
    y_s := cY[J];
    z_s := cZ[J];
// reference sphere normal points towards CoC of reference sphere
    n_s_i := x_c - x_s;
    n_s_j := y_c - y_s;
    n_s_k := z_c - z_s;
```

```
// translate
    x_s := x_s - x0;
    y_s := y_s - y0;
    z_s := z_s - z0;

// rotate reported point and normal into local system

{ Comment: algortihm is...
| x' |   | cos_theta_y   0  -sin_theta_y ||  1          0             0   || x |
| y' | = |      0        1        0      ||  0   cos_theta_x   sin_theta_x|| y |
| z' |   | sin_theta_y   0   cos_theta_y ||  0  -sin_theta_x   cos_theta_x|| z |
  End comment}

// transform coordinates

// intermediate used more than once
    y_int := -y_s*sin_theta_x + z_s*cos_theta_x;

    s_s[1] := x_s*cos_theta_y - y_int*sin_theta_y;
    s_s[2] := y_s*cos_theta_x + z_s*sin_theta_x;
    s_s[3] := x_s*sin_theta_y + y_int*cos_theta_y;

// transform normal

// intermediate used more than once
    y_int := -n_s_j*sin_theta_x + n_s_k*cos_theta_x;

    n_s[1] := n_s_i*cos_theta_y - y_int*sin_theta_y;
    n_s[2] := n_s_j*cos_theta_x + n_s_k*sin_theta_x;
    n_s[3] := n_s_i*sin_theta_y + y_int*cos_theta_y;
// Normalize [make unit normal vector]
    NormalizeVector(n_s,n_s);

// calculate asphere normal at s_a = s_s first, then iterate
    s_a := s_s;
    h_new := 0;

    REPEAT
      h_old := h_new;

      h_2 := s_a[1]*s_a[1] + s_a[2]*s_a[2];
      h_new := SQRT(h_2);
      h_4 := h_2*h_2;

      denom := SQRT(1 - (1 + cc)*c*c*h_2);
      n_a[1] := -sgn_c*(c*s_a[1]/denom + 4*a4*h_2*s_a[1] + 6*a6*h_4*s_a[1]);
      n_a[2] := -sgn_c*(c*s_a[2]/denom + 4*a4*h_2*s_a[2] + 6*a6*h_4*s_a[2]);
      n_a[3] := sgn_c;
// Normalize
      NormalizeVector(n_a,n_a);
// Subtract the 2 normal vectors
      VSub(n_s,n_a,d_a);
// Scale by radius of ball
      VMul(R_b,d_a,d_a);
// Add to s_s, new s_a
      VAdd(s_s,d_a,s_a);
    UNTIL (h_new - h_old) <= 1E-6;
```

```
// h[J] for scatter plots
    h[J] := h_new;
// calculate sag from transformed and processed coordinates
    vZ[J] := c*h_2/(1 + SQRT(1 - (1 + cc)*c*c*h_2)) + a4*h_4 + a6*h_2*h_4;
// calculate difference between measured and calculated sag
    dvZ[J] := s_a[3] - vZ[J];
// this quantity squared and summed
    TErr := TErr + SQR(dvZ[J]);
  end;
// this quantity (RMS) minimised by the optimiser
  CalcSurf := SQRT(TErr/CMax);
END ;
```