Foucault Pendulum Electronics Kit.

D09_Processing the Position Data.

www.foucaultpendulum.nl

Document version	2026-05-25
Related Documents	Source code GUI_Pendulum (Lazarus project)

In brief:
This document describes the steps in the processing of the Position Data from the PMS to the Ellipse Parameters Omega, Semi-Major axis, Semi-Minor axis, and Rotational Direction of the elliptical path.

History:
Interpreting the data from a Foucault Pendulum involves at first a symmetry problem. The pendulum generally follows an elliptical path and there is no way to distinguish a 180 degrees rotated ellipse from its original. The same way the first half of the precession rotation cannot be distinguished from the second half.
The method I used to overcome this problem is to label just one half swing as number 1 and then count 1-2-1-2-1-2 at the center crossings of the bob and not loosing count. Computers can do that quite well.

In the “old” dataprocessing I searched in the first HalfSwing for the datapoint the farest away from the center and marked that as the Precession Angle and the length of the semimajor axis. The length of the semiminor axis and the rotational direction of the ellipse were given by the datapoint at the time the CenterPass was detected. This method works, but actually uses only 2 of the 40 available datapoints, and so gave quite noisy results.

Early 2024 an important step was set to improve this, by introducing the SFFT method (later more) but my knowledge of math was insufficient to derive the ellipse parameters from the SFFT results. Recently I was able to find this missing link with some help of Google-AI and a special program written to test these algoritms.

What is the SFFT method? I call it this way because it can be seen as a Single Frequency Fourier Transform. (not to confuse with FFT, the Fast Fourier Transform, which is a smart way of calculating the Fourier Transform for many datapoints, e.g. spectrum analysis)
Other naming could be the Quadrature Method or the method with Harmonic Components.
It works this way:
The Position Measuring System (PMS) produces position data at a 10 Hz rate. The period time of my 4 meter pendulum is just over 4 seconds, so we have approximately 41 datapoints [X, Y, t]
The time t at which the position was measured is derived from the sample clock of 20 kHz which gouverns the measurements.
For a whole period T (in 20 kHz ticks) of the pendulum we calculate:

```
SinTheta = sin (t / T * 2 * pi);
CosTheta = cos (t / T * 2 * pi);
SumSinX += X * SinTheta;
SumCosX += X * CosTheta;
SumSinY += Y * SinTheta;
SumCosY += Y * CosTheta;
```

After the whole period T we divide the sums by the number of datapoints we had (this can vary +/- 1) and then we calculate:

```
AmplitudeX = sqrt (sqr(SumSinX) + sqr(SumCosX));
AmplitudeY = sqrt (sqr(SumSinY) + sqr(SumCosY));
PhiX = arctan2 (SumCosX, SumSinX);
PhiY = arctan2 (SumCosY, SumSinY);
```

In early 2024 I had already proven that I could reproduce the ellipse by X-Y plotting sinewaves with these amplitudes and phase. (the phase difference was sufficient) But the step to derive the ellipse parameters from these data was beyond my knowledge of math at that time.

Early 2026 I have found the following with some help of Google in AI mode:

```
{Pascal}
procedure CalculateEllipseParameters (AX, AY, PhiX, PhiY: double;
                                      var Major, Minor, Omega: double);
// When we X-Y plot two sinewaves with amplitudes Ax and AY
// and Phase PhiX and PhiY we get an ellipse.
// This procedure returns the length of the semimajor and semiminor axes
// and the tilt angle Omega of the ellipse.
// We assume the ellipse being centered around [X,Y] = [0,0].
var
  TwoPi, DeltaPhi, Discriminant, Numerator, Denominator: real;
begin
  TwoPi:= 2 * pi;
  DeltaPhi:= PhiX - PhiY;
  // Calculate the half lengths of the axes (invariant to the rotation)
  // Use the property: a^2 + b^2 = AX^2 + AY^2
  // and the discriminant of the quadratic form.
  Discriminant:= Sqrt (sqr (sqr (AX) - sqr AY)) +
                 sqr (2 * AX * AY * Cos (deltaPhi)));
  Major:= Sqrt (0.5 * (sqr (AX) + sqr (AY) + Discriminant));
  Minor:= Sqrt (0.5 * (sqr (AX) + sqr (AY) - Discriminant));

  // 2. Calculate the angle Omega
  Denominator:= sqr (AX) - sqr (AY);
  Numerator:=  2 * AX * AY * Cos (DeltaPhi);

  // ArcTan2 gives a result for 2 * Omega between -Pi en +Pi
  Omega := 0.5 * ArcTan2 (Numerator, Denominator);

  // 3. Correction to full circle (2*Pi)
  // Because Omega is between -Pi/2 en +Pi/2 after division by 2

  // 4. Normalize to 0..2 * Pi
  if Omega < 0 then Omega += TwoPi;
  if Omega >= TwoPi then Omega -= TwoPi;
end;
```

This procedure gives me the correct lengths of the semimajor an semiminor axes. The tilt angle however is not correct over the full TwoPi. Understandable, because of the symmetry problem mentioned above.

To overcome this problem I use these steps

- Find the farest datapoint after the first CenterPass, like in the “old” method. Calculate the approximate PrecessionAngle CoarseOmega. This works fine over the full TwoPi.
- Rotate all datapoints over the angle - CoarseOmega + pi/4. Now the algorithm above always gives a correct result for the tilt angle, it is always near +45 degrees.
- Add the CoarseOmega - pi/4 to the found tilt angle, and we have the correct PrecessionAngle. This works over the full precession circle.

In the code we use the CoarseOmega from the previous full swing because we do not like to temporally store a lot of datapoints.

The Essential Steps.

Given:
- The PMS produces 10 datapoints [N, S, E, W, t] per second. For a 4 second pendulum that is ca. 40 datapoints. The NSEW signals are 10 bit unsigned integers, with an inverse relation with the wire-detector distance. t is in the 20 kHz ticks of the detector clock.
- The Bob Synchronisation mechanism produces a boolean FirstHalf which switches at every CenterPass, a boolean LastPoint which identifies the the last datapoint of the second half, and the Total Period Time in ticks of the 20 kHz detector clock.

At each message arrival we do

Step 1: Normalize.

```
PosNS:= (Adc_North – Adc_South) / (Adc_North + Adc_South);
PosEW:= (Adc_East – Adc_West) / (Adc_East + Adc_West);
```

When the gains and offsets of the detection channels are well adjusted this yields position info with little non-linearity, although the raw Adc signals have a 1/distance relation.

Step 2: Rotate the position data to correct for a misaligned Top Unit.
It is nice to have the real North on top of the Compass display.
We use:

```
procedure Rotate (X, Y, Angle : double; var Xrotated, Yrotated: double);
var CosAngle, SinAngle: double;
begin
  CosAngle:= cos (Angle);
  SinAngle:= sin (Angle);
  Xrotated:= X * CosAngle – Y * SinAngle;
  Yrotated:= X * SinAngle + Y * CosAngle;
end;
```

Step 3: Finding the farest.
Starting at the CenterPass “FirstHalf” we track the distance from the center as given by Pythagoras:

```
DistanceSquared:= sqr (PosEW) + sqr (PosNS);
```

until we have found the farest away from the center. Then we apply:

```
CourseOmega:= arctan2 (PosNS, PosEW);
```

to have a rough estimate of the PrecessionAngle. Note that this works over the full 360 degrees of the precession rotation.

Step 4: Rotate the datapoints using minus the CourseAngle found in the previous swing.
Using the procedure in step 2.
The datapoints now represent an ellipse laying on a 45 degrees line, with the farest point in the first quadrant.

Step 5: Summing for the SFFT.

```
Theta:= 2 * pi * PositionCounter_PMS / MaxPositionCounts;
SinTheta:= sin (Theta);
CosTheta:= cos (Theta);
SumCosX += PosEW * CosTheta;
SumSinX += PosEW * SinTheta;
SumCosY += PosNS * CosTheta;
SumSinY += PosNS * SinTheta;
```

Here Theta is the momentary angle of the bob w.r.t. is rest position, PositionCounter_PMS is the time in 20 kHz clockticks at which the position data were taken and MaxPositionCounts is the number of clockticks for a full swing.

Step 6: Apply the Shoelace Method.
The Shoelace method is an invention of C.F.Gauss (1777) to approximate the surface area of any closed curve, if you know the coördinates of a sufficient number of points on the circumference. However we are not interrested in the surface area, but this algoritm is sensitive to the order in which the circumferene is walked through.The sign of the result gives us the rotational direction of the elliptical path. The name shoelace method comes from the similarity with a graphical representation of the cross products with the succesive data pairs.

```
//for the Shoelace method we sum the cross products with the previous
//datapoint
if Points = 0 then // save the first datapoint, we 'll need it later.
begin
  FirstPosEW:= PosEW;
  FirstPosNS:= PosNS;
end
else
  SumArea += ((PosEW * PrevPosNS) – (PosNS * PrevPosEW));
PrevPOsEW:= PosEW;
PrevPosNS:= PosNS;
Points += 1;
```

When the signal **"LastPoint"** arrives, that is after we have a full swing, we finish the processing and make everything ready for the next swing.

Step 7:
We finish the Shoelace method by connecting the first and last points. This gives us the rotational direction of the ellipse.

```
SumArea += (FirstPosEW * PosNS) – (FirstPosNS * PosEW);
EllipseRotationalDirection:= –sign (SumArea);
MinorAxis *= EllipseRotationalDirection;
```

Step 8: Calculate the SFFT

```
// normalize the SFFT data
SumCosX /= Points;
SumSinX /= Points;
SumCosY /= Points;
SumSinY /= Points;
// calculate the harmonic components
AmplitudeX:= sqrt (sqr (SumSinX) + sqr (SumCosX));
AmplitudeY:= sqrt (sqr (SumSinY) + sqr (SumCosY));
PhiX:= arctan2 (SumCosX, SumSinX);
PhiY:= arctan2 (SumCosY, SumSinY);
```

Step 9: Calculate the Ellipse parameters
From the SFFT data we calculate the semi major and minor axes and the tilt angle of the ellipse using the procedure mentioned above.

```
CalculateEllipseParameters
 (AmplitudeX, AmplitudeY, PhiX, PhiY, Major, Minor, Omega);
```

Step 10: Calibrate the semi minor axis in mm.
Because of our Amplitude Control mechanism we know the length of the semimajor axis.

```
// calibrate the Major and Minor axes
```

```
EllipseMajorAxis_mm:= SetPoint_Amplitude_mm;
EllipseMinorAxis_mm:=
  SetPoint_Amplitude_mm * EllipseMinorAxis / EllipseMajorAxis;
```

Step 11: Rotate Omega to what it should be.

```
Omega += CourseOmega – pi/4;
```

That's all.
Of coarse the practical code as it shows in the unit u_calc.pas contains many more statements to catch possible problems when the data is bad (in particular during startup of the system and when the pendulum is out of sync), to display intermediate results for diagnosis and to plot the scaled data.

The system also has a facility to auto-zero the center of the elliptical path.
During each whole swing the normalized PMS data are summed in X and Y direction. These sums are supposed to be zero, and if not they can be corrected to be so. Even if you decide not to do this autocorrection the correction data are calculated and logged.
At this moment I cannot tell what conclusions might be drawn when the correction data show some variance related to the precession angle, or what happens when we do no autozero corrections at all and there is a substantial offset in play.