Foucault Pendulum Electronics Kit.

D08_Description of the GUI Software.

www.foucaultpendulum.nl

Document version	2025-11-01
Related Documents	Source code GUI_Pendulum (Lazarus project) D02_Options D03_Functionality GUI_Pendulum D06_Arduino PinUse_Messages D13_Development_Environment & Coding Rules

In brief:
The GUI_Pendulum program (short: the GUI) is a PC program which monitors and displays the pendulum movements, allows the operator to set many parameters and logs the behaviour of the pendulum and the equipment.
This document gives short descriptions of the files in the project and explains several calculations which are done to present the data in a clear way.

The program is written in FreePascal with the Lazarus IDE. and communicates with the hardware (PCB BobControl en associated electronics) by Ethernet via the local area network (LAN)
The program is published only in the (complete) source code. The user needs to install the development environment (Lazarus IDE) on his/her own PC.
In all cases the user will need to make changes to the program to make it fit for the pendulum in case.

The program source code consists of a number of files, in the project directory. downloadable as a .zip file.

GUI_Pendulum.pas: The startup program. This is a very short piece of code which rarely needs attention.

Most of the work is done in the following 5 units and therefore they are prone to need changes:

u_main.pas
- Contains all controls on the main form, as much as possible grouped by the *GroupBoxes*.
- Procedures for reading and writing the the settings to/from the Parameter file.
- Has the procedure FormCreate where the other forms are initialized and the *GroupBoxes* get their sizes and positions.

If you need to add / remove / change controls it should be done here.

u_messages.pas
- Contains the message preparation, exchange and decoding for the Ethernet communication with the electronics.
- Here most of the values are put into the textboxes and status indicators are set.
- During processing the *InMessage* the procedure *ProcessPositionData* in **u_calc.pas** is called.

u_calc.pas
- Contains the more sophisticated calculations with the data from the Postion measuring System (PMS) like finding the Precession Angle and the lengths of the major and minor ellipse axis. From here also the Compass display is maintained.

More exhoustive description below.

u_logging.pas
- Has the routines for Data Logging and generating new logfilenames.
If you need to change the logging format, here is where to do it.

u_timedisplay.pas
- Maintains the graphs on the TimeDisplay.
The data are stored in *traces* from where they are displayed. Data points enter on the right side of the graph and shift to the left on each entry. We have a “fast” and a “slow” set of traces. The “fast” set is updated at each HalfSwing of the pendulum, the “slow” set goes in 15 minutes, 1 hour or 5 hours per division.

And we have a number of supporting units which will rarely or never need to be changed.

u_scope.pas
- Basic functions for the TimeDisplay.

u_compass.pas
- Basic functions for the CompassDisplay.

u_numstrings.pas
- Some numeric to / from string conversions.

u_timestrings.pas
- Some time functions, a.o. for filename generation.

and the files from the **synapse library** for the ethernet communication: **blcksock.pas** and names starting with **syn******.pas.**

There are two other forms in the project:

u_large_adc.pas
- Shows a form with large textboxes showing the raw PMS data.
This form is helpful during the gain adjustment of the PMS channels because it can be read from quite a distance.
It is activated by the checkbox *Show Large Form* in the PMS pane.

u_corrections.pas
- Shows a form with some textboxes to enter certain correction data for future use.
Background: I’ve found that the PMS suffers from a small amount of non-lineairity and cross talk between the NSEW channels, resulting in some pillow shaped distorsion. I’ve planned to develop correction algorithms for this.
It is activated by the checkbox *Show Corrections* in the PMS pane, but currently does nothing.

There are no special library files to be installed in the Lazarus / Freepascal IDE.

Files with the extension **.lfm, lpi** and **.res** are maintained automatically and never need attention.
The file with extension **.lpi** is the project startup file. It should have a file association set such that double clicking on it from within a file manager will start the *Lazarus IDE.*

Some more words about the unit u_calc,

After the contents of the *InMessage* buffer have been decoded in **u_messages** the prodedure *ProcessPositionData* in **u_calc** is called for each message arrival. (10 Hz rate).

We start with identifying and separating the *CenterPasses* in the first and the second *HalfSwing* of the pendulum. This is important to distinguish between the first and second 180 degrees of the Precession circle.

The switch *CalibratePMS* allows adjusting the gains of the 4 PMS channels to the same value.
Normally we continue to normalize the PMs results as:
PosNS = (North - South) / (North + South) giving a value -1 > pos > +1, where some first order non-linearities are reduced.

We then calculate the Automatic Zero Correction by first summing the data during a whole swing and then averaging the result with a slow averager.
Depending on the checkbox *Auto Zero* the correction is applied to the data for the further processing. (Later on, we apply the same correction to the PMS data frozen at the moment of the CenterPasses)

The procedure and purpose of the *QuadratureData* is descibed at the end of this document.

We continue with plotting the blips on the CompassDisplay in the color specified for the first or second *HalfSwing*.
We save the position of each blip to be able to remove it when *Persistence* is switched off.

Further processing is done only for the first “blue” HalfSwing.
The aim is here to find the datapoint the farest away from the center. This tells us the *PrecessionAngle.*
As soon as we see the Center1Pass, the start of the “blue” halfswing, we first normalize the PMS data from that CenterPass and apply the AutoZero correction.
Zero correction is important here. Without it a small offset would introduce a substantial error in the measurement of the minor ellipse axis length, particularly when there is little ellipse.
When enabled we plot a thick green dot to visualize this position.

Now we start searching for the farest datapoint. We use the sum of squares as the absolute distance (no need for the square root now) and track the distance. If *foundfarest* we plot that position as a red dot (when enabled) and after slow averaging a thick blue dot to visulalize that position (when enabled).

On the next Center1Pass we calculate the length of the minor ellipse axis, as the distance (pythagoras) from the Center (assumed [0,0] because of the *AutoZero* correction) and the PMS data captured at the CenterPass. Again in normalized PMS units.

Rest to find the rotational direction of the ellipse.
For this we rotate the positiondata from the CenterPass over *minus* the found *PrecessionAngle*. (If we had rotated all datapoints we would see the ellips laying with the major axis along the horizontal. But we need only the rotated CenterPass data)
If the Y-position of the rotated data point is negative we know that the rotation was CCW, or in the positive direction. if negative it is CW.

We now know the *PrecessionAngle* and the length of the *major ellipse axis* in the normalized PMS units, after taking the square root.
We plot a red circle on the edge of the CompassDisplay for the *PrecessionAngle.*

Now we can calibrate the ellipse axis in mm.
In case we use the Automatic Amplitude Control we know that the major axis has the length given as the SetPoint_mm. So we can calculate a calibration factor *EllipseCalibrationFactor_Auto* and use that for the length of the minor axis.
When no Automatic Amplitude Control is used we must supply a calibration factor ourselves. This is *CalibrateEllipseAxis_mm* from the textbox Cal_mm in the Amplitude Control pane. We apply this to both the minor an major axis.

At last we prepare some data for logging. We log at each second HalfSwing but need to know some data from the first one too.

About the procedure *CalulateQuadratureData (....);*
Here we consider the ellipse as consisting of two perpendicular sine waves in X and Y, with different amplitudes and phase. The aim is to determine these amplitudes and phase angles using all datapoints given by the PMS for each complete swing.
The expectation is that this will give much more accurate and noise free ellipse data.

In each message we get the PMS data along with the value of the *PositionCounter* at the moment of capture. This lets us relate each position to the time it was captured. We also know the total *PeriodTime*, as being the sum of two successive *CenterPass Times.*

We calculate a sine and a cosine function for each datapoint and sum the products.
We also count the number of datapoints we sum. After each whole swing we can now determine the amplitudes and phases of the X and Y components of the ellipse.

From these data I have been able to perfectly (*) reproduce the pendulum movements by constructing ellipses from X-Y plotted sinewaves with the given X-Y amplitude and phase difference.
Unfortunately here ends my knowledge of math and i have not yet been able to directly extract the ellipse parameters (major, minor axis and angle of the major axis) from these data. I do know that the rotational direction of the ellipse has been lost in this process, but for that I could use the metod described above.

(*) assuming that we deal with perfect sine waves, which is ‘nt exactly so.

Diagnostic Features.
Many textboxes and status indicators and the data in the logfiles give us information, not only about what the pendulum is doing but also about the correctness of the data processing. A good knowledge of the underlaying processes is required to judge the reliability of the data.