Foucault Pendulum Electronics Kit.

D07_Description_Firmware

www.foucaultpendulum.nl

Document version	2025-11_02
Related Documents	Firmware Source Code D02_Options. D05_Description Electronic Circuits. D06_Arduino_PinUse_Messages. D13_Development_Environment, Coding Rules ATMega2560_Manual

In brief:
This document describes the processes in the firmware which detect the Center- and Rim Passes, generate the DrivePulses and does the Automatic Amplitude Control.
On the end some words about the averaging algorithm used at several places.

General.
The system has 4 methods to synchronize with the Bob.
This can be selected in the panel Drive Parameters on the GUI.

CenterPass_Magnetic: The timing of the Drive Pulses is synchronized with CenterPasses detected by a separate CenterPass Detection coil or by the signal coming from the DriveCoil itself, when the bob flies over it.

CenterPasses Capacitive: The timing of the Drive Pulses is synchronized with CenterPasses detected by an electrode in the center below the bob, using the 465 kHz signal on the wire and bob.

Charron Ring: The electronics is prepared to detect the wire touching a Charron Ring, but no further implementation is done.

Resonance: The electronics and software can deliver DrivePulses at a very stable frequency which can be adjusted in very small steps. This is for future experiments where I will try to drive the pendulum with the frequency of the major ellipse axis and try not to excite the slightly higher frequency of the minor axis. This to limit ellipse growth.

In the firmware we have three detectors: for CenterPasses_Magnetic, CenterPasses_Capacitive and RimPasses_Magnetic. The electronic signal for Detecting RimPasses_Capacitive is implemented in the electronics, but in the firmware not further than the A/D conversion.

The three detectors function simultaneously, but quite independent of each other.
Each detector has its own time base: a *PositionCounter*. The PositionCounters for the CenterPass detectors are set to zero when their CenterPass is detected.
The PositionCounter for the RimPass detector is zeroed by the CenterPass which is selected for synchronizing the Drive Pulses.

There is a fourth PositionCounter which governs the timing of the DrivePulses and the associated gating of the magnetic CenterPass signal coming from the drive coil itself.
This *PositionCounter_Drive* is zeroed at the CenterPass selected for synchronisation by the variable *Drive_SyncMode.*

StateMachine DetectCenter_Cap.

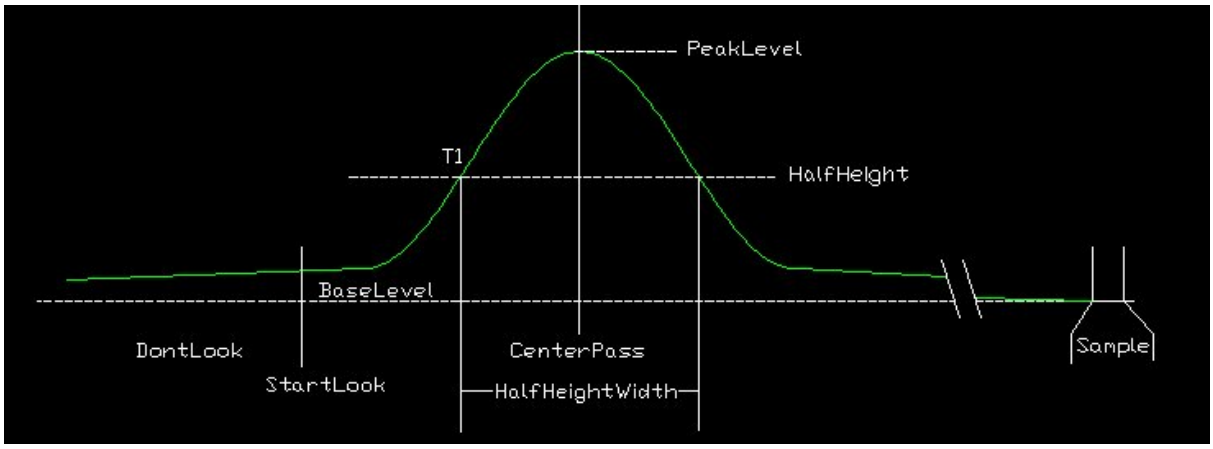

Fig. 1 The signal from the Center electrode and the steps of the detection process.

The 465 kHz signal (green) from the Center electrode is amplified, rectified and fed to the analog input channel *Adc_Center_Cap*. If the bob is far away there is a small signal called BaseLevel.
When the *PositionCounter_Cap* reaches the value *TStartLookForCenter_Cap* we start tracking the rising signal. When the value reaches the *HalfHeight* level at T1 we start a special *PulseWidthCounter* and continue tracking the rising signal until it starts falling. That is the moment of CenterPass detection, and we freeze the data from the PMS as *Center_North,S,E,W*, the PeakLevel as *APeakCenter_Cap* and the *PositionCounter_Cap* as *TPassCenter_Cap*, and zero the *PositionCounter_Cap*.
If the *SyncMode == SyncByCenter_Cap* we also zero the *PositionCounter_Drive* for the timing of the DrivePulse and *PositionCounter_Rim* for the RimPass detector.
From here on we track the falling signal until it becomes lower than the *HalfHeight* level. There we freeze the value of the *PulseWidthCounter* as *HalfHeightWidth* of the pulse.
We wait until the bob is near the maximum amplitude, when the *PositionCounter _Cap* reaches *TQuarterSwing*, which was calculated at the CenterPass.
Here we take some 20 samples and average them as *ABaseCenter_Cap.*
We also average the APeak Center_Cap and recalculate the HalfHeight level as halfway the avaraged PeakLevel and BaseLevel.

Outside the statemachine is constantly checked if
PositionCounter_Cap > TMissedCenter_Cap. In that case we decide that the CenterPass has been missed and restart a try to find synchronisation

Also when the signal *ForceResync* is given from the GUI the statemachine is kicked to the *Idle* state to try for a new synchronisation.

The background for the measurement of the half-height width of the capacitively detected center pulse is that it is a measure for the velocity of the bob and so for the amplitude of the pendulum. (A correction may be needed if there is ellipticity).

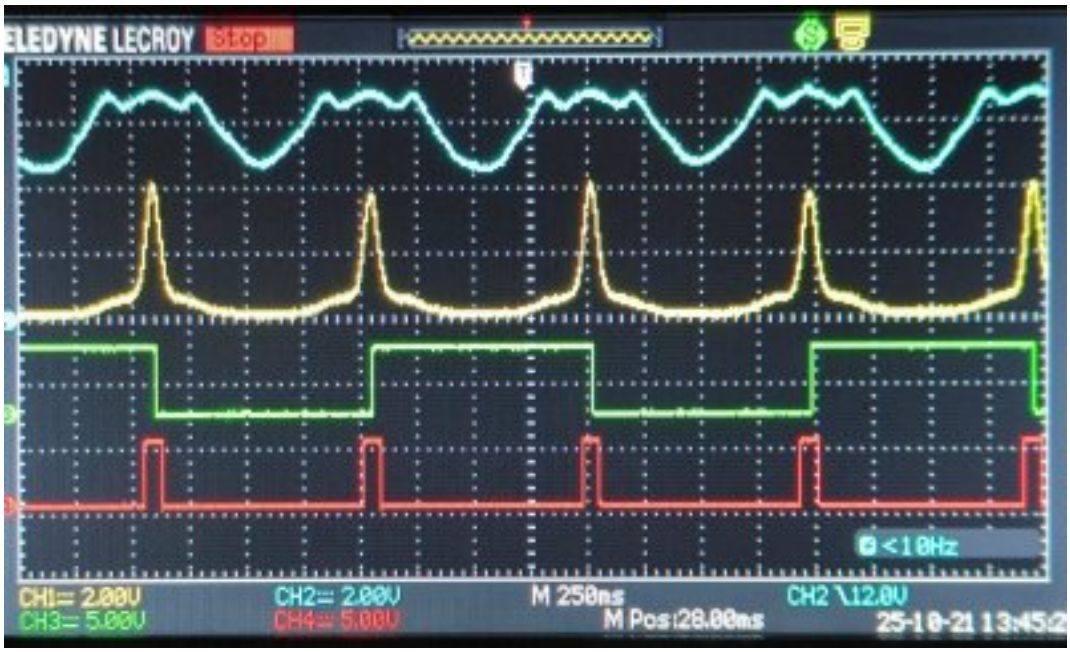

Fig 2. The signals from the Capacitive detection.

Blue: Rim Pass signal Capacitive at analog input A6. It is clearly to see that detecting the RimPasses requires a rather complex algorithm. For that reason I did not yet implement that detection. The shape of this signal might be much more usable when the bob is shaped more like a thin cylinder.
Yellow: The CenterPass signal at analog input A4.
Green: The diagnostic signal at pin A12 toggles at the detection of CenterPasses.
Red: The diagnostic signal at pin A11 shows the timing of the HalfHeight transitions for the HalfHeight Width determination. The HHWidth is an indication for the velocity of the pendulum and therefore for the Amplitude.

These signals are from my sub-meter pendulum with a period time of ca. 1.8 seconds. The pendulum had a ring shaped electrode below the bob in an attempt to detect rim passes and use them for amplitude control.

StateMachine DetectCenter_Mag.

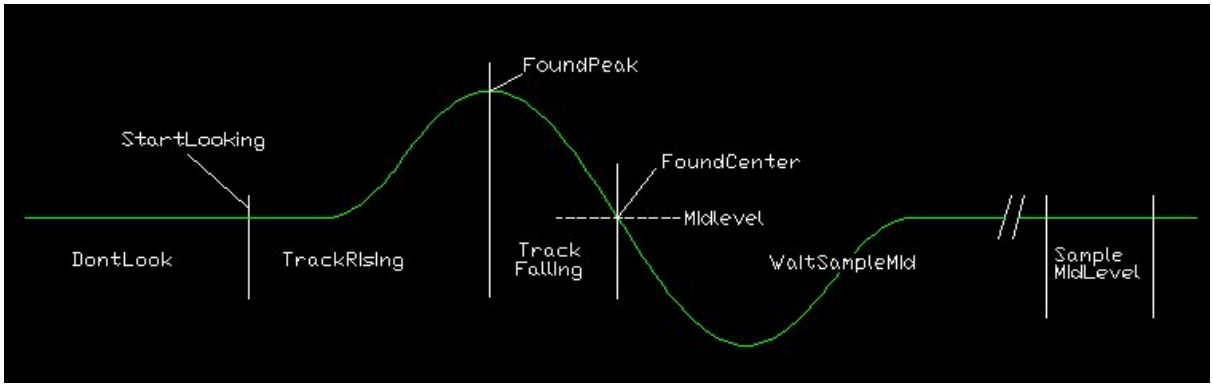

Fig 3. The signal from the CenterCoil and the steps in the detection process.

The signal (green) from the CenterCoil is amplified and lifted to aproximately half the range of the A/D converter (0..5V, 10 bit resolution, so 0..1023 in value. Halfscale is 512)

We start in the state *Idle* where some preparations are made and we jump to the state *DontLook* and wait until the *PositionCounter_Mag* has reached the value of the parameter *TStartLookForCenter_Mag.* Then some preparations are done and we jump to the next state *TrackRising* where we track the rising signal until it starts falling. We detect falling as 5 units below the preceding value to prevent false triggers from noise. Here we decide that we are at the peak of the signal and freeze that value as *APeakCenter_Mag*.
We jump to the state *WaitCenter* where we wait until the signal crosses *AMidCenter_Mag*. *AMidCenter_Mag* is initially set to 512, but it is augmented later on with a more precise value.
The *AMidCenter_Mag* crossing is the CenterPass event. Here we freeze the data from the PMS as *Center_North,S,E,W*, we copy the value of the *PositionCounter_Mag* as *TPassCenter_Mag* and zero the *PositionCounter_Mag* for the next halfswing.
If the SynchronisationMode is set to *SyncByCenter_Mag* we also zero the PositonCounter for the DrivPulse generation.
Also the PositionCounter_Rim is zeroed if configured that way.
From here on we wait until the *PositionCounter_Mag* reaches *TQuarterSwing*, which was calculated at CenterPass as one half of *TPassCenter_Mag* , that is when the bob is at its maximum distance from the center. Here we take a small number of samples and average them as the new *AMidCenter_Mag*. Then we return to the state *DontLook* and wait for the next HalfSwing.

Outside the statemachine is constantly checked if
PositionCounter_Mag > TMissedCenter_Mag. In that case we decide that the CenterPass has been missed and restart a try to find synchronisation.

Also when the signal *ForceResync* is given from the GUI the statemachine is kicked to the *Idle* state to try for a new synchronisation.

StateMachine DetectRim_Mag.

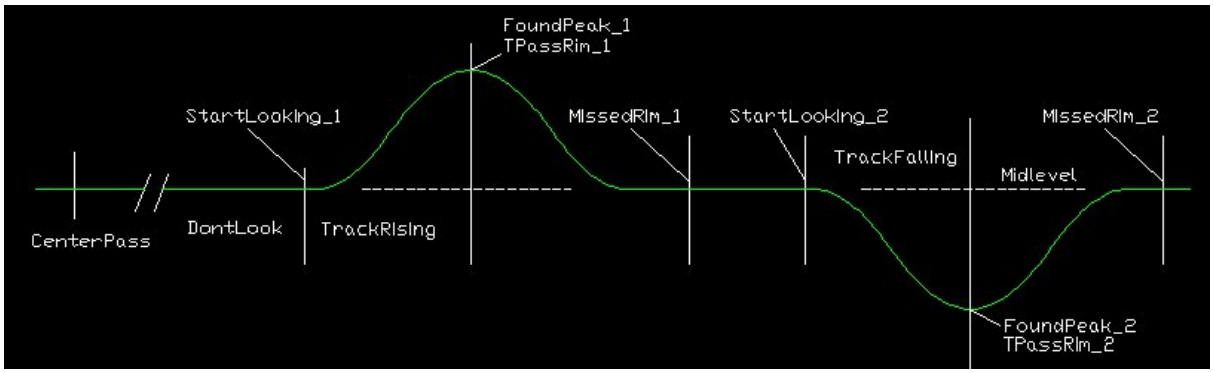

Fig. 4. The signal from the RimCoil and the steps of the detection process.

The signal (green) from the RimCoil is amplified and lifted to aproximately half the range of the A/D converter (0..5V, 10 bit resolution, so 0..1023 in value. Halfscale is 512)

Starting at the left at the time of a CenterPass we expect a positive going signal when the bob passes the RimCoil outwards and a negative signal when the bob goes back inwards to the center.
During some time after a CenterPass we ignore the signal as it may be distorted by the DrivePulse. From *TStartLookForRim1_Mag* we track the rising signal until it starts falling. At that moment we freeze the value of the *PositionCounter* as *TPassRim1_Mag*, and the value of the Rim signal as *APeakRim1_Mag*. We detect falling as 5 units below the prceding value to prevent false triggers from noise.

When the falling of the signal has not been detected when the *PositionCounter_Rim* reaches *TMissedRim1_Mag* we decide that the Rim1-pass is missed. This information is only reported to the GUI and has no further effect on the working of the system.
To detect *TPassRim2_Mag*, the time of the return passage, we follow the same procedure, but with the inverse signal polarity.

Shortly after a new CenterPass, that is when the bob is far away from the RimCoil we take a small number of samples and average them as the new *AMidRim_Mag*.
AMidRim_Mag is used to calculate the absolute amplitude of the APeakRim signals.
If the absolute amplitudes of the Rim1 and Rim2 signals differ substantially something very strange is going on.

In the GUI the amplitudes from successive HalfSwings are subtracted as *ADiff_Rim1_Mag*.
If that difference is not very small it tels us that the RimCoil is not laying perfectly horizontal.
Also the difference in TPassRim1_Mag is calculated. This difference tells us about how well the RimCoil is centered.

Fig 5. Magnetic Signals on the oscilloscope.
Yellow: CenterCoil signal on the Arduino Input A4
Green: Diagnose signal on pin A14 toggles on CenterPasses.
Red: RimCoil signal on the Arduino input A6.
Blue: Diagnose signal on pin A13 goes high at Rim1 pass and low on Rim2 pass.
In both analog signals we see the disturbance from the DrivePulse, shortly after the CenterPass.
These signals are from my 4 meter pendulum with a period time of 4.2 seconds.

DrivePulse Generation.

The Drive Pulse time is generated in the function *ISR (TIMER1_COMPA_vect),* the 20 kHz interrupt handler for Timer 1, which also calls the Center- and Rim Pass detectors.
See the document “DO4_Description of the Electronic Circuits” for the circuit description.
The Drive Current is set by the boolean *MaxDrive*, which is set by the active Amplitude Control mechanism, or forced from the GUI.

The DrivePulse timing is given by the values of *Drive_Start* and *Drive_Stop*, which in the GUI are calculated from the parameters *Drive_Position* and *Drive_Width*.

The option to use the signal from the DriveCoil to detect CenterPasses requires two other signals to be set, just before the DrivePulse begins and a longer time after it stopped. This gating prevents the sensitive pre-amplifier for the CenterPass signal to be overdriven by the DrivePulse.

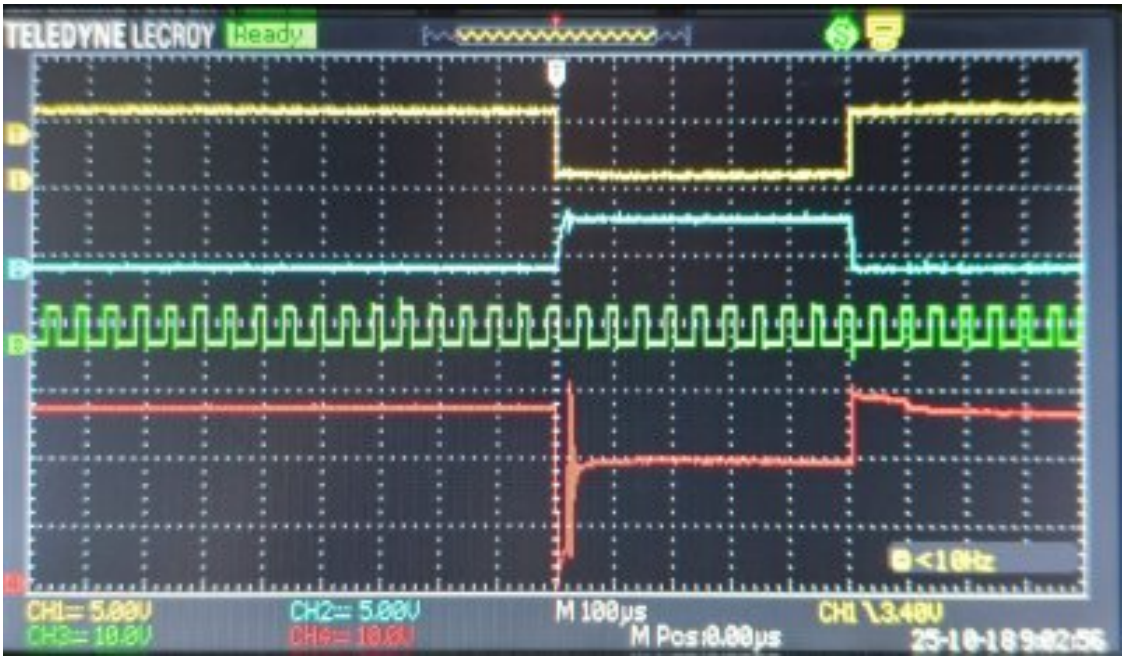

Fig 6. Signals in the DrivePulse circuit.
Yellow: The /DRV signal at Mega pin 41.
Blue: The signal at TP_I_DRV, across the current sensing resistors R32,35
Green : The signal at diagnose pin A15 showing the 20 kHz signal. We can see that the puls lasted 10 ticks, which corresponds to the setting at the time of taking this picture.
Red : The signal at the tab of transistor Q3. We can see that it is going down ca. 9 Volt during the pulse. The overshoot at the start and ending of the pulse are due to the inductance of the coil. This is a normal phenomenon.

Automatic Amplitude Control.
Two methods of keeping the pendulum's amplitude constant are implemented.
Using a Rim Coil.
Here the time from CenterPass (either Capacitive or Magnetic) to the first, outgoing RimPass is used as a measure for the actual amplitude.
The RimPass times are averaged and compared with a precalculated value *TargetAmplitudeControl*. When it takes to long to reach the RimCoil we decide the pendulum's amplitude is to small and we set the boolean *MaxDrive,* which causes the maximum drive strength to be used. Otherwise *MaxDrive* is set to false and the minimum drivestrength is used.
Using the width of the CenterPulse Capacitive.
The Half-Height width of this pulse is determined at each pass, averaged and compared with the precalculated value *TargetAmplitudeControl*. When the witdh is to large we decide the pendulum's amplitude is to small and we set the boolean *MaxDrive,* which causes the maximum drive strength to be used. Otherwise *MaxDrive* is set to false and the minimum drive strength is used.
So in both cases the compare action is the same, only the calculation of the target value differs. The compare action is done in the firmware, the precalculation is done in the GUI.
No Automatic Amplitude Control.

If this option is selected one should set the Drive strength to a fixed value with one of the checkboxes ForceMinimum or ForceMaximum in the Drive Parameter pane.
Changes in the amplitude can stil be seen and logged as changes in TPassRim1_Mag and / or THHWidth, if implemented and enabled.

Leaking Bucket Averager.
On several locations in the firmware and in the GUI an averaging algorithm is used, known as the “Leaking Bucket Averager”. The basic algorithm is:

av = av - av/F + New/F.

In words: to get the new average we subtract a fraction from the old average and add the same fraction of the new sample.

When we deal with integer numbers and we chose the fraction F as 1/n, where n is a power of two then the implementation becomes very effective, because division and multiplying by a power of two involves only bitwise shifting of the number. So we do:

av = av - (av >> n) + (new >>n).

Unfortunately if we do it this simple the least significant n bits would drop off by the shifts. Therefore we think everything shifted n bits to the left and it becomes.

avs = avs - (avs >> n) + (new) ; and av = avs >> n;

In fact we use fixed point aritmetic, where the separating point is n bits from the right. Often a 32 bit long integer is needed for avs to prevent overflow.

Such a filter behaves as a first order low pass filter with a time constant of $Fs / 2^n$ where Fs is the sample frequency.
The -3dB corner frequency of such a filter is $F_{-3dB} = Fs / (2 \pi 2^n)$.

On the PC platform we can just use floating point aritmetic because of the much faster CPU’s there.