Lookup and Interpolate

Nonlinear functions of sensors

Many sensors and other functions show a nonlinear curve. This means that the relation between an input value and the corresponding output value is not linear. Examples: PT100, PTC, NTC, light-detectors, and also diodes. This is only a very small count of nonlinear sensors. Normally the measure result has a fixed relation to the external events. The temperature for example. But in many cases this relationship is not linear, but logarithmic, cubic etc. A PT100 shows a resistance of 100 Ohms at 0degC, at 50degC 124Ohms and at 100degC 143Ohms. The relation between temperature and the resistance value is nonlinear.

There are two ways to calculate the temperature from the resistance:

1. With a proper formula, which mostly is a complex thing, one can calculate the the temperature which corresponds with the resistance.
2. Build a so called LookUp-Table. Insert in steps resistance values und the related temperature values. Access the table with the resistance value as an index. The result is the temperature. If the input value reaches from 100 to 200 there must be 100 value pairs in the table. More difficulty is a large span of the input values (0 to 1023). Then the table must have 1024 entries.

Linearising

The herein used implementation is table based, with paired values (search/result). The LookUp algorithm searches with a known value in the table until either this value is found or this argument fits between two values. The searching is done with a binary search function for best speed results.

If a proper value(s) is found it will be linear interpolated. These method allows short tables, dependent of the required accuracy. If the count of the value pairs is relative high, the linear interpolation results in an acceptable accuracy.

System Functions:

```
function InterPolX(const LookUp  : pointer; x : integer; var y : integer) : boolean;
function InterPolX(const LookUp  : pointer; x : longint; var y : longint) : boolean;
function InterPolX(const LookUp  : pointer; x : float; var y : float) : boolean;

function InterPolY(const LookUp  : pointer; y : integer; var x : integer) : boolean;
function InterPolY(const LookUp  : pointer; y : longint; var x : longint) : boolean;
function InterPolY(const LookUp  : pointer; y : float; var x : float) : boolean;
```

The pointer must point into a table in the ROM. The second argument is the search value. It's type defines the operation (Integer, LongInt or Float) of the functions. The result is placed into the location of the third argument, if the function was succesful.

LookUp Table definition and import

The parameter **const LookUp : pointer** which must be passed to the functions, is a Pointer which must point to a ROM constant table. The table should be defined in this way:

```
const
  IntLookUp   : array[1..(size * 4) + 3] of byte = FileName;
  // size point.x point.y of integer, 3bytes info

  LongLookUp  : array[1..(size * 8) + 3] of byte = FileName;
  // size point.x point.y of longint, 3bytes info

  FloatLookUp : array[1..(size * 8) + 3] of byte = FileName;
  // size point.x point.y of float, 3bytes info
```

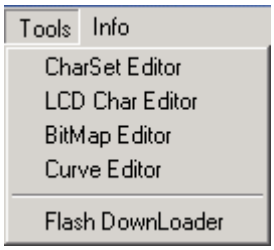
The array definition is only a place holder for the binary file which must be loaded. The parameter **Size** defines the count of the x/y-datapairs and the following multiply factor (4 or 8) is the count of bytes needed for one datapair (sizeOf(integer), sizeOf(LongInt) or sizeOf(float)). The three bytes info contain the used data type and the count of data pairs.

Creating the LookUp Table

As a support tool for the creation of the lookup table a Table Generator **CurveGen** is included. With it's help one can interactively and graphically create a curve and then store it into a binary file which can be imported into the application.

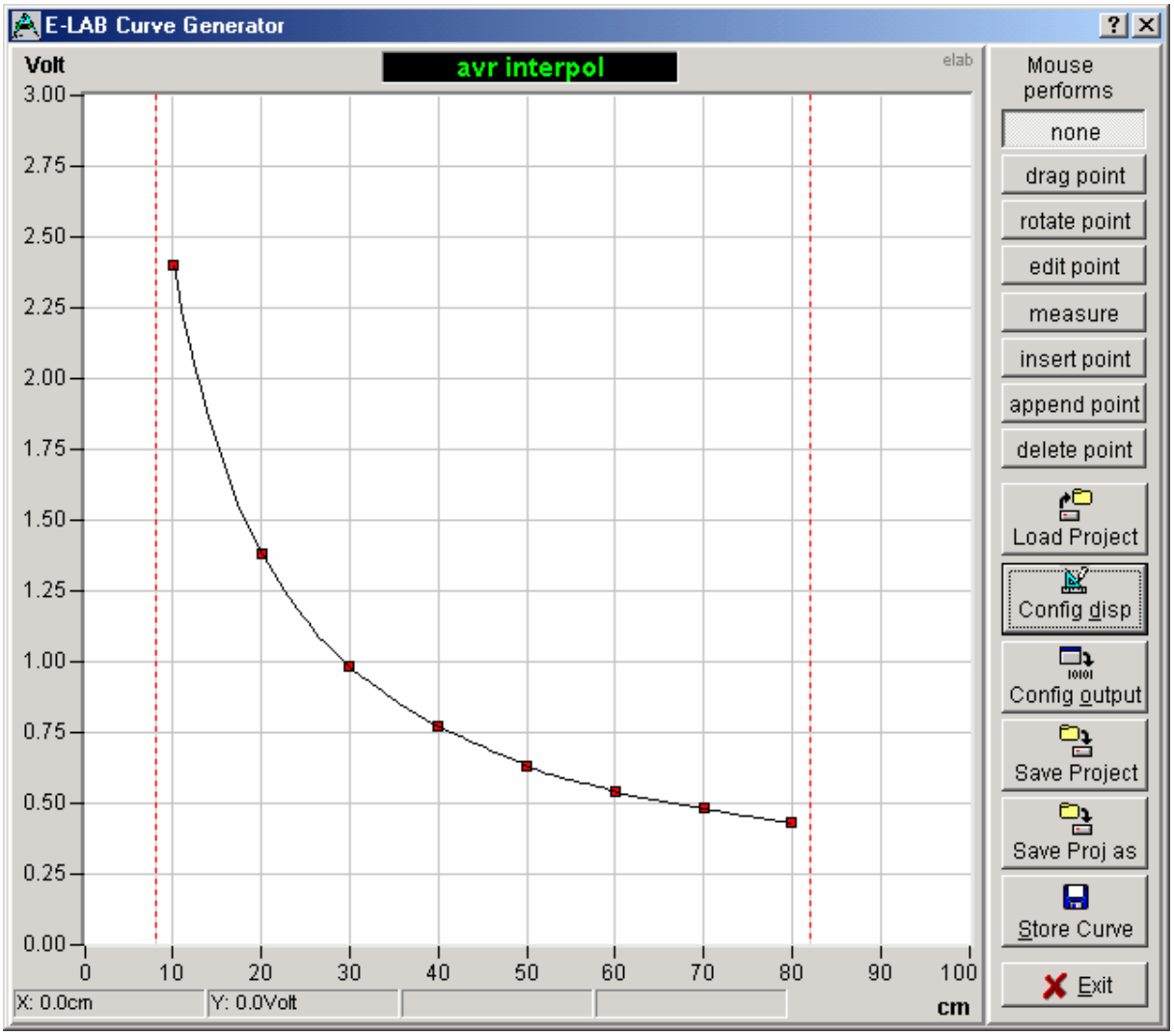
A sample program can be found in the Demos directory as "AVR Interpol". Here an optical distance/proximity sensor (Sharp) is sampled by the ADC and then linearised. The result is displayed in cm. A datasheet can be found in the DOCs directory.

Program start

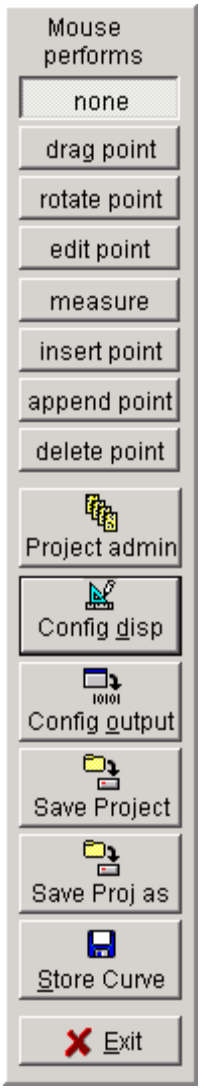


The program *CurveGen.exe* has to be started from the IDE PED32 with the button  or with the menu item *curve Editor* on the left. After setting all the needed parameters and setpoints a binary file can be created which is necessary for the interpolation and lookup in the application.

CurveGen is absolutely interactive and graphic based. Because of this there is always an immediately reaction and view when parameters are changed. The curves between the supplied setpoints are calculated with a special spline-like algorithm. So it's also possible to generate a stairway curve which is impossible with a pure spline.



The above picture shows a screenshot of *CurveGen* with the project **AVR Interpol** loaded.
The two vertical red lines define the start and end point for the table generation.
The red squares show the predefined set points.



The upper half of the picture at left shows the properties which can be choosen for mouse clicks and mouse moves.
None means that the mouse clicks and moves have no affect to the display.
Drag point moves a clicked red set point as long as the mouse key is pressed.

Rotate point allows a click to a point and then manipulate the point's direction (vector) and weight by dragging the white attribute circle.
With **edit point** a click to a set point opens a dialog where the exact position (x/y) of a set point can be edited.
Measure allows to move with the left or right mouse button two cross hair cursors which are used as measuring points. The result is shown at the bottom of the display.
With **Insert point** additional set points can be inserted with mouse clicks between two existing set points.
Append point appends with each mouse click a set point to the last right point.
With **delete point** an existing set point can be deleted.

Project admin opens a dialog described later on below.

Config display opens a dialog for defining the display's parameters like scaling, partition etc. See description below.

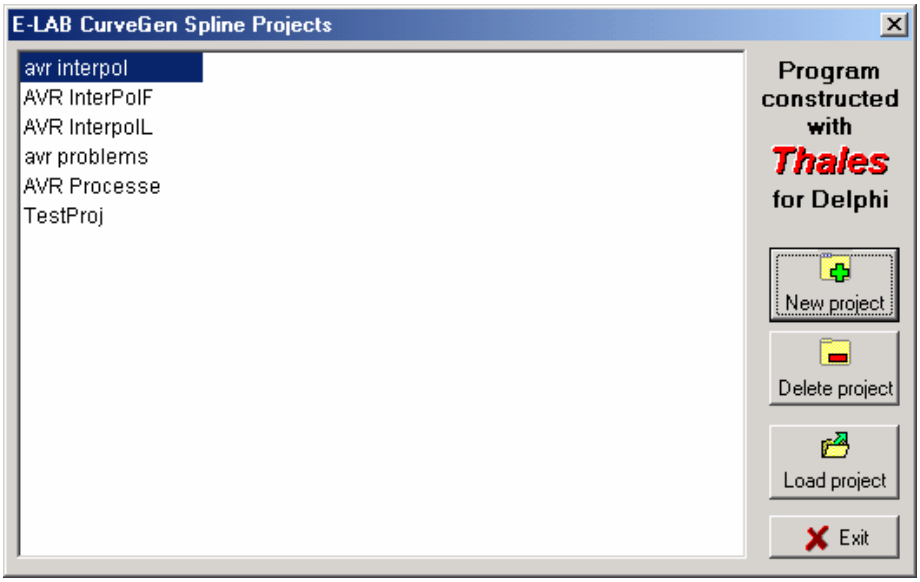
Config output opens a dialog for editing the output/export parameters like data type (integer, longint etc). Description see below.

With **Save Project** all project data and setup is saved in the project file *ProjectName*.inicg

With **Save Proj as** the current project and it's parameters is saved in a new project file.

Store Curve opens the store dialog (see below). A text field shows the generated output in a readable form. So a preview of the generated lookup table is possible.

Exit closes the program and exits.

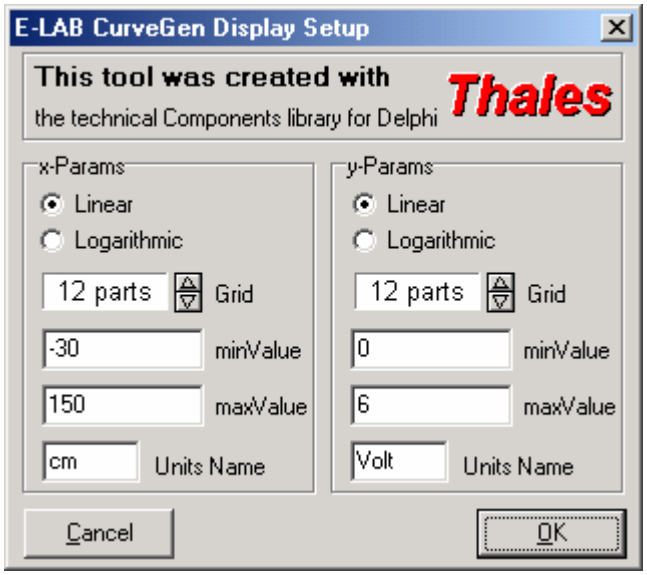


The project admin dialog offers besides the loading of existing projects also project deletion and creation of new projects.

With a double click onto a table entry this project is loaded.

With a click to the **New project** button a dialog appers for editing the new project's name and it's working directory.

This new project now has some default settings and a small count of set points. All parameters and setup values now must be changed to fit to the sensor's behaviour.



The display setup dialog is used to set scalings and partition for the display.

The setting must be done separate for the x and y axis. The scalings can be linear or logarithmic.

The scala partition must be selected so that the text of the scale shows no rounding errors.

The minimal values of the scale are related to the bottom left corner and the maximal values to the upper right corner of the display.



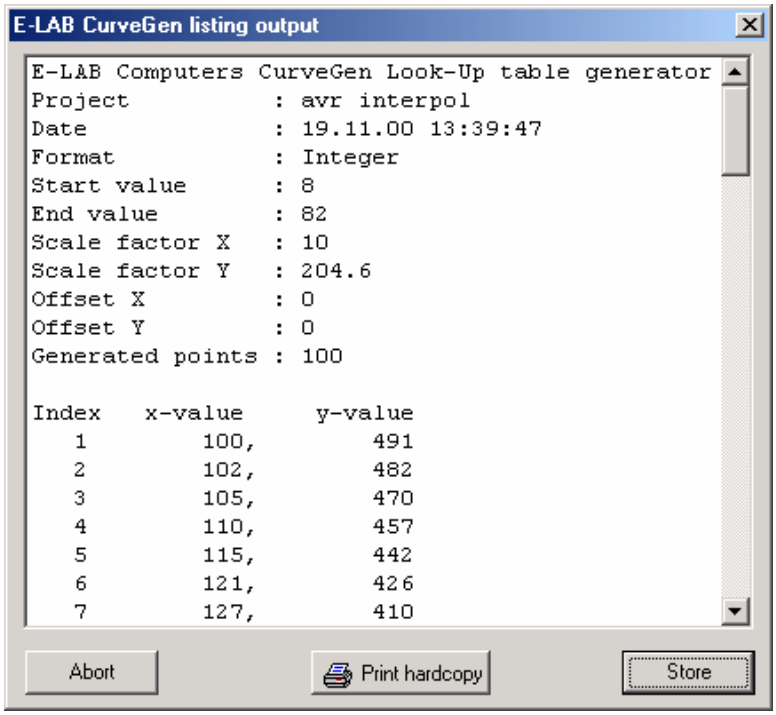
The **Export** dialog on the left serves to edit definitions how the binary file must be build. The data type is defined with **Exported type**

With **Ranges** a part of the display's curve is choosen. This range is displayed by vertical red lines. Start and End Value is only related to the x-axis. Only these points of the curve are exported which reside between these two limiters.

The value **point count** defines value pairs within the selected range should be exported into the binary destination file. It must be even.

The x and y values can be scaled and offsetted for the export. The calculation will be done with the formula:

```
val:= (val – offset) * factor
```



The **Output** dialog shows all relevant settings, parameters and lists the generated lookup value pairs These list can be printed out.

With the **Store** button the binary file *ProjectName*.crvg can be written. This file then must be imported into the application source like described above.

Eigenschaften von CurveGen

Precision:

This program is based on a special spline function. Spline has the property that the generated curve always touches the set points. This results in the consequence that each error in the coordinates of the **setpoint** atleast at this point becomes effective and also somewhat weaker in it's nearness. Because of this the precision of the interpolated values depend heavily on the precision of the set points.

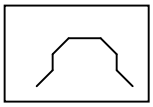
Less important for the precision is the **count** of the set points. With „normal" quadratic, cubic or logarithmic curves a few points in the range of 5..20 are sufficient. The more komplex a curve is the more set points are necessary. An extreme one could be a logarithmic curve with linear parts and stairways.

Curve shape:

By the count and position of the set points the shape of a curve normally is well defined. But a new curve in a new project still has a strong concentration of the point's attributes and directions to the point itsself. This easily can be seen because the parts between the points are absolutely linear.

These behaviour is not what we expect normally. A curve should have a curve shape ☺. Because of this the attributes of the set points must be adjusted in to proper values. This must be done with the mouse and the selection **rotate point**. With a pressed left mouse button the point must be dragged.

But the point itself isn't move but the visible attributes which are shown by a white curcle. With dragging of such a circle the weight and direction (vector) of the point is changed. The longer the distance between the point and it's white circle is the more the curve is pulled ib this direction. This setup also must be done carefully in order to get a curve which is as similar as possible to the origin curve. But don't fear, the human eye is very well suited to adjust this.



Ambiguousity: Curves must be unambiguous from the point of view of the used LookUp parameter. In the leftside picture there are two x-values for one y-entry value. Which of these values will be returned is uncertain. The similar is true for the x-entry into such a curve. Later implementations of the AVRco LookUp functions support the area limitation for a LookUp so this problem can be handled then.

Generating:

If the curve is completely defined the binary file can be generated. But before this the scaling factors, offsets and the count of the needed curve points must be defined. (**Export** dialog)

The **section** of the curve can be nearly arbitrary. But it makes no sense to set the limits beyond the x-start and end values. In this case only points within the existing x-values of the curve will be exported. If the section is too small propably the interpolation results in inaccurate values.

The offset and scaling factors must be defined in a way that later on the program generates usable values. If for example the input value is read from a 10bit AD-converter, the result from ((curve-y-val – offset) * factor) should be within 0..1023. Similar is true for the x-value. The values of **offset** and **factor** strongly depend of the total gain of the system. This means the gain from sensor to OP-amps upto the AD-converter and its resolution.

The count of points to generate is defined by the dialog field **point count**. Here a compromise is necessary. As an ideal a huge count of points eliminates (or nearly does so) the interpolation and therefore reduces the errors found in the linear interpolation. But this results in huge table with an according long search time. How much points are needed can be found out only with tests with the hardware. An upper limit is the resolution of the data gathering system, eg. AD-converter. With 10bits there are only upto 1024 possibly entry values.

Basically there is one question when the precision has to be checked: which part of the system is the most unprecise one. In most cases the sensor is that one. If this device has large tolerances it makes no sense to trim the software precision to 0.1%.