### Circular controls

Sometimes, it may happen that you need elements for your GUI but you can’t find an appropriate component in the Nextion Editor. That’s the moment to remember  what you’ve already read multiple times in this Sunday Blog over the last two years: “If you don’t have an appropriate component at hands, build it yourself!” And that’s what we’ll do today. In this first step, we want a clock or rotary potentiometer like component which, besides displaying a numeric set value, does also display it for more ergonomic visualization with a spot on an almost full circle.

In a first step, I decided to use a picture component as a “container” for what I’m intending to do. This allowed me to save memory and Nextion CPU resources by packing the clear blue arc and the (-) and (+) buttons in a simple jpg which I created with the open source Photoshop clone “The Gimp”. In our example, I choose to build a room temperature “setter” with temperatures going from 17°C to 27°C. There are thus 11 possible values plus the empty position an the bottom which allows to place them similar to the hours of a clock in 30° steps on the circle which greatly simplifies all trigonometric considerations.

Graphical user interface

Description automatically generated

#### The mathematics

Seen from the center of our circle which as an outer radius of 80px and is 8px thick, the center of the spot to display has to move on a circle with a radius of 76px. The 11 y values are thus cos(30°), cos(60°), cos(90°), cos(120°), …, cos(300°), cos(330°) considering that 0° is the empty position at the bottom. Computed, multiplied with the 76px radius, and rounded I got 66, 38, 0, -38, -66, -76, -66, -38, 0, 38, 66 as values to add to the center y coordinate for the 11 different spot positions.  
The 11 x-values are -sin(30°), -sin(60°), -sin(90°), -sin(120°), …, -sin(300°), -sin(330°). Again, took the pocket calculator and got -38, -66, -76, -66, -38, 0, 38, 66, 76, 66, 38 as values to add to the center x coordinate for the 11 different spot positions.

Now, I could have used trigonometric identities like sin(-x) = -sin(x), or sin(x) = cos(90°-x) to shorten the above number series at the price of more complex computations in Nextion language, but I decided to keep things simple and safe. Since the sin() and cos() functions are basically identical with a 90° phase shift, we see that our number series are identical when we shift the second by 3 positions (3 \* 30° = 90°). Thus, an array with 14 elements [66, 38, 0, -38, -66, -76, -66, -38, 0, 38, 66, 76, 66, 38] allows us to retrieve the 11 y values with the index going from 0 to 10 and the 11 corresponding x values with the index going from 3 to 13.

##### Looking a few weeks back to the array trick

A few weeks ago, we learnt in this Sunday Blog that we’ve to use a little trick because there are no numeric arrays available: We pack our values into a Text variable component using a separator and can then use the ***spstr*** function to extract a value by its index into another Text variable component and use the ***covx*** function to retrieve the numeric equivalent.

The rest is simple. We need the center coordinates of our blue circle segment which are half of the width of our picture plus its .x attribute for x and half of the height plus its .y attribute for y. To which we add the values which we retrieve form our look-up array to obtain the center coordinates of our spot to draw with the ***cirs*** function.

#### Coding everything

In program.s, we declare the required variables as follows, setting our maximum array index to 10 because we have 11 positions, and an offset of 17 to translate the index values from 0 to 10 into a displayed temperature of 17°C to 27°C and vice-versa:

##### program.s

//Declare the pseudo-component attributes and helpers:

int c\_x\_cen, c\_y\_cen,c\_x\_plot,c\_y\_plot,c\_intval,c\_ulimit

int c\_r\_plot=8

int c\_d\_offs=17

int c\_maxval=10

c\_ulimit=c\_d\_offs+c\_maxval

page 0 //Power on start page 0

Then, we place two 20px \* 20px hotspots above the visual (-) and (+) buttons to decrease or to increase the temperature setting, each with a little if() clause to prevent going out of range.

##### m0 Touch Press event

if(n0.val>c\_d\_offs)

{

n0.val--

}

##### and m1 Touch Press event

if(n0.val<c\_ulimit)

{

n0.val++

}

Finally, we need some lines of code to take the temperature value in n0, to detect the position index for the corresponding spot, calculate its drawing coordinates, remove the previously drawer spot and draw the new one. This is done in the m1 (+) button Touch Release event. To double this function for the m0 (-) button without maintaining the identical code twice, the m0 Touch Release event calls simply ***click*** ***m1,0***. This separation allows also to set the temperature display from an external MCU (Arduino) by sending “n0.val=23ÿÿÿ” and “click m1,0ÿÿÿ”.

##### m1 Touch Release event

//Compute the array index

c\_intval=n0.val-c\_d\_offs

//Retrieve and convert rel. y-coordinate and add center coordinate

spstr lut.txt,lval.txt,";",c\_intval

covx lval.txt,c\_y\_plot,0,0

c\_y\_plot+=c\_y\_cen

//Retrieve and convert rel. x-coordinate and add center coordinate

spstr lut.txt,lval.txt,";",c\_intval+3

covx lval.txt,c\_x\_plot,0,0

c\_x\_plot+=c\_x\_cen

//Refresh the page to clear the old spot

ref page0

doevents

//Draw the new spot at the calculated coordinates

cirs c\_x\_plot,c\_y\_plot,c\_r\_plot,65535

I suggest that you download the project code and resources here [sblog220410.HMI](https://cdn.nextion.tech/wp-content/uploads/2022/04/sblog220410.HMI_.zip) and you play with it in the Nextion Editor and Simulator to better understand everything although the code is short and simple. In one of my next articles, we will still add touch functionality, so that you will also be able set the temperature by touching a specific point on the circle. But making a fully functional circular slider requires some more tricks. Stay tuned!

#### How to add touch functionality (and how not to do it)

A picture containing text, light

Description automatically generatedNow, let’s move on: To make our “custom control” more interactive, which means allowing to set the temperature (in our example project) not only with the (+) and (-) buttons, but also by touching the corresponding position on the circle, a few arbitrations are necessary. A “pure” mathematician would probably try to solve this by working with reverse sine and cosine to detect an angle, round it to fixed degree steps and then, by further scaling and offsetting get the required integer value. Additional care would be needed for the radius, since you can’t expect your user to touch the circle exactly in a defined distance from the center. I rejected this approach for being too complicated to be efficiently realized in Nextion language.

Following the “keep it simple and safe” principle, I decided for a touch raster approach. That means, that I imagine a grid laid over my control as in the picture above. The trick is to optimize the number of squares in a way that none of the squares contains more than a single target dot, and, at the same time, to minimize the number of squares to keep them big enough to be easily hit by touch events. Thus I ended up with a 5 x 5 grid

The next step could be placing Touch hotspot components in the place of the squares containing a target dot. Rejected, because of the lack of flexibility!

#### Flexibility first!

Again, when modifying the number of target dots, you’d have to resize and move all hotspots before adding new ones. I decided to take the real time touch coordinates (***tch0*** for x and ***tch1*** for y), subtract the respective x and y coordinates of our component to get coordinates relative tou the upper left corner. Then, since we have 5 squares in each row or column, I’ll divide these relative coordinates by a fifth of the height and width of the component to get each time values between 0 and 4. Multiplying the y-value by 5 and adding the x-value gives me a result between 0 and 24, a kind of field index, where 0 is in the upper left corner, 4 the upper right, 20 the lower left and 24 the lower right corners.

#### Using known techniques

The last step is to find a way to get the corresponding integer value for each square or cell containing a target spot, and a common “out of range” value for all other cells without interest. Again, as last week, we build a very small pseudo-array with 25 elements in a string, so that the ***spstr*** function, called with the field index variable returns either a result between 0 and 10 for the fields containing one of the target spots, and -1 for all others. I called the containing text variable ***rlt*** (reverse lookup table) because the common ***lut*** (look up table) is already used for the drawing coordinates. Its .txt attribute looks thus like this:  
“-1;4;5;6;-1;3;-1;-1;-1;7;2;-1;-1;-1;8;1;-1;-1;-1;9;-1;0;-1;10;-1” and everything is done. The rest is trivial as you can see looking at the code.

#### Declaring additional variables

The few required additional variables are declared in a single line (bold) in program.s:

//Declare the pseudo-component attributes and helpers:

int c\_x\_cen, c\_y\_cen,c\_x\_plot,c\_y\_plot,c\_intval,c\_ulimit

int c\_r\_plot=8

int c\_d\_offs=17

int c\_maxval=10

**int t\_scale,t\_x\_rel,t\_y\_rel,fld\_idx**

c\_ulimit=c\_d\_offs+c\_maxval

page 0 //Power on start page 0

#### Initializing the touch scale factor

The touch scale factor is required to scale down the relative touch coordinates from [0..169] to [0…4] to fit our 5 x 5 grid. This is done with a single additional line in the PostInitialize event of the page (again, in bold):

c\_x\_cen=p0.w/2+p0.x

c\_y\_cen=p0.h/2+p0.y

**t\_scale=p0.w/5**

click m1,0

#### Handling the touch event

is done in the Touch Press event of our container component p0:

// transform touch x coordinate relative to the upper left corner

t\_x\_rel=tch0-p0.x

// scale into rough steps from 0 to 4

t\_x\_rel/=t\_scale

// transform touch y coordinate relative to the upper left corner

t\_y\_rel=tch1-p0.y

// scale into rough steps from 0 to 4

t\_y\_rel/=t\_scale

// calculate the field index from 0 (upper left) to 24 (lower right)

fld\_idx=5\*t\_y\_rel+t\_x\_rel

// retrieve the corresponding integer into c\_intval

spstr rlt.txt,lval.txt,";",fld\_idx

covx lval.txt,c\_intval,0,0

// if there is a target spot, update the temperature display, numeric and circular

if(c\_intval>=0)

{

n0.val=c\_intval+c\_d\_offs

click m1,0

}

And that’s it! These few additional lines improve greatly our “custom control” which we created last week, still following the principle “If you can’t find a suitable control in the Nextion Editor, simply build it yourself”.

Play with it in the Editor and Simulator, upload it on a Nextion HMI, or re-use and adapt this in your own projects, the full code is available here: [sblog220417.HMI](https://cdn.nextion.tech/wp-content/uploads/2022/04/sblog220417.HMI_.zip)